۳- با دستور cut، User ID ها را جدا کنید.

cut -c برای این کار مفید نمی‌باشد چون هر user ID تعداد حروف متفاوتی با بقیه دارد.

#cut -f1 /etc/passwd

وقتی f1 می‌زنیم یعنی فیلد اول (f1:field 1)

فیلد اول چگونه جدا می‌شود؟ توسط جدا کننده(delimiter)

به‌صورت پیش‌فرض delimiter (جدا کننده)، blank است ولی ما اینجا «:» می‌زنیم (چون در فایل /etc/passwd جداکننده «:» می‌باشد.)

#cut –d: -f1 /etc/passwd

کار با Regex (Regular Expression) توسط awk

در قسمت‌های قبلی sed و cut را معرفی کردیم. در این قسمت، کار با Regex ها را توسط awk به‌صورت مشروح آموزش می‌دهیم.

زبان برنامه‌نویسی awk یک زبان کوچک، با نوشتاری شبیه زبان C و طراحی شده برای پردازش متون مرتب شده با قواعد خاص مانند متون خروجی از بانک‌های اطلاعاتی و فایل‌های ثبت رخداد سیستم می‌باشد. همانند پرل، awk تماما حول regular expression و patern handling شکل گرفته است. اگر بخواهیم دقیق‌تر نگاه کنیم، پرل نواده زبان awk به شمار می‌رود.

نام awk برگرفته شده از نخستین نگارنده‌های این زبان به نام‌های Alfred V.Aho، Brian W. Keringhan و Peter J. Weinberger است. شاید بدانید که Keringhan یکی از پدران زبان C و یکی از قدرت‌ها در دنیای یونیکس است.

استفاده از awk در یک خط

برای شروع آموزش چگونگی استفاده از awk، مثالی در مورد چگونگی چاپ چند فیلد خاص را در یک خروجی ارائه می‌کنیم:

```
$ ls /home/fatemeh | awk '{print $1"\t"$6"\t"$8}'
```

خروجی دستور بالا مشابه زیر خواهد بود:

```
-rw-r--r--      2004-05-28      tsql-banner.gif
-rw-r--r--      2004-05-19      tsql-banner.jpg
-rw-r--r--      2004-05-19      tsql-banner.png
-rw-r--r--      2004-05-19      tsql-banner.xcf
drwxr-xr-x      2004-11-21      tsql-dev
drwxr-xr-x      2004-10-01      tsql_files
drwxr-xr-x      2005-07-05      tsql_guide
-rw-r--r--      2005-07-05      tsql-guide-1.0.1.tar.gz
-rw-r--r--      2005-05-13      tsql-home.html
-rwxrwxrwx      2004-12-25      tsql.html
-rw-r--r--      2004-12-25      tsql-logo-main.png
-rw-r--r--      2004-03-26      tsql-logo-main.xcf
```

همان‌طور که مشاهده می‌کنید، تنها فیلدهای اول، ششم و هشتم بر روی خروجی نمایش داده می‌شوند. این سه فیلد نیز هر یک با یک کاراکتر tab از هم جدا شده‌اند. بعدا خواهید دید که از awk تا چه اندازه‌ای برای نوشتن این‌گونه برنامه‌های یک خطی استفاده می‌شود.

ساختار یک برنامه awk

یکی از ویژگی‌های بسیار خوب awk در کنار قابلیت‌های فراوانش در مقابل پرل و پیتون، راحت خوانده شدن برنامه‌های آن است. هر برنامه awk دارای سه بخش است: یک بلوک BEGIN که قبل از اینکه هر گونه ورودی خوانده شود اجرا می‌شود، یک حلقه اصلی که به ازای هر

خط از ورودی یک‌بار اجرا می‌شود و یک بلوک END که پس از اتمام خواندن ورودی اجرا می‌شود.

در زیر یک برنامه awk نمایش داده شده است که سعی دارد برخی از قابلیت‌های این زبان را بکار ببندد.

```
#!/usr/bin/awk -f
#
# check the sulog for failures..
# copyright 2001 (c) jose nazario
#
# works for Solaris, IRIX and HPUX 10.20
# modified to work with Linux by Fatemeh Baghumian
BEGIN {
  print "--- checking sulog"
  failed=0
}
{
  if ($6 == "-") {
     print "failed su:\t"$7"\tat\t"$1"\t"$2"\t"$3
     failed=failed+1
    }
}
END {
 print "---------------------------------------"
 printf("\ttotal number of records:\t%d\n", NR)
 printf("\ttotal number of failurs:\t%d\n",failed)
}
```

ابتدا اجازه دهید تا نگاهی به فرمت ورودی بیندازیم:

```
Jun 17 14:34:06 localhost su[15927]: - fatemeh:root fatemeh:root
```

اگر اسکریپت را بخوانید متوجه می‌شوید که همه چیز با بلوک BEGIN شروع می‌شود. در برنامه ما، کار این بلوک چاپ یک سرایند در

بالای خـــروجـــی بـــرنـــامه است. همچنین متغیر failed را برابـر صـفر قرار می‌دهد. سپس حلقـه اصـلی برنامـه از ورودی خـط فـــرمـــان فایـل /var/log/auth.log را می‌خواند. در این فایل ثبت رخداد تمـامی ورود و خـروج‌هـا بـه سیسـتم ثبـت مـی‌شـوند، از جملـه ورود و خـــروج‌هـای نـــامـــوفق و دستـــورات su. مـا در این فایل به دنبـال دسـتورات su ای هستیم که با شکست مواجه شده‌اند. درصورتی‌که چنین خطی در ورودی یافت شـــود، بـــرنامه زمان، محل و نام کاربر نـاموفق را بـر روی صـفحه چاپ خواهد کرد. در پایان نیز تعـداد کـل رکوردهـا و تعـداد رکوردهـای شکست‌خورده را چـــاپ می‌کـنـد. کافـــی است کدهای فوق را در یـک فایل ذخیره کرده (مثلا بـا نـام authlogck) و سـپس بـــا اسـتـــفاده از دستـــور chmod به آن مجوز اجـرا دهیـد. سـپس آن را بـه‌صـورت زیـر توسط کاربر ریشه اجرا کنید:

```
# cat /var/log/auth.log | ./authlogck
```

خروجی مشابه با زیر دریافت خواهید کرد:

```
--- checking authlog
failed su:    fatemeh:root  at   Feb   3    21:03:27
failed su:    fatemeh:root  at   Feb   5    17:18:43
failed su:    fatemeh:root  at   Feb   6    17:27:57
failed su:    fatemeh:root  at   Feb   7    17:23:00
failed su:    fatemeh:root  at   Feb   8    21:39:45
failed su:    fatemeh:root  at   Feb   10   11:23:19
failed su:    fatemeh:root  at   Feb   10   11:23:24
failed su:    fatemeh:root  at   Feb   10   14:17:27
failed su:    fatemeh:root  at   Feb   11   18:33:20
failed su:    fatemeh:root  at   Feb   14   18:37:55
failed su:    fatemeh:root  at   Feb   14   18:51:36
failed su:    fatemeh:root  at   Feb   18   11:27:14
failed su:    fatemeh:root  at   Feb   18   11:27:19
failed su:    fatemeh:root  at   Feb   18   13:56:21
```

```
failed su:    fatemeh:root  at    Feb    19     08:42:16
failed su:    fatemeh:root  at    Feb    19     11:26:47
failed su:    fatemeh:root  at    Feb    21     19:54:28
failed su:    fatemeh:root  at    Feb    25     00:02:49
failed su:    fatemeh:root  at    Feb    25     09:06:34
failed su:    fatemeh:root  at    Feb    25     18:22:56
failed su:    fatemeh:root  at    Feb    25     18:37:13
failed su:    fatemeh:root  at    Feb    26     19:11:34
failed su:    fatemeh:root  at    Mar    4      08:35:03
failed su:    fatemeh:root  at    Mar    5      18:35:48
failed su:    fatemeh:root  at    Mar    9      17:00:50
failed su:    fatemeh:root  at    Mar    9      20:06:15
failed su:    fatemeh:root  at    Apr    9      11:26:55
failed su:    fatemeh:root  at    Apr    13     18:21:39
failed su:    fatemeh:root  at    Apr    29     20:01:25
failed su:    fatemeh:root  at    May    3      18:32:04
failed su:    fatemeh:root  at    May    8      19:58:03
failed su:    fatemeh:root  at    May    11     20:52:19
failed su:    fatemeh:root  at    May    16     19:51:12
failed su:    fatemeh:root  at    May    17     20:15:12
failed su:    fatemeh:root  at    May    20     11:05:41
failed su:    fatemeh:root  at    Jun    3      16:30:50
failed su:    fatemeh:root  at    Jun    5      14:59:20
failed su:    fatemeh:root  at    Jun    17     11:01:26
failed su:    fatemeh:root  at    Jun    17     12:26:40
failed su:    fatemeh:root  at    Jun    17     14:13:58
failed su:    fatemeh:root  at    Jun    17     14:34:06
---------------------------------------
        total number of records:    14136
        total number of failurs:    41
```

در برنامه بالا می‌توانید به‌خوبی ببینید که دستور printf چگونه کار می‌کند. بسیار شبیه فراخوانی printf در زبان C است. به‌صورت پیش‌فرض، برای awk جداکننده فیلدها، فضای خالی است. می‌توانید

ایـــن مقـــدار پیش‌فرض را تغییر دهیـد. بـرای مثـال در مـواردی ماننـد فایل‌های کلمه عبور، آن را می‌توانید برای روی یک کالن (:) تنظیم کنید. اسکریپت کوچک زیر فایل passwd را به دنبال حساب‌های کاربری ریشه (با شماره کاربری صفر) و حساب‌های بدون کلمه عبور می‌گردد:

```
#!/usr/bin/awk -f
BEGIN { FS=":" }
{
 if ($3 == 0) print $1
 if ($2 == "") print $1
}
```

سایر جداکننده‌های داخلی مورد استفاده در awk عبارتند از:

RS: یا record separator که به‌صـورت پـیش فـرض بـر روی خـط جدید یا n\ تنظیم شده است.

OFS یا output field separator

ORS یا output record separator که به‌صـورت پـیش فـرض بـر روی خط جدید یا n\ تنظیم شده است.

تمامی این مقادیر از میان اسکریپت قابل تعریف کردن هستند.

اجرای AWK از خط فرمان

با دستور "awk" ما از خط فرمان ترمینال لینوکس قـادر بـه اجـرای awk خواهیم بود.

در ساده ترین حالت، ما کدهای awk را در همان خط فرمان نوشته و اجرا می‌کنیم. فرمت کلی اجرای دستورات نوشته شده در خط فرمـان بـه این شکل خواهد بود:

```
awk 'commands' inputFile
```

ما دستورات awk را داخل یک جفت علامت ' ' نوشته و در پایان، فایل ورودی که باید اطلاعات آن بررسی شود را به عنوان ورودی به awk می‌دهیم. هر بلاک کد awk باید داخل یک جفت علامت {} قرار گیرد.

مثال ۱)

```
awk '{ print $2 }' input.dat
```

در این خط از دستور، گفته ایم که اطلاعات ستون ۲ از فایل دیتای file.dat را در خروجی چاپ کن. دستور print خروجی را در همان صفحه نمایش چاپ خواهد کرد.

در واقع برنامه AWK مجموعه ای از عبارات به شکل زیر است:

```
condition { action}
```

اگر قسمت condition را نگذاریم کل دستورات داخل گیومه برای تمام سطرها و بدون قید و شرط اجرا می‌شود. اگر در قسمت شرط، عبارتی بنویسیم، فقط برای سطرهایی که آن عبارت شرط صحیح باشد، مد داخل گیومه اجرا خواهد شد.

در نوشتن برنامه های awk چند متغیر سیستمی هستند که به طور مکرر استفاده می‌شوند. به توضیح این متغیرها توجه نمایید:

NF: متغیری است که به آخرین ستون از فایل، اشاره دارد.

N0: متغیری است که به اولین ستون از فایل، اشاره دارد.

FNR: متغیری است که به تعداد سطرهای فایل، اشاره دارد.

نکته) ستون‌های داده در awk با پیشوند $ و سپس شماره ستون مشخص می‌شوند. $۱ نشان دهنده ستون اول فایل دیتا و $NF نشان دهنده ستون آخر است.

مثال ۲)

```
awk ' {sum=0; for(i=1; i<NF; i++) sum += $i; print
sum/NF}' input.dat
```

در این مثال، یک حلقه از اولین فیلد یک سطر تا فیلد آخر آن اجراشده، مقادیر موجود در فیلدها را جمع می‌کند و در پایان با تقسیم بر تعداد فیلدها (ستون‌ها) میانگین مقادیر موجود در هر سطر را بدست می‌آورد. این مثال ساده برای درک زبان awk است. در واقع چنین کاری به ندرت در عمل صورت می‌گیرد که اطلاعات موجود در فیلدهای یک سطر را میانگین بگیریم!

توجه داشته باشید که این کار برای کل سطرهای موجود در فایل دیتا تکرار خواهد شد. کلا دستوراتی که داخل بلاک {} نوشته می‌شود برای کل سطرها تکرار می‌شود.

وقتی که حاصل یک دستور awk یک سری اطلاعات حجیم باشد، بهتر است آن را به فایل جدیدی هدایت کنیم تا قابل مدیریت باشد. در مثال زیر نحوه انجام این کار نمایش داده شده است:

مثال ۳)

```
print "(($1+$2)/2)" > "output.txt" input.dat
```

در این مثال، خروجی دستور پرینت، به یک فایل جدید، هدایت شده است.

نکته) برای بستن فایل‌هایی که به منظور ذخیره نتایج، باز شده اند، از دستور close استفاده می‌شود.

```
close(fileName)
```

Regular Expression در awk

زبان awk از Regular Expression ها بسیار بهتر از grep پشتیبانی می‌کند. امکان استفاده از عملگرهایی مانند && و || به همراه عملگر ! وجود دارد. برای مثال، برای پیدا کردن خطوطی در فایل

auth.log که در ماه Feb و Jun نگاشته نشده‌اند و توسط CRON نگاشته شده‌اند، باید دستور زیر را وارد کرد:

```
# awk '!(/Feb/ || /Jun/) && /CRON/' /var/log/auth.log | less
Mar 1 17:48:50 localhost CRON[5886]: (pam_unix) session opened for user logcheck by (uid=0)
Mar 1 17:48:58 localhost CRON[5886]: (pam_unix) session closed for user logcheck
Mar 1 17:50:01 localhost CRON[7273]: (pam_unix) session opened for user root by (uid=0)
Mar 1 17:50:01 localhost CRON[7273]: (pam_unix) session closed for user root
Mar 1 18:00:01 localhost CRON[7577]: (pam_unix) session opened for user root by (uid=0)
Mar 1 18:00:01 localhost CRON[7577]: (pam_unix) session closed for user root
Mar 1 18:02:01 localhost CRON[7608]: (pam_unix) session opened for user logcheck by (uid=0)
Mar 1 18:02:05 localhost CRON[7608]: (pam_unix) session closed for user logcheck
Mar 1 18:09:01 localhost CRON[8305]: (pam_unix) session opened for user root by (uid=0)
Mar 1 18:09:01 localhost CRON[8305]: (pam_unix) session closed for user root
Mar 1 18:10:01 localhost CRON[8325]: (pam_unix) session opened for user root by (uid=0)
```

حتی این امکان وجود دارد تا مقایسه regex را تنها بر روی یکی از فیلدها انجام دهید:

```
# awk '$1 !~ /Feb/ && $5 !~ /CRON/' /var/log/auth.log | less
```

چاپ خروجی در awk

در awk بـــا اســـتـــفاده از دســـتورات print و printf مـــی‌توانیـــد خروجی برنامه را چاپ کنید. همانند آنچه در بخش‌های پیشـین دیدیـد، می‌توانید برای چاپ از شماره فیلد مانند $۵ و یا از متنی که بین "" قـرار گرفته باشد استفاده کنید؛ مثلا:

```
$ awk 'BEGIN { print “line one\nline two\nline three" }'
line one
line two
line three
```

همانطور که می‌بینید، درج کـاراکتر \n در بـین مـتن باعـث شکسـته شـدن مـتن چـاپی در خطـوط جدیـد مـی‌شـود. تعـدادی از ایـن نـوع کاراکترهای قابل استفاده عبارتند از:

- کاراکترهای \n: برای چاپ متن در خط جدید.
- کاراکترهای \a: برای ایجاد یک alert که معمولا یک بوق می‌باشد.
- کاراکترهای \b: به‌صورت یک backspace عمل می‌کند.
- کاراکترهای \t: به‌صورت کلید tab عمل می‌کند.
- کاراکترهای \f: برای ایجاد یک Form Feed.
- کاراکترهای \v: برای ایجاد یک tab عمودی.
- کاراکترهای \\: برای ایجاد یک backslash.
- کاراکترهای "\"": برای قرار دادن یک کلمه در گیومه مانند:

```
$ awk 'BEGIN { print "line \"one\"\nline two\nline three" }'
line "one"
line two
line three
```

این امکان وجود دارد تا بتوانید فیلدهای مورد نظر خود را در خروجی چاپ کنید. به مثال زیر توجه کنید:

```
# awk '{print $1, $2}' /var/log/auth.log
Feb 1
Feb 1
Feb 1
```

توجه داشته باشید که کامای بین دو فیلد مهم است. درصورتی‌که این کاما فراموش شود، دو رشته در خروجی برنامه به هم متصل خواهند شد. خروجی برنامه بالا بدون کاما به‌صورت زیر خواهد بود:

```
# awk '{print $1 $2}' /var/log/auth.log
Feb1
Feb1
Feb1
```

برای تمیزکاری بیشتر و برای نمایش توضیحی خروجی به افرادی که ممکن است با فایل شما آشنا نباشند، می‌توانید با استفاده از تابع BEGIN یک سرایند برای آن چاپ کنید. برای مثال:

```
# awk 'BEGIN{print "Date\t\Time\t\tHost\n"}{print $1,$2"\t"$3"\t"$4}' /var/log/auth.log
Date Time Host

Feb 1 18:09:01 localhost
Feb 1 18:09:01 localhost
Feb 1 18:14:40 localhost
Feb 1 18:17:01 localhost
Feb 1 18:17:01 localhost
Feb 1 18:17:25 localhost
Feb 1 18:17:25 localhost
Feb 1 18:39:02 localhost
Feb 1 18:39:02 localhost
```

جداکننده‌های خروجی

همانطور کـــه در مثـــال قبل مشاهده کردید، بطور پیش‌فرض، با قرار دادن کامـا، یـک فضـای خـالی (space) بـین فیلـدهای خروجـی چـاپ مـی‌شـود. بـا تنظـیم متغیـر OFS یـــا Output Field Separator می‌توانید بجای فضای خالی، کاراکتر دیگـری را بعنـوان جداکننـده فیلـد خروجی قرار دهید.

متغیر دیگری وجود دارد کـه نـام آن ORS یـــا Output Record Separator می‌باشد. این متغیر وظیفه جداسازی رکوردها را در خروجـی به عهده دارد. مقدار پیش‌گزیده ایـن متغیـر \n یـا کـاراکتر خـط جدیـد می‌باشد. مقـــادیـــر ایـن دو متغیر را باید در بخش BEGIN اسـکریپت خود تنظیم کنید. به مثال زیر توجه کنید:

```
# awk 'BEGIN{ORS="\n----\n"; OFS="\t"; print "Date\t\Time\t\tHost\n"}
> {print $1 $2,$3,$4}' /var/log/auth.log
Date Time Host
----
Feb1 18:09:01 localhost
----
Feb1 18:09:01 localhost
----
Feb1 18:14:40 localhost
----
Feb1 18:17:01 localhost
----
Feb1 18:17:01 localhost
```

عملگرهای Regular Expression

این امکان وجود دارد تا بتوانید Regular Expression ها را با کاراکترهای ویژه‌ای که به آن‌ها عملگرهای Regular Expression اطلاق می‌شود، ترکیب کنید. این کار باعث می‌شود تا بتوانید عملکرد و قدرت Regular Expression ها را افزایش دهید.

می‌توانید کاراکترهایی که در بخش چاپ خروجی به آن‌ها اشاره شد را به همراه Regex ها به کار بگیرید. این کاراکترها در مراحل نخست پردازش شناسایی شده و به کاراکترهای اصلی تبدیل می‌شوند.

اکنون به این کاراکترها می‌پردازیم:

- کاراکتر \: بکاربردن این کاراکتر، معنی مخصوص برخی از کاراکترها را از میان می‌برد. برای مثال، \$ به معنی خود کاراکتر $ تفسیر می‌شود و نه یک متغیر خاص.
- کاراکتر ^: به معنای نقطه شروع یک رشته متنی است. برای مثال @chapter^ به‌صورت @chapter تفسیر شده و برای مثال می‌تواند برای تشخیص شروع فصول در یک فایل منبع Texinfo بکار گرفته شود. کاراکتر ^ به نام anchor خوانده می‌شود زیرا باعث می‌شود الگو تنها در جایی که یک رشته متنی شروع شده باشد، مصداق داشته باشد. توجه داشته باشید که از این کاراکتر نمی‌توان برای تشخیص نقطه شروع یک خط جدید در میان یک رشته استفاده کرد. برای مثال شرط زیر صحیح نخواهد بود:

```
if (“line1\nLINE 2” ~ /^L/) ...
```

- کاراکتر $: این کاراکتر مشابه با کاراکتر ^ بوده با این تفاوت که نقطه پایانی یک رشته متنی را مشخص می‌کند. برای مثال p$ رکوردی را مشخص می‌کند که با یک کاراکتر p خاتمه پیدا کرده باشد. همانند کاراکتر ^ از $ نیز نمی‌توان برای مشخص کردن نقطه پایان خط که در میان یک رشته متنی قرار دارد استفاده کرد. بنابراین شرط زیر صحیح نیست:

```
if (“line1\nLINE 2” ~ /1$/) ...
```

- کاراکتر . (نقطه): می‌تواند نشانگر هر تک کاراکتر شامل کاراکتر خط جدید باشد. برای مثال، P. به معنی هر تک کاراکتری است که پس از آن کاراکتر P قرار دارد. با این حساب می‌توانید یک Regex به‌صورت U.A ایجاد کنید که معنی آن هر کلمه سه کاراکتری است که ابتدای آن U و انتهای آن A قرار دارد.
- [...] (لیست کاراکترها): هر کاراکتری را که در بین براکت‌ها قرار داده شده باشد را شامل می‌شود. برای مثال [MVX] هرکدام از کاراکترهای M، V و X را در یک رشته شامل می‌شود. محدوده کاراکترها را می‌توان با استفاده از یک خط تیره یا کاراکتر hyphen مشخص کرد. برای مثال، [a-dx-z] معادل [abcdxyz] می‌باشد. توجه داشته باشید این امر هنگامی صادق است که تنظیمات محلی سیستم روی C قرار داشته باشد. بسیاری از تنظیمات محلی دیگر معمولا کاراکترها را با ترتیب واژه‌نامه‌ای لیست

می‌کنند، بنابراین مثال بالا معادل خواهد بود با [aBbCcDdxXyYz]. با تنظیم متغیر محیطی LC_ALL بر روی C می‌توانید از براکت‌ها به حالت سنتی تنظیمات محلی C استفاده کنید، حتی اگر locale سیستم بر روی مقدار دیگری تنظیم شده باشد. برای اینکه بتوانید یک کاراکتر \ یا [یا – یا ^ را در یک لیست کاراکتر بگنجانید، قبل از آن \ قرار دهید. برای مثال:

```
[d\]]
```

که d و [را شامل خواهد شد. این رفتار کاراکتر \ در کلیه نگارش‌های مختلف زبان awk تعریف شده است. یکی دیگر از ویژگی‌های قابل استفاده، کلاسهای کاراکتری تعریف شده در استاندارد POSIX هستند. این کلاسها شامل یک :]، یک کلیدواژه و یک [: می‌باشند. این کلاسها به قرار زیر هستند:

[:alnum:]	کاراکترهای Alphanumeric
[:alpha:]	کاراکترهای الفبایی
[:blank:]	کاراکترهای Space و Tab
[:cntrl:]	کاراکترهای کنترل
[:digit:]	کاراکترهای عددی
[:graph:]	کاراکترهای قابل دیدن و چاپ
[:lower:]	کاراکترهای الفبایی کوچک
[:print:]	کاراکترهای قابل چاپ

[:punct:]	کاراکترهای نشانه گذاری
[:space:]	کاراکترهای شامل فضای خالی مانند tab و space
[:upper:]	کاراکترهای الفبایی بزرگ
[:xdigit:]	کارکترهایی که ارقام هگزادسیمال هستند

- [...^] (لیست کاراکترهای متمم): این یک امکان تکمیل کننده برای لیست کاراکترها است. نخستین کاراکتر پس از براکت آغازین باید یک کاراکتر ^ باشد. معنی آن تمامی کاراکترهاست بجز آنچه در لیست قرار داده شده‌اند.
- کاراکتر | (تناوب): از این کاراکتر برای مشخص کردن یک گزینه جایگزین برای شرط، استفاده می‌شود. همانند عملگر or در زبان‌های برنامه نویسی. برای مثال

```
^P|[[:digit:]]
```

شامل تمامی رشته‌های متنی است که با P شروع شده یا حاوی عدد باشند.

- (...): برای دسته کردن Regex هایی است که در میان آن‌ها از کاراکتر تناوب استفاده شده است. برای مثال:

```
;@(samp|code)\{[^}]+\}
```

می‌تواند در مورد هر دو رشته زیر مصداق داشته باشد:

```
;@code{foo} or @samp{bar};
```

در مورد کاراکتر + که در مثال بالا آورده شده است، جلوتر توضیح خواهیم داد.