به نام خدا

آموزش سریع Regular Expression

نسخه ۱/۰ (بتا)

گردآوری و تألیف

علی قنواتیان

www.aqlinux.ir

تابستان ۱۳۸۹

فهرست

مقدمه

بعد از جستجو در اینترنت متوجه شدم که ظاهراً راهنمای فارسی برای regular expression نیست. پس تصمیم گرفتم این راهنما رو بنویسم و تقدیمش کنم به همه‌ی اعضای لاگ خوزستان. امیدوارم که مفید واقع بشه.

چه کسانی به این راهنما نیاز دارند؟
کاربران گنو/لینوکس دیر یا زود به یادگیری این استاندارد نیاز پیدا خواهند کرد. علاوه بر کاربران گنو/لینوکس، برنامه‌نویسانی که به یکی از زبان‌های Perl, PHP, Java و یا یکی از زبان‌های .NET. کد می‌نویسند می‌توانند به خوبی از مزایای یادگیری این استاندارد در کار با متن‌ها کمک بگیرند.

پیشنیازهای این راهنما چیست؟
این راهنما برای آن دسته از کاربران گنو/لینوکس نوشته شده که آشنایی مختصری با ترمینال (bash) دارند. اگر با این محیط آشنایی ندارید، بهتر است یک راهنمای سریع در این رابطه مطالعه کنید. در صورتی که کاربر گنو/لینوکس نیستید می‌توانید مثال‌ها را با مراجعه به مستندات زبان برنامه‌نویسی مورد نظرتان، اجرا کنید.

چطور باید این راهنما خوانده شود؟
این راهنما به دو بخش تقسیم شده، بخش اول بر اجرای دستورات و کار عملی متمرکز است و بیشتر برای کاربران مبتدی نوشته شده که قبلاً هیچ آشنایی با مفهوم Regular Expression نداشته و یا تجربه‌ای در کار با آن ندارند. لازم است کاربران تازه‌کار حتماً بخش اول را به طور کامل مطالعه و تمرین کنند تا آشنایی و تجربه‌ی مختصری با مفهوم «عبارات باقاعده» پیدا کنند. بخش دوم مباحث بخش اول را نیز پوشش می‌دهد ولی تمرینات عملی در آن وجود ندارد، این بخش برای یادگیری عمقی‌تر «عبارت‌های باقاعده» توصیه می‌شود. بنابراین کاربران تازه‌کار در صورتی‌که در درک بعضی الگوها ابهام داشتند، می‌توانند به بخش دوم بروند و جواب سؤال خود را در آنجا جستجو کنند.

Regular expression چیست؟
یک «عبارت باقاعده» (regular expression) یا به اختصار regex، الگویی است که بخش معینی از متن را مشخص می‌کند.

«عبارت با قاعده» به بیان ساده
یک عبارت باقاعده، که به الگو هم معروف است، عبارتی است که مجموعه‌ای از رشته‌های کاراکتری را توصیف می‌کند. عبارت باقاعده معمولاً توصیف دقیقی از یک مجموعه است، بدون اینکه نیاز باشد همه‌ی اعضای آن مجموعه ذکر شود. به عنوان مثال، مجموعه‌ای که شامل هر سه رشته متنی "Handel", "Haendel" و "Händel" باشد می‌تواند به کمک یک الگو چنین توصیف شود
H(ä|ae?)ndel

بخش اول

آماده کردن محیط

یک توصیه پیش از شروع؛ برای یادگیری regex بهتر است، همگام با این راهنما دستورات را در ترمینال لینوکس وارد کنید و نتایج را از نزدیک ببینین. برای شروع فایل حاوی متن نمونه را دانلود کرده و در مسیر مناسبی ذخیره کنید یا خطوط زیر را کپی کرده و در یک فایل متنی به اسم demofile.txt ذخیره کنید. من برای دیدن خروجی «عبارات با قاعده» از ابزار grep استفاده خواهم کرد (برای اطلاعات بیشتر در مورد این ابزار، به راهنمای grep مراجعه کنید).

```
Hello World.
This is vivek from Poona.

I love linux.
It is different from all other Os

My brother Vikrant also loves linux who also loves unix.
He currently learn linux.
Linux is cooool.

Linux is now 10 years old.
Next year linux will be 11 year old.

Rani my sister never uses Linux
She only loves to play games and nothing else.

Do you know?
. (DOT) is special command of linux.

Okay! I will stop.
```

اولین اجرا

بدون اتلاف وقت، یک ترمینال باز کرده و به مسیری که فایل را در آنجا ذخیره کرده‌اید بروید.
حاضرید؟ خیلی خب برای شروع دستور زیر را وارد می‌کنم:

```
$ grep 'linux' demofile.txt
```

خروجی این دستور به این شکل خواهد بود:

I love linux.
My brother Vikrant also loves linux who also loves unix.
He currently learn linux.
Next year linux will be 11 year old.
. (DOT) is special command of linux.

این خروجی شامل تمام خطوطی است که حاوی linux بوده‌اند.

حالا این دستور را اجرا می‌کنم.

```
$ grep 'is' demofile.txt
```

در خروجی می‌آید:

This is vivek from Poona.

It is different from all other Os
Linux is cooool.
Linux is now 10 years old.
Rani my sister never uses Linux
. (DOT) is special command of linux.

همانطور که می‌بینید خط Rani my sister never uses Linux نیز به دلیل دارا بودن is در sister در خروجی آمده است. پس:

موتور جستجوگر «عبارت باقاعده» عملیات جستجو را کاراکتر به کاراکتر انجام می‌دهد و معمولاً جستجو را با رسیدن به اولین عبارت مطابق با الگو متوقف می‌کند مگر اینکه ادامه یافتن جستجو در سراسر متن از او خواسته شده باشد.

نکته: در دستور grep به صورت پیش‌فرض، جستجو در سراسر متن ادامه پیدا می‌کند.

جستجوی محدود

دستور زیر همان عملیات جستجوی بالا را پس از اولین خط انطباق متوقف می‌کند.

```
$ grep 'is' -m 1 demofile.txt
This is vivek from Poona.
```

برای اطلاعات بیشتر در خصوص پارامترها و سایر امکانات grep راهنمای این دستور را مطالعه نمایید.

حساسیت به حروف بزرگ و کوچک

بدیهی است که «عبارت باقاعده» به حروف کوچک و بزرگ حساس است، به عنوان مثال خروجی زیر را ببینید:

```
$grep 'Linux' demofile.txt
Linux is cooool.
Linux is now 10 years old.
Rani my sister never uses Linux
```

همانطور که دیده می‌شود خطوط حاوی linux به دلیل اینکه از نظر بزرگ و کوچک بودن حروف در کاراکتر L با الگوی Linux تطبیق نداشتند در خروجی نیامده.

ابتدای خط یا انتهای خط

برای جستجو کردن عبارتی که در ابتدای خط بیاید کافی است قبل از آن یک کاراکتر ^ درج کنم و برای رسیدن به عبارتی که در انتهای خط آمده باشد کافی است بعد از آن یک کاراکتر $ اضافه کنم.
به عنوان مثال دستور زیر را وارد می‌کنم:

```
$ grep '^Linux' demofile.txt
```

و جواب می‌گیرم

```
Linux is cooool.
Linux is now 10 years old.
```

این همه‌ی خطوطی است که با Linux شروع شده اند، و همچنین نتیجه‌ی این دستور

```
$ grep 'Linux$' demofile.txt
Rani my sister never uses Linux
```

همه‌ی خطوطی است که در انتهایشان Linux آمده است. خب بر همین اساس می‌توانم حدس بزنم که برای پیدا کردن خطوط خالی در یک متن می‌توانم از چه عبارت با قاعده‌ای کمک بگیرم...

بد نیست قبل از ادامه کمی به آن فکر کنید...
...
..
خیلی خب. این است: '\(<\|>\)\[^<>]+<

نه این فقط یک شوخی بود! برای خندان شما. جواب درست اینجاست:
^$
توصیف جالبی برای یک خط خالی است، خطی که با کارکتر انتهای خط شروع شود!

براکت برای محدوده‌ی کاراکترها

برای پیدا کردن محدوده‌ای از کاراکترها در یک متن از [] استفاده می‌شود. به عنوان مثال برای پیدا کردن Linux و linux در متن وارد می‌کنم:

```
$ grep '[Ll]inux'
I love linux.
My brother Vikrant also loves linux who also loves unix.
He currently learn linux.
Linux is cooool.
Linux is now 10 years old.
Next year linux will be 11 year old.
Rani my sister never uses Linux
. (DOT) is special command of linux.
```

خروجی این دستور خطوطی است حاوی linux و Linux بوده‌اند. در علامت [] می‌توان محدوده‌ای از کاراکترها را نیز مشخص کرد، به عنوان مثال برای رسیدن به خطوطی که حاوی عدد هستند، می‌توان به جای نوشتن [0123456789]، چنین نوشت [0-9]. این الگو را خودتان روی متن امتحان کنید.

خب اینجا سعی می‌کنم فایل را بدون خطوط خالی نمایش دهم، پس وارد می‌کنم:

```
$ grep '[^^$]' demofile.txt
```

و در جواب می‌آید

```
Hello World.
This is vivek from Poona.
I love linux.
It is different from all other Os
My brother Vikrant also loves linux who also loves unix.
He currently learn linux.
Linux is cooool.
Linux is now 10 years old.
Next year linux will be 11 year old.
Rani my sister never uses Linux
She only loves to play games and nothing else.
Do you know?
. (DOT) is special command of linux.
Okay! I will stop.
```

ولی چه اتفاقی افتاد؟
در این دستور از یک [^] استفاده کردم که مشخص کننده‌ی محدوده‌ای از کاراکترهاست که نمی‌بایست در متن باشند. برای توضیح بیشتر الگوی [^abc] را در نظر بگیرید. شاید گمان کنید خروجی این الگو متنی است که هرگز حاوی a یا b یا c نباشد! ولی درواقع چنین است که این الگو

خطوطی را نمایش می‌دهد که **حاوی کاراکترهایی غیر از کاراکترهای a یا b یا c باشند**.
پس اگر این الگو را روی این متن وارد کنید:

abccbc
adc
ghg

در خروجی می‌آید

adc
ghg

خط اول حذف شده است ولی خط دوم به دلیل اینکه یک کاراکتر به جز a، b و c داشت (مشخصاً d) در خروجی آمد.

نکته: در استفاده از sed یا ex برای حذف شدن خطوط خالی از خروجی، می‌بایست از الگوی زیر استفاده شود.

[^/^$]

استفاده از دسته کاراکترهای استاندارد ***POSIX***

یک خبر خوب اینکه در regular expression دسته کاراکترهای استاندارد POSIX وجود دارد که می‌توان از آن‌ها کمک گرفت. این جدول در بخش دوم در قسمت «کلاس‌های کاراکتری POSIX» منتظر دیدار شماست! با توجه به این جدول، برای دیدن خطوطی که از علایم نگارشی در آن‌ها استفاده شده، وارد می‌کنم

```
$ grep '[[:punct:]]' demofile.txt
Hello World.
This is vivek from Poona.
I love linux.
My brother Vikrant also loves linux who also loves unix.
He currently learn linux.
Linux is cooool.
Linux is now 10 years old.
Next year linux will be 11 year old.
She only loves to play games and nothing else.
Do you know?
. (DOT) is special command of linux.
Okay! I will stop.
```

به وضوح مشخص است که خطوط خالی و دو خطی که علایم نگارشی در آن‌ها نبود، در خروجی ظاهر نشده‌اند.

نکته ۱: برای استفاده از دسته کاراکترهای استاندارد POSIX می‌بایست آن‌ها را درون [] قرار داد.
به عنوان مثال برای حروف بزرگ می‌بایست وارد شود [[:upper:]]

اما اگر در مثال قبل بخواهم خطوطی که علایم نگارشی ندارند را ببینم، وارد می‌کنم:

```
$ grep '[^[:punct:]] demofile.txt
Hello World.
This is vivek from Poona.
I love linux.
It is different from all other Os
My brother Vikrant also loves linux who also loves unix.
```

He currently learn linux.
Linux is cooool.
Linux is now 10 years old.
Next year linux will be 11 year old.
Rani my sister never uses Linux
She only loves to play games and nothing else.
Do you know?
. (DOT) is special command of linux.
Okay! I will stop.

می‌بینین؟ این منطق کامپیوتر است!! همه‌ی خطوطی که حتی یک کاراکتر غیر نگارشی در آن‌ها استفاده شده بود در خروجی آمد. پس در استفاده از regular expression ها، به منطق چیزی که نوشته می‌شود از دید کامپیوتر باید توجه کرد! اما برای رسیدن به خطوطی که هیچ علایم نگارشی در آن‌ها استفاده نشده باشد به اطلاعات بیشتری نیاز است. پس بعد از مطرح کردن پیش‌نیازها به بررسی آن می‌پردازم.

جستجو به دنبال کلمه

گاهی ممکن است دنبال یک کلمه در متن باشیم (حالتی که قبل و بعد از عبارت مورد نظر یک کاراکتر جدا کننده (یعنی هر چیزی بجز حرف و عدد) باشد)، مثال مربوط به is را به خاطر دارید؟ در آن مثال خط حاوی sister هم در خروجی آمده بود. برای اینکه فقط کلمه‌ی is در خروجی بیاید وارد می‌کنم:

```
$ grep '\<is\>' demofile.txt
```

و در خروجی می‌آید

This is vivek from Poona.
It is different from all other Os
Linux is cooool.
Linux is now 10 years old.
. (DOT) is special command of linux.

پس الگوی <\>\ مربوط به جستجوی یک کلمه در متن است.

نقطه (.) و ستاره (*) در ***regular expression***

نقطه معادل هر یک کاراکتری است که می‌تواند هر چیزی باشد. مثلاً اگر دنبال کلمات دو حرفی باشم که با حرف o خاتمه پیدا کنند مثل so یا go یا ... می‌بایست وارد کنم:

```
$ grep '\<.o\>' demofile.txt
```

She only loves to play games and nothing else.
Do you know?

البته grep خطوط حاوی این کلمات را برمی‌گرداند، برای برگرداندن خود کلمات می‌بایست از پارامتر o- استفاده شود.
ستاره نیز، معادل صفر یا هر تعداد کاراکتری است که قبل از آن آمده باشد. به عنوان مثال اگر دنبال کلماتی باشم که با حرف c یا C شروع می‌شوند و تعداد حروفشان برایم اهمیتی نداشته باشد، وارد می‌کنم:

```
$ grep '\<[cC][[:alpha:]]*\>'
```

He currently learn linux.
Linux is cooool.
. (DOT) is special command of linux.

مثال خوبی بود نه؟ حالا اطلاعات کافی برای رسیدن به خطوطی که هیچ علایم نگارشی ندارند در اختیار دارید. کافی است آن‌ها را کنار هم بگذارید. اگر احساس می‌کنید هنوز نمی‌توانید یا مشکلی دارید، چند الگوی ساده برای خودتان آماده کنید و سعی کنید آن‌ها را انجام دهید. این دقیقاً مثل

برنامه‌نویسی است، تا وقتی خودتان دست به کار نشده باشید، چیزی یاد نخواهید گرفت! ولی وقتی شروع کردید، کم کم عاشقش می‌شوید (تضمین می‌کنم ;))

Escaping – وقتی که می‌بایست گریخت

حالا بیاید تصور کنیم، می‌خواهم همه‌ی خطوطی که با نقطه شروع می‌شوند را ببینم.

```
$ grep '^.'
Hello World.
This is vivek from Poona.
I love linux.
It is different from all other Os
My brother Vikrant also loves linux who also loves unix.
He currently learn linux.
Linux is cooool.
Linux is now 10 years old.
Next year linux will be 11 year old.
Rani my sister never uses Linux
She only loves to play games and nothing else.
Do you know?
. (DOT) is special command of linux.
Okay! I will stop.
```

نشد! این خروجی حاصل شد به خاطر اینکه نقطه یک کاراکتر رزرو شده برای Regex است، از این خروجی مشخص است که نقطه معرف هر یک کارکتری است که ممکن است هر چیزی باشد. برای گرفتن خروجی مورد نظرم می‌بایست به موتور پردازشگر regular expression بگویم که این یک نقطه است نه هیچ چیز دیگر. پس از یک بک اسلش "\" قبل از آن استفاده می‌کنم.

```
$ grep '^\.'
. (DOT) is special command of linux.
```

بسیار خب؟ این تکنیک در برنامه‌نویسی به escaping «گریز» معروف است. می‌توانید قرار دادن \ قبل از کاراکترهای رزرو شده را هر کجا لازم باشد به کار ببرید.

چند پیشنهاد

خب حالا شما با اصول اولیه‌ی regular expression آشنایی پیدا کرده‌اید، هنوز مطالب زیادی برای یادگیری وجود دارد و البته برای تسلط پیدا کردن بر «عبارت‌های باقاعده» بیشتر از هر چیزی به تمرین نیاز خواهید داشت. پیشنهاد من این است که برای خودتان سناریوهایی تعریف کنید و آن‌ها را به انجام برسانید این به شما تجربه و اعتماد به نفس خوبی می‌دهد.
و در ضمن فراموش نکنید که چندین روش برای رسیدن به یک نتیجه وجود دارد و این کاملاً به سلیقه‌ی شما برمی‌گردد که از چه الگویی استفاده کنید و قبل از اینکه فراموش کنم! بخش دوم این راهنما به شما اطلاعات جامع و مفیدی در خصوص کاربرد «عبارت‌های باقاعده» می‌دهد، مطالعه و تمرین کردن آن را فراموش نکنید!

و یک پیشنهاد دیگر، بد نیست پوستر آموزشی regular expression را تهیه کنید و روی دیوار، جایی که در دیدتان باشد قرار دهید.

بخش دوم

مقدمه

عبارات باقاعده در بسیاری از ویرایشگرهای متنی، ابزارهای سودمند (utility)، و زبان‌های برنامه‌نویسی برای جستجو و پردازش متن، به کار گرفته می‌شوند. به عنوان مثال، بیشتر زبان‌های برنامه‌نویسی مثل Perl, Ruby, Awk و Tcl موتورهای پردازش regular expression دارند که ذاتا با گرامر این زبان‌ها تلفیق شده است. بد نیست بدانید در ابتدا ابزارهای سودمند توزیع کنندگان یونیکس -مثل ویرایشگر ed و ابزار فیلتر grep- باعث رواج یافتن مفهوم regular expression شدند.
به عنوان یک مثال از عبارت باقاعده، عبارت باقاعده‌ی \bex برای جستجوی همه‌ی نمونه رشته‌های "ex"ای که در «محدوده‌ی کلمه» باشند (به خاطر وجود \b) استفاده می‌شود. به بیان ساده‌تر، \bex با عبارت ex در دو حالت تطبیق می‌کند، ۱) در حالتی که در ابتدای کلمه‌ای آمده باشد و ۲) حالتی که بعد از یک کاراکتر جداکننده‌ی کلمه (مثل علایم نگارشی یا ...) آمده باشد (کاراکتر(های) بعد از عبارت ex اهمیتی ندارد.) پس، در عبارت «Texts for experts»، عبارت باقاعده \bex با ex در expert تطبیق می‌کند ولی با Texts نه. (چون ex در یک کلمه قرار دارد ولی بلافاصله بعد از مرز کلمه (در اینجا کاراکتر فاصله جدا کننده‌ی کلمه است) نیامده)
اکثر کامپیوترهای امروزی از کاراکترهای wildcard در یافتن اسامی فایل‌ها پشتیبانی می‌کنند. این یکی از ویژگی‌های پایه‌ای اکثر ترمینال‌های خط فرمان است و با عنوان globbing شناخته می‌شود. Wildcardها در واقع نوع بسیار محدودی از regular expressionها هستند و با آن‌ها فرق می‌کنند.

امکانات عبارت‌های باقاعده

بیشتر پیاده‌سازی‌های موجود از موتور عبارت‌های باقاعده، امکانات زیر را برای ساخت یک عبارات باقاعده (الگو) ارائه می‌کنند.

«یا»ی منطقی

یک خط عمودی که جایگزین‌ها را از هم جدا می‌کنند. به عنوان مثال الگوی `gray|grey` هم با `gray` و هم با `grey` تطبیق می‌کند.

گروه بندی

پرانتزها برای تعریف گروه‌ها و محدوده‌ها به کار می‌روند. به عنوان مثال الگوهای gray|grey و gr(a|e)y معادل هم هستند.

کمیت‌سنج

یک کمیت‌سنج بعد از یک قطعه (token) (مثل یک کاراکتر) یا یک گروه می‌آید و مشخص می‌کند که آیتم قبلی تا چند بار مجاز به تکرار است. متداولترین کمیت‌سنج‌ها علامت سؤال ?، ستاره * و علامت به‌علاوه + هستند.

نماد	عملکرد
?	علامت سوال، یعنی آیتم قبلی صفر یا یک بار تکرار شده باشد. به عنوان مثال colou?r هم با color و هم با colour تطبیق می‌کند.
*	ستاره، یعنی آیتم قبلی صفر بار یا بیشتر تکرار شده باشد. به عنوان مثال ab*c با ac, abc, abbc, abbbc و ... تطبیق می‌کند.
+	به‌علاوه، یعنی آیتم قبلی یک بار یا بیشتر تکرار شده باشد. به عنوان مثال ab+c با abc, abbc, abbbc و ... تطبیق می‌کند ولی نه با ac .

جدول کمیت‌سنج‌ها

از ترکیب این کمیت‌سنج‌ها می‌توان برای ساخت عبارات پیچیده کمک گرفت. درست مثل ساختن عبارت‌های محاسباتی در ریاضیات، که از اعداد و عملگرهای جمع و ضرب و تفریق و تقسیم حاصل می‌شوند. به عنوان مثال H(ae?|a)ndel و H(a|ae|a)ndel هر دو الگوهایی معتبر هستند و رشته‌های ” Handel”, “Haendel” و ”Händel” را برمی‌گردانند.

همانطور که در تعاریف استاندارد عبارت باقاعده آمده، برای پرهیز از پرانتز گذاری، اولویت بندی خاصی لحاظ شده است: ستاره بالاترین اولویت را دارد، پس از آن کنار هم قرار گیری و بعد از آن «یا»ی منطقی.

اولویت	عملگر	عنوان عملگر	مثال
1	*	ستاره کلین	ab*
2	ندارد	الحاق	ab
3	|	جایگزینی	a|b

مثال‌ها

الگوی نمونه	خروجی الگو
a|b*	a, b, bb, bbb, ...
(a|b)*	a,b, aa, ab, ba, bb, aaa,...
ab*(c|)	a, ac, ab, abc, abb, abbc,...

گرامر ***POSIX***[1]

عموم مفاد این استاندارد است که گرامر عبارت باقاعده‌ی یونیکس را تشکیل می‌دهد. با این حال در ابزارهای یونیکسی تفاوت‌های گرامری وجود دارد. استاندارد (IEEE POSIX Basic Regular Expressions (BRE که در کنار جایگزینی به نام Extended Regular Expressions یا ERE عرضه شد، بیشتر به خاطر حفظ سازگاری با گرامر استاندارد سنتی Simple Regular Expression طراحی شده بود. اما بعدها این استاندارد میان بسیاری از ابزارهای یونیکسی مرتبط با عبارت‌های باقاعده رایج گردید. این ابزارها از این استاندارد به عنوان گرامر پیش‌فرض کمک گرفتند. با وجود این، هنوز در این ابزارها تفاوت‌ها و ویژگی‌های اضافه‌ای نیز وجود دارد. بیشتر این ابزارها پشتیبانی از گرامر ERE را از طریق آرگومان‌های خط فرمان فراهم می‌کنند.

در گرامر BRE، بیشتر کاراکترها به صورت معادل خودشان شناخته می‌شوند. - یعنی آن‌ها فقط با معادل خودشان تطبیق می‌کنند (مثلاً یک a با "a" مطابقت می‌کند). استثناها در زیر لیست شده‌اند و به عنوان متاکاراکتر یا metasequences شناخته می‌شوند.

متاکاراکتر	توضیحات
.	معادل هر یک کاراکتر نامشخص (در بسیاری برنامه‌ها این شامل newlineها نمی‌شود، و اینکه چه کاراکترهایی معادل newline باشند کاملاً سلیقه‌ای، وابسته به encoding و وابسته به پلت‌فرم است، اما ایرادی ندارد اگر بگوییم کاراکتر line feed همیشه جزو آن‌هاست. در یک عبارت POSIX که بین براکت باشد، نقطه با خودش "." تطبیق می‌کند. به عنوان مثال، a.c با "abc" و ... تطبیق می‌کند ولی [a.c] با "."، "a" یا "c" مطابقت می‌کند.
[ ]	یک عبارت براکت‌دار. با هر کاراکتری که بین براکت‌ها باشد تطبیق می‌کند. به عنوان مثال، [abc] با "a"، "b" یا "c" تطبیق می‌کند. [a-z] محدوده‌ی حروف کوچک از a تا z را مشخص می‌کند. این دو فرم می‌توانند ترکیب شوند: [abcx-z] با "a","b","c","x","y" یا "z" تطبیق می‌کند یا به شکل دیگر: [a-cx-z]. در براکت اگر کاراکتر - در آخر یا اول (بعد از [) بیاید، به عنوان خود کاراکتر - در نظر گرفته می‌شود: [abc-],[-abc]. نکته اینجاست که برای این کاراکتر استفاده از یک اسلش "\" مجاز نیست. کاراکتر [می‌تواند داخل یک عبارت براکتی باشد به شرطی که بلافاصله بعد از شروع براکت بیاید. مثل: []abc]
[^]	کاراکتری که لازم است در متن مورد نظر نباشد. به عنوان مثال، الگوی [^abc] با هر کاراکتری به جز a, b یا c تطبیق می‌کند. الگوی [^a-z] با هر کاراکتری که جزو حروف کوچک انگلیسی نباشد تطبیق می‌کند. مثل مورد بالایی کاراکترهای تکی و محدوده‌ها می‌توانند ترکیب شوند.
^	مشخص کننده‌ی موقعیت شروع رشته متنی. در ابزارهای پردازش متن خطی، این با شروع هر خط مطابقت می‌کند.
$	مشخص کننده‌ی موقعیت پایان رشته یا موقعیت درست قبل از رشته کاراکتر خط جدید یا new line است. در ابزارهای پردازش متن خطی با موقعیت انتهای هر خط تطبیق می‌کند.
BRE: \(\) **ERE: ( )**	یک زیر الگوی مارک‌دار تعریف می‌کند. رشته‌ای که با زیر الگو مطابقت کند، بعداً می‌تواند فراخوانی شود (ردیف بعدی این جدول را ببینید (\n)). به یک زیر الگوی مارک‌دار، بلوک block یا گروه گیری capturing group نیز می‌گویند.
\n	با زیر الگوی مارک‌دار n ام تطبیق می‌کند، که n عددی بین ۱ تا ۹ است. این ساختار از نظر تئوری نامنظم است و در گرامر POSIX ERE پشتیبانی نشده است. در بعضی ابزارها امکان گروه گیری بیش از ۹ بار نیز وجود دارد.

1- POSIX (Portable Operating System Interface [for Unix])

*	با صفر یا چند بار تکرار آیتم پیشین تطبیق می‌کند. به عنوان مثال، الگوی ab*c با ac یا abc یا abbc و... تطبیق می‌کند. یا *[xyz] با "" ، " "، "x" ، "y" ، "z" ، "zx" ، "zyx" ، "xyzzy"" و غیره تطبیق می‌کند. *(ab)\ با "" ، "abab" ، "ababab" و غیره تطبیق خواهد کرد.
BRE: \ **{m,n\}** **ERE: {m,n}**	با تکرار حداقل m بار آیتم پیشین و نه بیش از n بار تکرار آن تطبیق می‌کند. به عنوان مثال، a\{3,5\} فقط با aaaa ،aaa و aaaaa تطبیق می‌کند. این ویژگی در نسخه‌های قدیمی‌تر عبارت‌های باقاعده یافت نمی‌شود.

مثال:

الگو	توضیحات
.at	با هر رشته‌ی سه کاراکتری که با at پایان یافته باشد. مثل hat, cat و bat.
[hc]at	با hat و cat
[^b]at	همه‌ی رشته‌هایی که مورد اول با آن‌ها تطبیق می‌کند بجز bat .
^[hc]at	با hat و cat اما در حالتی که فقط اول رشته یا خط بیایند.
[hc]at$	با hat و cat اما فقط وقتی که در پایان متن یا خط بیایند.
\[.\]	هر کاراکتری که بین علامت‌های براکت باشد چون از براکت‌ها گریز شده است، مثل [a] یا [b] و...

عبارت‌های باقاعده افزوده‌ی ***POSIX***[2]

در عبارت‌های باقاعده افزوده‌ی POSIX یا ERE معنای بعضی متاکاراکترهایی که با یک اسلش "\" گریز شده باشند برعکس است. در این گرامر (Syntax)، استفاده از یک اسلش باعث می‌شود که با یک متاکاراکتر مثل یک کاراکتر معمولی رفتار شود. بنابراین به عنوان مثال، الگوی \(\) به شکل ( ) و \{ \} به شکل { } تطبیق می‌شود. علاوه بر این، پشتیبانی \n برای حفظ سازگاری با گذشته حذف شده و موارد زیر اضافه شده است:

متاکاراکتر	توضیح
?	آیتم قبلی صفر یا یک بار تکرار شده باشد. به عنوان مثال، ba? با b یا ba تطبیق می‌کند
+	آیتم قبلی یک بار یا بیشتر تکرار شده باشد. به عنوان مثال، ba+ با ba، baa یا baaa و ... تطبیق می‌کند.
|	کاراکتر انتخاب (مثل جایگزینی یا «یا»ی منطقی در مجموعه‌ها) با حالتی که عبارت قبل یا بعد از آن تطبیق کند، مطابقت می‌کند. به عنوان مثال، abc|def با abc یا def تطبیق می‌کند.

مثال:

الگو	تطبیق‌ها
[hc]+at	با hat, cat, hhat, hcat,ccchat و... اما با at تطبیق نخواهد کرد.
[hc]?at	با hat, cat, at
[hc]*at	با hat, cat, hhat, chat, hcat, ccchat, at و ...
cat|dog	فقط با cat یا dog

عبارت‌های باقاعده‌ی افزوده‌ی POSIX را اغلب می‌توان در خط فرمان با پارامتر E- فعال کرد. (البته اگر به صورت پیش‌فرض فعال نباشند)

2- POSIX Extended Regular Expressions

کلاس‌های کاراکتری ***POSIX***

استاندارد POSIX چند دسته کاراکتر طبق جدول زیر تعریف کرده است.

POSIX	Non-standard	Perl	ASCII	توضیحات
[:alnum:]			[A-Za-z0-9]	همه کاراکترهای حرفی و عددی
	[:word:]	\w	[A-Za-z0-9_]	همه کاراکترهای حرفی و عددی به علاوه کاراکتر _
		\W	[^A-Za-z0-9_]	همه‌ی کاراکترها بجز آن‌هایی که در ردیف بالای این جدول مطرح شده‌اند
[:alpha:]			[A-Za-z]	همه حروف کوچک و بزرگ
[:blank:]			[\t]	فاصله و tab
[:cntrl:]			[\x00-\x1F\x7F]	کاراکترهای کنترلی
[:digit:]		\d	[0-9]	اعداد
		\D	[^0-9]	کاراکترهای غیر عددی
[:graph:]			[\x21-\x7E]	کاراکترهای قابل رؤیت
[:lower:]			[a-z]	حروف کوچک
[:print:]			[\x20-\x7E]	کاراکترهای قابل رؤیت و فاصله
[:punct:]			[-!"#$%&'()*+,./:;<=>?@[\\\]^_`{|}~]	کارکترهای نگارشی
[:space:]		\s	[\t\r\n\v\f]	کاراکترهای فضای خالی
		\S	[^ \t\r\n\v\f]	کاراکترهای غیر فضای خالی
[:upper:]			[A-Z]	حروف بزرگ
[:xdigit:]			[A-Fa-f0-0]	اعداد مبنای شانزده (هگزادسیمال)

دسته کاراکترهای POSIX را فقط می‌توان در یک عبارت براکت‌دار استفاده کرد. به عنوان مثال، [[:upper:]ab]
با حروف بزرگی که بعد از آن‌ها حروف کوچک a یا b بیاید تطبیق می‌کند.
در عبارت باقاعده Perl کلاس [:print:] حاصل اجتماع [:graph:] و [:space:] است. بعضی ابزارها دسته کاراکترهای غیر استاندارد (non-POSIX) را نیز درک می‌کنند. مثل [:word:]، که معمولاً به شکل [:alnum:] به اضافه کاراکتر زیرخط _ تعریف می‌شود. این نشان دهنده‌ی این حقیقت است که در بسیاری زبان‌های برنامه‌نویسی این کاراکترها به عنوان شناساگرها[3] شناخته می‌شوند.
ویرایشگر Vim حتی دسته‌بندی word و word-head ساخته است (با استفاده از ترکیب‌های \w و \h).
دلیل این تعریف این بوده که در بسیاری از زبان‌های برنامه‌نویسی کاراکترهایی که می‌توانند یک شناساگر را مشخص کنند مثل بقیه‌ی کاراکترهایی که در هر جای دیگر می‌توانند بیایند نیستند.

3 یک شناساگر در زبان‌های برنامه نویسی می‌تواند نام یک متغییر،تابع، ثابت، شئ یا مواردی از این دست باشد.

به یاد داشته باشید، آنچه که استاندارد عبارت باقاعده‌ی POSIX به آن «دسته‌های کاراکتر» می‌گوید در پیاده‌سازی‌های دیگر عبارت‌های باقاعده که آن‌ها را پشتیبانی کنند به اسم «دسته کاراکترهای POSIX» شناخته می‌شوند.

عبارت‌های باقاعده مشتق شده از ***Perl***

Perl گرامر غنی‌تر و با ثبات‌تری نسبت به استانداردهای POSIX basic (BRE) و عبارت باقاعده افزوده (ERE) دارد. یک نمونه از ثبات این گرامر \ است که همیشه یک کاراکتر غیر الفبایی را گریز می‌کند. مثال دیگری از قدرت کارآیی Perl در مقابل عبارت‌های باقاعده مبتنی بر POSIX مفهوم کمیت‌سنج تنبل (بخش بعد را ببینید) است.
به خاطر قدرت مشخص و گسترده‌ی Perl، بسیاری از ابزارها و زبان‌های برنامه‌نویسی گرامری شبیه گرامر Perl اختیار کرده‌اند. - به عنوان مثال، جاوا، جاوا اسکریپت، PCRE، پایتون، Ruby و پلت‌فرم net. مایکروسافت و همچنین XML Schema مربوط به W3C همه از گرامری شبیه Perl استفاده می‌کنند. بعضی زبان‌ها و ابزارها مثل Boost و PHP از چند عبارت باقاعده پشتیبانی می‌کنند. پیاده‌سازی‌های عبارت باقاعده مشتق شده از Perl همیشه شبیه هم نیستند، و بسیاری از پیاده‌سازی‌ها فقط بخشی از ویژگی‌های Perl را دارند.

کمیت‌سنج تنبل

کمیت‌سنج‌های استاندارد عبارت‌های باقاعده حریص هستند! یعنی تا آنجا که امکان‌پذیر باشد کارکترهای بیشتری تطبیق می‌کنند، و این کار را تا زمانی که بقیه‌ی متن هنوز از الگو خارج شده باشد ادامه می‌دهند. به عنوان مثال برای پیدا کردن اولین آیتم بین < و > در این مثال:
Another Whale sighting occurred on <January 26>, <2004>
کسی که تازه با regex آشنا شده باشد، این الگو <*.> یا چیزی شبیه این پیشنهاد می‌دهد. که البته این به جای ”<January 26>” که مورد انتظار است، چنین چیزی برگشت داده می‌شود
”<January 26>, <2004>”
دلیل این موضوع هم حریص بودن کمیت‌سنج ستاره * می‌باشد. - این کمیت‌سنج هر چقدر که بتواند از ورودی کاراکتر می‌گیرد، واضح است که January 26> <2004 کاراکترهای بیشتری نسبت به January 26 دارد.
این مسأله به چند روش قابل اجتناب است. (مثلاً با مشخص کردن کاراکتری که نمی‌بایست تطبیق شود: <*[<^]>)، ابزارهای عبارت باقاعده‌ی جدید اجازه می‌دهند که یک کمیت‌سنج به شکل تنبل (یا غیر حریصانه، کمینه، ناحریصانه، بی‌میل) تعریف شود به این ترتیب که یک علامت سؤال بعد از کمیت‌سنج درج شود. (مثل <?*.>) با استفاده از یک کمیت‌سنج تنبل، عبارت سعی می‌کند کمترین رشته کاراکتر قابل تطبیق را اعلام کند.

منابع

- Wikipedia, the free encyclopedia
  - http://en.wikipedia.org/wiki/Regular_expression
- Linux Shell Scripting Tutorial v1.05r3 (A Beginner's handbook) , Vivik G. Gite