Linux Professional Institute

LPIC-1

جزوه‌ی دوم: آشنایی با فایل‌های متنی و Regular Expressionها

By: The Albatross

thealbatross@yandex.com
https://github.com/TheAlbatrossCodes/Linux-In-Persian

فهرست مطالب

مقدمه

جلسه‌ی قبل به طور کلی در مورد دوره‌ی LPIC-1 و سرفصل دوره صحبت کردیم. سپس در مورد فلسفه‌ی لینوکس و توزیع‌های متفاوت آن صحبت کردیم. در نهایت به سراغ دانلود، نصب و آشنایی اولیه با سیستم عامل CentOS رفتیم. در این جلسه، بیشتر با محیط shell آشنا می‌شویم و با ابزارهای موجود برای خواندن فایل‌های متنی و ایجاد تغییراتی در آنها آشنا می‌شویم. سپس به سراغ ابزار قدرتمند grep و regular expressionها رفته و سعی می‌کنیم توضیحی ابتدایی در مورد مفهوم آنها داده و طریقه‌ی استفاده از آنها را یاد بگیریم.

مشاهده‌ی فایل‌های متنی در Shell

برای مشاهده‌ی فایل‌های متنی در Shell، کافی است از دستور `cat` استفاده کنید. برای مثال:

```
[root@localhost ~]# cat /etc/centos-release
CentOS Linux release 7.7.1908 (Core)
```

همانطور که می‌بینید، این دستور محتویات فایل `centos-release`، که حاوی ورژن سیستم عامل نصب شده بود را به ما در خروجی نشان داد. دستور cat علاوه بر قابلیت نمایش فایل‌ها، قابلیت concatکردن یا به هم چسباندن چند فایل متنی را نیز دارد. بعدا با این قابلیت `cat` بیشتر آشنا می‌شویم.

آشنایی بیشتر با bash و متغیرهای آن

در جلسه‌ی قبل به صورت ابتدایی با bash و برخی از دستورات internal آن آشنا شدیم. حال می‌خواهیم با برخی از متغیرهای موجود در آن نیز آشنا شویم. این متغیرها به ما کمک می‌کنند تا کنترل بیشتری روی رفتار سیستم داشته باشیم.

ابتدا بیایید با دستوری به نام `echo` آشنا شویم. استفاده از این دستور بسیار ساده است:

```
[root@localhost ~]# echo kill me
kill me
```

همانطور که می‌بینید، هر عبارتی که جلوی دستور `echo` بنویسیم، در خروجی ترمینال به ما نشان داده می‌شود. آیا می‌توانید نام خود را با دستور `echo` در ترمینال نمایش دهید؟ 😉

> شاید فکر کنید این دستور، دستور به‌دردنخوری باشد. اما این دستور در اسکریپتینگ و کارهایی نظیر مشاهده‌ی متغیرهای موجود در bash، تغییر یک متغیر، اضافه‌کردن متن به فایل‌ها و... بسیار کاربردی می‌باشد.

حال بیایید با برخی از متغیرهای موجود در bash آشنا شویم. تا به حال فکر کردید که bash، از کجا می‌فهمد که دستورهای external (که مربوط به برنامه‌هایی هستند که روی سیستم نصب کرده‌اید) را باید از کجا پیدا کند؟ مثلا دستور `man` را در نظر بگیرید. جلسه‌ی قبل دیدیم که `man`، یک دستور external است، یا به عبارتی، برنامه‌ای است که روی سیستم نصب کرده‌ایم. ما برای دسترسی به برنامه‌های نصب شده، باید به محل نصب آن برویم تا بتوانیم از آن استفاده کنیم؛ اما برای دستور `man` چنین کاری را انجام نمی‌دهیم.

ابتدا بیایید ببینیم که دستور `man`، در کجا نصب شده است. این کار را با استفاده از دستور `which` انجام می‌دهیم:

```
[root@localhost ~]# which man
/usr/bin/man
```

همانطور که می‌بینید فایل اجرایی برنامه‌ی man، در /usr/bin/man قرار دارد. اما ما برای دسترسی به manpage ها به سراغ /usr/bin/man نمی‌رویم، بلکه فقط با نوشتن man در محیط bash، وارد این برنامه می‌شویم.

bash، با استفاده از متغیر $PATH می‌تواند برنامه‌هایی که در مسیرهای خاصی نصب شده‌اند را بیابد. بیایید اول محتوای این متغیر را ببینیم:

```
[root@localhost ~]# echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/root/bin
```

همانطور که می‌بینید، /usr/bin در متغیر $PATH وجود دارد، پس bash به سادگی می‌تواند فایل اجرایی دستور man را پیدا کرده و آن را اجرا کند. شما وقتی یک دستور مانند man را وارد bash می‌کنید، bash به سراغ مسیرهایی که در متغیر $PATH خود ذخیره کرده رفته و به دنبال آن برنامه یا دستور می‌گردد و در صورت پیدا کردن، آن را اجرا می‌کند. بعدا که با اسکریپینگ آشنا شویم، می‌بینید که با قرار دادن اسکریپت خود در یکی از مسیرهای موجود در $PATH، می‌توانید از هر جایی در سیستم، به آن اسکریپت دسترسی پیدا کنید.

با نوشتن دستور env، می‌توانید همه‌ی متغیرهای bash را مشاهده کنید:

```
[root@localhost ~]# env
...
MAIL=/var/spool/mail/root
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/root/bin
PWD=/root
LANG=en_US.UTF-8
SELINUX_LEVEL_REQUESTED=
HISTCONTROL=ignoredups
SHLVL=1
HOME=/root
...
```

همانطور که می‌بینید، دستور env متغیرهای محیطی یا environment variableهای bash را به ما نشان می‌دهد. البته در این اینجا، ما فقط بخشی از خروجی را نمایش دادیم.

آشنایی با Redirectorها

سیستم‌های لینوکسی، ورودی‌هایی که به سیستم می‌دهیم و خروجی‌هایی که از سیستم می‌گیریم را در متغیرهای متفاوتی می‌ریزند و به هر متغیر، یک عدد ثابت اختصاص می‌دهند (که به آن File Descriptor می‌گویند). این متغیرها و عملکرد آنها به شرح زیر می‌باشند:

- Standard Input یا STDIN

هر ورودی که به یک دستور (معمولا از طریق کیبورد) بدهیم، به عنوان STDIN در نظر گرفته می‌شود. عدد مربوط به این متغیر، 0 می‌باشد.

- Standard Output یا STDOUT

دستورها یا برنامه‌های لینوکسی، در صورت اجرای موفق، خروجی خود را از طریق STDOUT برای کاربر می‌فرستند. به صورت پیش‌فرض STDOUT خروجی را روی صفحه‌ی command line یا ترمینال کنونی نشان می‌دهد. عدد مربوط به این متغیر، 1 می‌باشد.

- Standard Error یا STDERR

دستورها یا برنامه‌های لینوکسی اطلاعات مهم نظیر errorها یا اجرای ناموفق خود را از طریق STDERR برای کاربر می‌فرستند. به صورت پیش‌فرض STDERR و STDOUT هر دو بر روی صفحه‌ی ترمینال کاربر نمایش داده می‌شوند. عدد مربوط به این متغیر، 2 می‌باشد.

ممکن است از خود بپرسید این مسائل به چه درد شما می‌خورند. این متغیرها برای ساده کردن کار ما ایجاد شده‌اند و برای انجام کارهایی نظیر اسکیرپتینگ بسیار کاربردی خواهند بود. مثلا ما می‌توانیم از bash بخواهیم که خروجی اجرای ناموفق یک دستور را درون یک فایل بریزد. bash با استفاده از اطلاعات STDERR و مفهومی به نام Redirection این کار را انجام می‌دهد.

Redirectorها عملگرهایی هستند که می‌توانند خروجی یک دستور، اعم از STDOUT و STDERR، و همچنین هر متنی را به فایل یا ورودی برنامه‌ای دیگر redirect کنند. مثلا:

```
[root@localhost ~]# echo behnam > myName.txt
```

این دستور، اسم behnam را درون فایل `myName.txt` می‌ریزد. حال بیایید نگاهی به فایل `myName.txt` بیاندازیم:

```
[root@localhost ~]# cat myName.txt
behnam
```

البته اینکار فقط محدود به یک نوشته نیست، ما حتی می‌توانیم محتوای یک متغیر bash را درون یک فایل بریزیم:

```
[root@localhost ~]# echo $PATH > path.txt
```

اگر به فایل `path.txt` نگاه کنید، می‌بینید که محتویات متغیر $PATH درون فایل path.txt قرار گرفته است.

اما اینجا یک مشکل وجود دارد. مگر دستور `echo`، خروجی خود را روی صفحه نیز به ما نشان نمی‌داد؟ چرا الان چیزی در ترمینال به ما نشان نمی‌دهد؟

> اینجاست که باید به مفهوم STDOUT بازگردیم. گفتیم که دستورها خروجی اجرای موفق خود را درون STDOUT می‌ریزند و STDOUT نیز به صورت پیش‌فرض، خروجی خود را روی صفحه نشان می‌دهند، اما زمانی که از یک redirector، نظیر عملگر > استفاده می‌کنیم، به سیستم می‌گوییم که محتویات STDOUT را به جای نمایش روی ترمینال، درون یک فایل بریزد.

همانطور که گفتیم، ما می‌توانیم خروجی یک دستور را نیز در یک فایل بریزیم. مثلا:

```
[root@localhost ~]# pwd > pwd.txt
[root@localhost ~]# cat pwd.txt
/root
```

همانطور که می‌بینید، خروجی دستور pwd، به جای نمایش بر روی ترمینال، درون فایل pwd.txt ریخته شد. بیایید این کار را بر روی دستور man نیز اجرا کنیم!

```
[root@localhost ~]# man > man.txt
What manual page do you want?
```

اگر خروجی این دستور نگاه کنید، می‌بینید که سیستم به ما متن What manual page do you want? را روی ترمینال نشان داد و اگر فایل man.txt را نیز باز کنید، محتوایی درون آن نمی‌بینید. مگر قرار نبود خروجی دستور با عملگر > درون یک فایل ریخته شود؟

> برای درک دلیل این قضیه، باید به مفهوم STDERR باز گردیم. گفتیم که دستورها در صورت اجرای ناموفق یا برخورد به یک خطا، خروجی خود را درون STDERR می‌ریزند. در بالا، ما فقط دستور man را وارد ترمینال کردیم، اما به man نگفتیم که صفحه‌ی راهنمای کدام دستور را می‌خواهیم. از نظر برنامه‌ی man، این یک دستور اشتباه است. به همین دلیل، دستور man، عبارت What manual page do you want? را درون STDERR ریخت و STDERR هم طبق رفتار پیش‌فرض خود، محتویات خود را روی ترمینال به ما نشان داد. خوب، با همه‌ی این حرف‌ها، مشکل در چیست؟

مشکل این است که عملگر >، فقط محتوای STDOUT را redirect می‌کند. برای این که کاری کنیم که محتوای STDERR درون یک فایل یا ورودی یک برنامه redirect شود، از عملگر 2> استفاده می‌کنیم:

```
[root@localhost ~]# man 2> man.txt
[root@localhost ~]# cat man.txt
What manual page do you want?
```

همانطور که می‌بینید با استفاده از عملگر 2>، خروجی اجرای ناموفق دستور man را درون یک فایل دیگر redirect کردیم. پرواضح است که در امر مدیریت سیستم و اسکریپتینگ، این عملگر بسیار کاربردی می‌باشد.

> پس تا اینجا یاد گرفتیم که عملگر >، خروجی اجرای موفق دستور یا STDOUT را redirect کرده و عملگر 2>، خروجی اجرای ناموفق یا خطای یک دستور، یا همان STDERR را redirect می‌کند.

حال بیایید ابتدا یک متن دلخواه را با استفاده از دستور echo درون یک فایل بریزیم و سپس خروجی دستور which man را که بالاتر با آن آشنا شدیم را درون همان فایل قبلی، redirect کنیم. اول متن دلخواه خود را می‌نویسیم و در فایلی با نام test.txt، ریدایرکت می‌کنیم:

```
[root@localhost ~]# echo kill me please! > test.txt
[root@localhost ~]# cat test.txt
kill me, please!
```

حال به سراغ redirect کردن خروجی دستور which man در همان فایل test.txt می‌رویم:

```
[root@localhost ~]# which man > test.txt
[root@localhost ~]# cat test.txt
/usr/bin/man
```

همانطور که می‌بینید، فایل test.txt فقط خروجی دستور which man را درون خود دارد. دلیل این امر چیست؟

دلیل این است که عملگر > (وهمچنین 2>) در هر بار اجرا، یک فایل جدید ایجاد می‌کنند و اگر فایلی با آن نام در این مکان وجود داشته باشد، آن فایل را پاک کرده و فایل جدید را به جای آن قرار می‌دهند؛ یا به عبارت دیگر، فایل را overwrite میکنند.

برای حل این مشکل، ما از عملگر >> برای STDOUT و 2>> برای STDERR استفاده می‌کنیم. پس با این دانش، می‌توانیم مشکل مثال قبل را حل کنیم:

```
[root@localhost ~]# echo kill me please! > test.txt
[root@localhost ~]# cat test.txt
kill me, please!
[root@localhost ~]# which man >> test.txt
[root@localhost ~]# cat test.txt
kill me please!
/usr/bin/man
```

همانطور که می‌بینید، با استفاده از عملگر >>، سیستم به جای ایجاد فایل جدید، محتویات STDOUT را درون فایل مشخص شده، append کرد.

در جدول زیر، عملگرهای متفاوت redirection و عملکرد هر کدام را می‌بینید:

عملگر	عملکرد
>	خروجی STDOUT را درون یک فایل جدید می‌ریزد. اگر آن فایل از قبل وجود داشته باشد، آن فایل پاک شده و فایل جدیدی ایجاد می‌شود.
>>	خروجی STDOUT را درون فایل مشخص شده الحاق (append) می‌کند. اگر فایل مشخص شده وجود نداشته باشد، آن فایل را ایجاد می‌کند.
2>	خروجی STDERR را درون یک فایل جدید می‌ریزد. اگر آن فایل از قبل وجود داشته باشد، آن فایل پاک شده و فایل جدیدی ایجاد می‌شود.
2>>	خروجی STDERR را درون فایل مشخص شده الحاق (append) می‌کند. اگر فایل مشخص شده وجود نداشته باشد، آن فایل را ایجاد می‌کند.
&>	خروجی STDOUT و STDERR را درون یک فایل جدید می‌ریزد. اگر آن فایل از قبل وجود داشته باشد، آن فایل را پاک کرده و فایل جدیدی ایجاد می‌شود.
&>>	خروجی STDOUT و STDERR را درون فایل مشخص شده الحاق (append) می‌کند. اگر فایل مشخص شده وجود نداشته باشد، آن فایل را ایجاد می‌کند.
<	محتویات فایل مشخص شده را درون STDIN می‌ریزد.
<<	نوشته‌های موجود در چند خط بعدی را تا زمان رسیدن به عبارت مشخص شده توسط کاربر، درون STDIN می‌ریزد. (این عملگر از ما نام فایل دریافت نمی‌کند.)
<>	محتویات موجود در فایل مشخص شده را درون STDIN می‌ریزد و سپس STDOUT را درون همان فایل می‌ریزد.

آشنایی با دستور less

ابتدا بیایید با دستوری به نام ls آشنا شویم. ما با استفاده از دستور whatis، می‌توانیم یک توضیح یک خطی در مورد دستور مورد نظر دریافت کنیم:

```
[root@localhost ~]# whatis ls
ls (1)               - list directory contents
```

همانطور که می‌بینید، whatis به صورت مختصر به ما می‌گوید که دستور ls، محتویات موجود در یک فولدر را به ما نشان می‌دهد. بیایید این دستور را امتحان کنیم:

```
[root@localhost ~]# ls
anaconda-ks.cfg  ip.txt  man.txt  myName.txt  pwd.txt  test.txt
```

همانطور که می‌بینید، ls محتویات موجود در فولدر کنونی را به ما نشان داد.

آیا می‌توانید با مراجعه به manpage دستور ls، کاری کنید که این دستور محتویات یک فولدر را به صورت یک لیست به ما نشان دهد؟

برای این که کاری کنیم که ls محتویات یک فولدر را به صورت یک لیست به ما ارائه دهد، باید از آپشن l- استفاده کنیم. به صورت زیر:

```
[root@localhost ~]# ls -l
total 20
-rw-------. 1 root root 1257 Mar 20 10:48 anaconda-ks.cfg
-rw-r--r--. 1 root root    0 Mar 23 12:55 ip.txt
-rw-r--r--. 1 root root   30 Mar 24 14:04 man.txt
-rw-r--r--. 1 root root    7 Mar 24 12:12 myName.txt
-rw-r--r--. 1 root root    6 Mar 24 13:48 pwd.txt
-rw-r--r--. 1 root root   29 Mar 24 14:39 test.txt
```

همانطور که می‌بینید، با استفاده از آپشن l- توانستیم کاری کنیم که محتویات فولدر کنونی در قالب یک لیست نمایش داده شود.

حال بیایید این دستور جدید را در چند قسمت دیگر سیستم هم امتحان کنیم. احتمالا فولدر /etc را از جلسه قبل به خاطر داشته باشید. بیایید محتویات درون آن را ببینیم:

```
[root@localhost ~]# ls -l /etc
```

همانطور که می‌بینید، خروجی این دستور کل صفحه‌ی ما را پر کرد و اصلا مشخص نیست در بالای صفحه چه فولدر یا فایل‌های موجود می‌باشد. این یک مشکل بزرگ است. به نظر شما چطور می‌توانیم آن را حل کنیم؟

در بخش قبل، با redirectorها آشنا شدیم. اگر خروجی دستور ls -l /etc را درون یک فایل متنی بریزیم، می‌توانیم آن را به راحتی بخوانیم! پس بیایید این کار را انجام دهیم:

```
[root@localhost ~]# ls -l /etc > ls.txt
```

حال بیایید محتوای ls.txt را به راحتی ببینیم:

```
[root@localhost ~]# cat ls.txt
```

عجب. باز هم به همان مشکل برخوردیم 😐. ما به دنبال دستوری هستیم که به ما اجازه دهد محتویات یک فایل متنی را بخوانیم و آنرا scroll یا بالا و پایین نیز کنیم.

خوشبختانه، دستور less به ما قابلیت این کار را می‌دهد:

```
[root@localhost ~]# less ls.txt
```

همانطور که می‌بینید، دستور less، فایل ls.txt را از خط اول به ما نشان می‌دهد. عملیات در بسیار این صفحه ساده است:

- با زدن دکمه‌ی ↑ و ↓ روی کیبورد، می‌توانید فایل را بالا یا پایین ببرید.
- با زدن دکمه‌ی Q، از این محیط خارج می‌شوید.
- با زدن دکمه‌ی / و نوشتن متن مورد نظر و سپس زدن دکمه‌ی Enter، می‌تواند عبارت مورد نظر را در این فایل جستجو کنید.

آشنایی با مفهوم Pipe

اگر به عملیاتی که در بالا انجام دادیم نگاه کنید، می‌بینید که اصلا کار ما منطقی نیست. این که اول لیست محتویات یک فولدر را درون یک فایل بریزیم و سپس آن را مشاهده کنیم، بسیار زمان بر و طاقت‌فرسا می‌باشد. اینجاست که باید با مفهومی به نام Pipe آشنا شویم. بسیاری از برنامه‌های لینوکسی، می‌توانند از خروجی یک برنامه در ورودی خود استفاده کنند.

کاری که Pipe انجام می‌دهد، قرار دادن خروجی یک برنامه (STDOUT) درون ورودی یک برنامه‌ی دیگر (STDIN) است. این یعنی ما می‌توانیم خروجی ls -l /etc را درون برنامه‌ی less، پایپ کنیم.

ما با استفاده از کاراکتر |، می‌توانیم عمل پایپینگ را انجام دهیم. به صورت زیر:

```
[root@localhost ~]# ls -l /etc | less
```

همانطور که می‌بینید، الان خروجی ls -l /etc را به راحتی و با کمک برنامه‌ی less، می‌توانیم مشاهده کنیم و بالا و پایین ببریم.

> پس عملگر |، خروجی دستور اول را به ورودی دستور دوم می‌دهد. عمل پایپ کردن در لینوکس بسیار کاربردی می‌باشد و از آن به وفور استفاده می‌کنیم.

نکته: Pipeکردن می‌تواند بین بیش از ۲ دستور نیز صورت گیرد.

ایجاد دستورات Command Line

برنامه‌ی xargs، خروجی یک دستور را گرفته (STDOUT) و سپس آن را به ورودی یک دستور دیگر می‌دهد (STDIN). شاید بپرسید این امر چه تفاوتی با پایپ کردن دارد. برخی از دستورات، خروجی دستورات دیگر را به عنوان ورودی خود قبول نمی‌کنند. بیایید با یک مثال این نکته را امتحان کنیم.

اول بیایید ۳ فایل خالی ایجاد کنیم. برای ایجاد فایل خالی از دستور touch استفاده می‌کنیم:

```
[root@localhost ~]# touch File1.txt File2.txt File3.txt
[root@localhost ~]# ls File?.txt
File1.txt  File2.txt  File3.txt
```

همانطور که می‌بینید، ما با استفاده از دستور touch، سه فایل خالی ایجاد کردیم و سپس با استفاده از دستور ls و ?، این سه فایل را لیست کردیم. حالا فرض کنید ما می‌خواهیم این سه فایل را پاک کنیم. برای این کار، اول باید با دستور rm آشنا شویم. دستور rm، نام یک فایل را از ما می‌گیرد و سپس آن را پاک می‌کند:

```
[root@localhost ~]# rm test
rm: remove regular empty file ‘test’? y
```

همانطور که می‌بینید، دستور rm از ما می‌خواهد که درخواست خود برای پاک‌کردن فایل را تایید کنیم. با نوشتن حرف y و زدن دکمه‌ی enter، این درخواست را تایید می‌کنیم.

حال که با دستور rm آشنا شدیم، می‌توانیم به سراغ صورت مسئله خود برویم. ما می‌خواهیم سه فایلی که ایجاد کردیم را پاک کنیم؛ اما وارد کردن دستی نام هر فایل زمان‌بر است. پس بیایید خروجی دستور ls را درون دستور rm، پایپ کنیم! چقدر ما باهوشیم!

```
[root@localhost ~]# ls File?.txt | rm
rm: missing operand
Try 'rm --help' for more information.
```

همانطور که میبینید، دستور rm متوجه درخواست ما نشده است. این همان امری است که در ابتدای این بخش به آن اشاره کردیم. برخی از دستورها، نمی‌توانند از خروجی یک دستور دیگر به عنوان ورودی استفاده کنند. برای حل این مشکل، ما از دستور xargs استفاده می‌کنیم:

```
[root@localhost ~]# ls File?.txt | xargs rm
```

همانطور که می‌بینید، ما نتیجه‌ی دستور ls را درون برنامه‌ی xargs پایپ کردیم و xargs نیز خروجی را وارد دستور rm کرد و فایل‌های ایجاد شده پاک شدند. اگر توجه کرده باشید، این بار برای پاک کردن فایل‌ها از ما درخواست تایید نکرد.

تمرین: آیا می‌توانید کاری کنید که دستور xargs قبل از اجرای rm، از ما درخواست تایید کند؟

همانطور که دیدید، ما با استفاده از xargs، یک دستور جدید ایجاد کردیم. بدین صورت، ما می‌توانیم لیستی از فایل‌ها را به سادگی پاک کنیم!

روش دیگر برای انجام این چنین کاری، استفاده از کاراکتر ` (backtick) روی کیبورد می‌باشد. این کاراکتر معمولا در کنار دکمه‌ی 1 قرار دارد. بیایید سه فایل دیگر ایجاد کنیم:

```
[root@localhost ~]# touch file1 file2 file3
```

حال برای پاک‌کردن این فایل‌ها، به جای استفاده از xargs و دستور rm، از علامت ` و دستور rm استفاده می‌کنیم:

```
[root@localhost ~]# rm `ls file?`
rm: remove regular empty file ‘file1’? y
rm: remove regular empty file ‘file2’? y
rm: remove regular empty file ‘file3’? y
```

همانطور که می‌بینید، با استفاده از `، دستور rm خروجی ls را به عنوان ورودی خود قبول کرد و به ازای هر فایلی که پیدا کرد، از ما اجازه برای پاک کردن خواست. ما با وارد کردن حرف y، این امر را تایید کردیم.

نکته: اگر بخواهیم دستور rm برای پاک کردن هر فایل از ما سوال نپرسد، از آپشن f- استفاده می‌کنیم.

آشنایی ابتدایی با ادیتور vi و vim

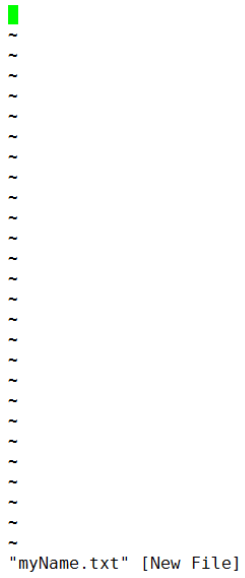
ما تا اینجا زیاد با فایل‌های متنی کار کرده‌ایم و چندین فایل متنی هم با کمک دستور echo و عملگرهای redirection ایجاد کرده‌ایم. اما راه بهتری هم برای ایجاد فایل‌های متنی وجود دارد. در این قسمت می‌خواهیم

به صورت خیلی ابتدایی با ادیتور vi و vim آشنا شویم. کاری با تاریخچه و... این ادیتورها نداریم، چون بعدا بیشتر با آنها آشنا می‌شویم. فعلا فقط می‌خواهیم با استفاده از vi، چندین فایل متنی ایجاد کنیم.

خوب بیایید با کمک vi، یک فایل متنی با نام myName.txt ایجاد کنیم:

```
[root@localhost ~]# vi myName.txt
```

اگر همه چیز به خوبی پیش رفته باشد، باید با صفحه‌ای نظیر تصویر ۱ مواجه شوید:

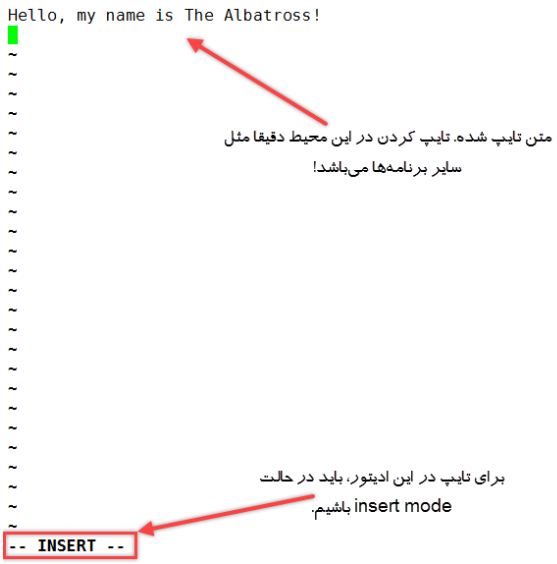


تصویر ۱- در این محیط می‌توانیم درون فایل ایجاد شده متنی بنویسیم.

نکته: اگر دستور vi به شما جوابی نمی‌دهد، از دستور vim استفاده کنید.

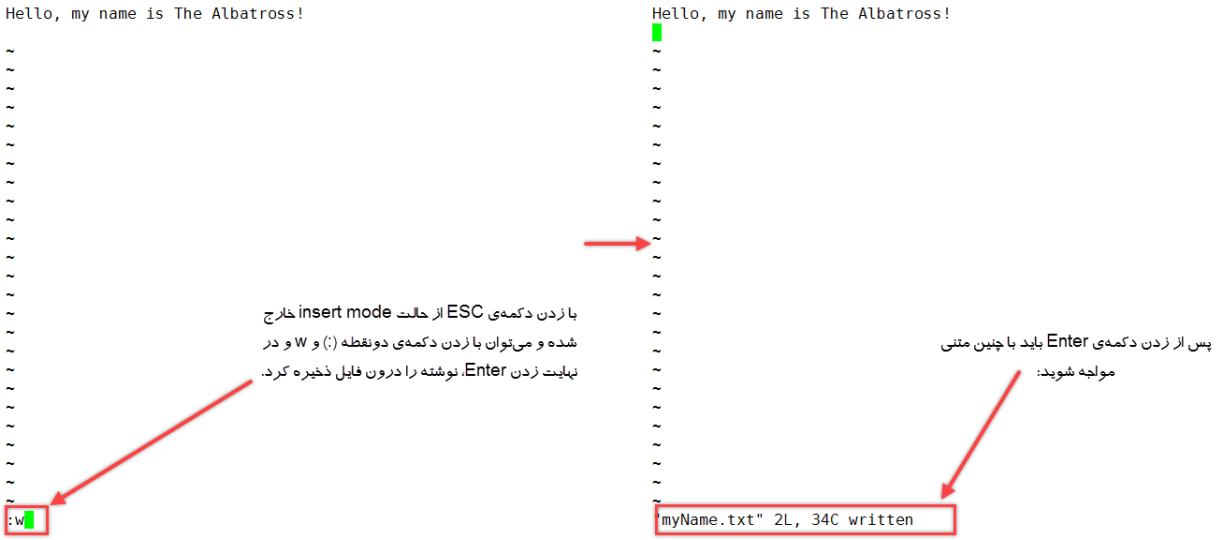
برای این که بتوانید در این برنامه شروع به تایپ کنید، از روی کیبورد خود، دکمه‌ی i را فشار دهید. به محض فشردن دکمه‌ی i، در پایین صفحه کلمه‌ی **-- INSERT --** ظاهر می‌شود و این یعنی شما در حالت insert mode قرار دارید و می‌توانید متن دلخواه خود را تایپ کنید.

خوب، بیایید اسم خود را درون vi تایپ کنیم:



تصویر ۲- حالت insert mode و تایپ کردن متن در vi

خوب، حال که متن خود را نوشتیم، باید بتوانیم آن را درون فایل خود (myName.txt) ذخیره کنیم. برای این کار، ابتدا دکمه‌ی ESC را از روی کیبورد خود فشار دهید تا از حالت insert mode خارج شوید. سپس دکمه‌ی دونقطه (:) را فشار دهید و حرف w را بنویسید و سپس دکمه‌ی Enter را بزنید:



تصویر ۳ - نحوه‌ی ذخیره کردن فایل در vi

حال که فایل ما ذخیره شده، باید از vi بیرون آییم. برای این کار، از روی کیبورد دکمه‌ی دونقطه (:) را فشار دهید و سپس حرف q را بنویسید و Enter را بزنید. اگر همه چیز به درستی پیش رفته باشد، باید به shell بازگشته باشید و باید بتوانید فایلی که ایجاد کردید را مشاهده کنید:

```
[root@localhost ~]# cat myName.txt
Hello, my name is The Albatross!
```

نکته: با وارد کردن حرف wq پس از علامت دو نقطه (:)، می‌توانید در یک مرحله هم فایل را ذخیره کنید و هم از vi بیرون آیید.

به هم چسباندن فایل‌های متنی

قبلا گفتیم که دستور cat، قابلیت به هم چسباندن فایل‌های متنی را نیز دارد. ما با استفاده از ادیتور vi، نام خود را درون یک فایل، و نام خانوادگی خود را درون یک فایل دیگر قرار داده‌ایم. حال می‌خواهیم با استفاده از دستور cat، آنها را به هم بچسبانیم.

برای چسباندن دو فایل متنی به هم، کافی است به صورت زیر عمل کنید:

```
[root@localhost ~]# cat firstName.txt
Behnam
[root@localhost ~]# cat lastName.txt
Sajjadi
[root@localhost ~]# cat firstName.txt lastName.txt
Behnam
Sajjadi
```

همانطور که می‌بینید، با استفاده از دستور cat، دو فایل firstName.txt و lastName.txt را به هم چسباندیم. ما می‌توانیم خروجی به هم چسبانده شده را با استفاده از redirectorها درون یک فایل جدید قرار دهیم؛ اما این کار را به عهده‌ی شما می‌سپاریم.

نکته: دستور cat قابلیت‌های دیگری نیز دارد. برای مثال، آپشن v- می‌تواند کاراکترهای مخفی (non-printable characters) موجود در یک فایل را پیدا کند. خیلی اوقات ممکن است فایلی در سیستم داشته باشید که رفتار عجیبی از خود نشان می‌دهد، اما با مشاهده‌ی فایل، نکته‌ی غیر معمولی در مورد آن مشاهده نکنید. با استفاده از آپشن v-، می‌توانید کاراکترهای مخفی موجود در آن فایل را پیدا کنید و منبع مشکل را پیدا کنید.

همانطور که می‌بینید، دستور cat، فایل‌های متنی را زیر هم قرار داد. اگر بخواهیم فایل‌های متنی کنار هم قرار گیرند، از دستور paste استفاده می‌کنیم:

```
[root@localhost ~]# paste firstName.txt lastName.txt
Behnam Sajjadi
```

اگر دو فایل متفاوت دارای یک فیلد مساوی باشند، می‌توانیم این دو فایل را با دستور join در کنار هم قرار دهیم. فیلدها، معمولا یک تکه نوشته هستند که در یک خط با یک فاصله‌ی خالی (به صورت پیش‌فرض) از هم جدا شده‌اند. برای مثال، فرض کنید دو فایل داریم: یکی از فایل‌ها شماره دانشجویی دانشجویان و نام آنها، و دیگری شماره دانشجویی و وضعیت مالی آنها را درون خود دارد:

```
[root@localhost ~]# cat studentName
8008135    Behnam
6969420    Abbas
5318008    Sepehr
0249696    Milad
[root@localhost ~]# cat studentFinance
8008135    BAD
6969420    SAD
5318008    NO
0249696    WHY
```

همانطور که می‌بینید، این دو فایل در فیلد شماره دانشجویی، مشترک هستند. پس برای کنار هم قرار دادن این دو فایل، از دستور join استفاده می‌کنیم:

```
[root@localhost ~]# join studentName studentFinance
8008135 Behnam BAD
6969420 Abbas SAD
5318008 Sepehr NO
0249696 Milad WHY
```

همانطور که می‌بینید، دستور join این دو فایل را با توجه به فیلد مشترکشان در کنار هم قرار داد.

تبدیل فایل‌ها به اعداد هشت‌هشتی

ما می‌توانیم فایل‌های موجود در سیستم را با استفاده از دستور od، تبدیل به اعداد هشت‌هشتی کنیم یا به عبارتی دیگر، octal dump آنها را در خروجی دریافت کنیم:

```
[root@localhost ~]# od myName.txt
0000000 062510 066154 026157 066440 020171 060556 062555 064440
0000020 020163 064124 020145 066101 060542 071164 071557 020563
0000040 005012
0000042
```

همانطور که می‌بینید، این دستور فایل متنی ما را تبدیل به اعداد هشت‌هشتی کرد.

ممکن است از خود بپرسید که این دستور به چه درد می‌خورد. کاربرد اصلی این دستور برای کارهای forensics می‌باشد. مثلا ممکن است یک فایل به نظر معمولی داشته باشیم که در حال ایجاد مشکل در سیستم می‌باشد. ما با استفاده از octal dump آن فایل، می‌توانیم ببینیم که آن فایل دقیقا چه چیزی درون خود دارد. لازم به ذکر است که od می‌تواند خروجی‌های دیگر، نظر خروجی hex نیز به ما بدهد.

تمرین: آیا می‌تواند با مراجعه به manpage این دستور، کاری کنید که این دستور به شما خروجی hex بدهد؟

مشاهده‌ی فایل‌های متنی با دستور head و tail

یکی از ابزارهای کاربردی برای مشاهده‌ی بخشی از فایل‌های متنی، دستور head می‌باشد. این دستور به صورت پیش‌فرض، ده خط اول یک فایل را به ما نشان می‌دهد. برای مثال:

```
[root@localhost ~]# head /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
operator:x:11:0:operator:/root:/sbin/nologin
```

همانطور که می‌بینید، با استفاده از این دستور، فقط ۱۰ خط اول فایل /etc/passwd را نشان دادیم.

نکته: بعدا با فایل /etc/passwd بیشتر آشنا خواهیم شد. این فایل، لیستی از اطلاعات کلیه کاربران سیستم را به ما نشان می‌دهد.

با استفاده از آپشن n-، می‌توانیم تعداد خطوط نمایش داده شده توسط head را تغییر دهیم. مثلا اگر بخواهیم فقط سه خط اول فایل /etc/passwd به ما نشان داده شود:

```
[root@localhost ~]# head -n 3 /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
```

```
[root@localhost ~]# head -3 /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
```

با نگاه به دو دستور بالا، می‌بینید که به جای استفاده از n- و سپس وارد کردن تعداد خطوط مورد نظر، می‌توانیم صرفا با استفاده از - و عدد مورد نظر، تعداد خطوط مشاهده شده را تغییر دهیم.

یکی دیگر از ابزارهای کاربری برای مشاهده‌ی فایل‌های متنی، دستور tail می‌باشد. این دستور، به صورت پیش‌فرض، ده خط پایانی یک فایل را به ما نشان می‌دهد:

```
[root@localhost ~]# tail /etc/passwd
operator:x:11:0:operator:/root:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
nobody:x:99:99:Nobody:/:/sbin/nologin
systemd-network:x:192:192:systemd Network Management:/:/sbin/nologin
dbus:x:81:81:System message bus:/:/sbin/nologin
polkitd:x:999:998:User for polkitd:/:/sbin/nologin
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
postfix:x:89:89::/var/spool/postfix:/sbin/nologin
tss:x:59:59:used by tcsd daemon:/dev/null:/sbin/nologin
```

نکته: دقیقا مثل قبل، شما با استفاده از آپشن n-، می‌توانید تعداد خطوط نمایش داده شده را تغییر دهید.

یکی از کاربردی‌ترین آپشن‌های این دستور، آپشن f- می‌باشد. این آپشن به ما کمک می‌کند که تغییراتی که روی یک فایل اعمال می‌شود را به صورت زنده ببینیم. این امر، برای مشاهده‌ی logfileها بسیار کارآمد می‌باشد، چون می‌تواند به ما در diagnoseکردن سیستم و سرویس‌های آن کمک کند:

```
[root@localhost ~]# tail -f /var/log/maillog
Mar 28 11:46:03 localhost postfix/postfix-script[1478]: starting the
Postfix mail system
Mar 28 11:46:03 localhost postfix/master[1480]: daemon started --
version 2.10.1, configuration /etc/postfix
Mar 28 11:46:10 localhost postfix/postfix-script[1495]: stopping the
Postfix mail system
Mar 28 11:46:10 localhost postfix/master[1480]: terminating on
signal 15
Mar 28 11:46:18 localhost postfix/postfix-script[1578]: starting the
Postfix mail system
Mar 28 11:46:18 localhost postfix/master[1580]: daemon started --
version 2.10.1, configuration /etc/postfix
^C
```

نکته: برای خروج از وضعیت مشاهده‌ی فایل با استفاده از tail -f، باید دکمه‌ی Ctrl + C را فشار دهید.

شماره‌گذاری خطوط یک فایل با استفاده از nl

یکی از دستورهای جالب برای کار با فایل، دستور nl می‌باشد. این دستور، خط‌های موجود درون یک فایل را شماره‌گذاری می‌کند. مثلا:

```
[root@localhost ~]# cat ball
i
have
a
ball

which
is
round-y round
```

```
[root@localhost ~]# nl ball
     1      i
     2      have
     3      a
     4      ball

     5      which
     6      is
     7      round-y round

     8      it's
     9      red
    10      and
    11      white
    12      and
    13      blue
```

همانطور که می‌بینید، دستور nl، تک‌تک خطوط غیر خالی فایل ما را شماره‌گذاری کرد.

برای این که این دستور خطوط خالی را نیز شماره‌گذاری کند، باید از آپشن ba- استفاده کنید. همچنین می‌توان خروجی دستور nl را نیز درون یک فایل ریخت. مثلا:

```
[root@localhost ~]# nl -ba ball > numberedBall
```

حال بیایید با هم یک معما حل کنیم. با توجه به آموخته‌های خود، می‌توانید بگویید دستور زیر چه چیزی در خروجی به ما می‌دهد؟

```
[root@localhost ~]# head numberedBall | tail -3
```

توجه کنید که ما به این دستور، فایلی که تمامی خطوط آن شماره‌گذاری شده بود را دادیم.

مرتب‌سازی محتویات یک فایل با sort

دستور sort، می‌تواند بدون اعمال هیچ تغییری در خود فایل، محتویات آن را مرتب کند. برای مثال:

```
[root@localhost ~]# cat names
Mohammad
Ali
Behnam
Milad
Abbas
[root@localhost ~]# sort names
Abbas
Ali
Behnam
Milad
Mohammad
```

همانطور که می‌بینید، این دستور محتویات فایل names را گرفت و آنها را بر اساس حروف الفبا، مرتب کرد.

اگر بخواهید فایلی دارای اعداد می‌باشد را با استفاده از sort مرتب کنید، جواب صحیحی نخواهید گرفت:

```
[root@localhost ~]# sort numbers
11
350
420
45
69
73
98
```

برای این که دستور sort بتواند اعداد را به شکل صحیح مرتب کند، از آپشن n- استفاده می‌کنیم:

```
[root@localhost ~]# sort -n numbers
11
45
69
73
98
350
420
```

شمارش محتویات فایل با wc

با استفاده از دستور wc، می‌توانید تعداد خطوط، تعداد کلمات و تعداد بایت‌های موجود در یک فایل را به دست آورید:

```
[root@localhost ~]# wc file
 19  21 101 file
```

همانطور که می‌بینید، دستور wc به ما می‌گوید که در file، ۱۹ خط (یا به عبارتی دیگر، newline)، ۲۱ کلمه و ۱۰۱ بایت داریم.

> این دستور می‌تواند اطلاعات دیگری نظیر تعداد کاراکترهای موجود در فایل و... را نیز به ما بگوید. اطلاعات بیشتر را می‌توانید از manpage این دستور به دست آورید.

آشنایی با grep

grep یکی از ابزارهای بسیار قوی برای جستجو و فیلتر کردن متن می‌باشد. ابزار grep، در ساده‌ترین حالت خود، می‌تواند یک کلمه را درون یک فایل سرچ کرده و هر خطی که شامل آن کلمه باشد را به ما نشان دهد. برای مثال، فایل COPYING، که در موقعیت زیر قرار دارد را در نظر بگیرید:

```
[root@localhost ~]# cd /usr/share/licenses/gmp-6.0.0
[root@localhost gmp-6.0.0]# cat COPYING
...
  The GNU General Public License does not permit incorporating your program
into proprietary programs.  If your program is a subroutine library, you
may consider it more useful to permit linking proprietary applications with
the library.  If this is what you want to do, use the GNU Lesser General
Public License instead of this License.  But first, please read
<http://www.gnu.org/philosophy/why-not-lgpl.html>
```

این فایل که حاوی لایسنس GPL نسخه‌ی ۳ می‌باشد، بسیار طولانی است. فرض کنید ما فقط دنبال خطوطی هستیم که در آن کلمه‌ی GNU وجود داشته باشد. برای این کار به سراغ grep می‌رویم:

```
[root@localhost gmp-6.0.0]# grep "GNU" COPYING
                    GNU GENERAL PUBLIC LICENSE
  The GNU General Public License is a free, copyleft license for
the GNU General Public License is intended to guarantee your freedom to
GNU General Public License for most of our software; it applies also to
  Developers that use the GNU GPL protect your rights with two steps:
  "This License" refers to version 3 of the GNU General Public License.
  13. Use with the GNU Affero General Public License.
...
```

همانطور که می‌بینید، نرم‌افزار grep، کلیه‌ی خطوطی که در آن کلمه‌ی GNU وجود داشت را در خروجی به ما نشان داد. ما ابتدا به grep می‌گوییم که دنبال چه کلمه‌ای هستیم و سپس فایلی که می‌خواهیم در آن جستجو کند را به grep نشان می‌دهیم.

نکته: قرار دادن عبارت مورد نظر بین "" لازم نیست، اما شدیدا پیشنهاد می‌شود.

grep در حالت پیش‌فرض خود، case sensitive می‌باشد؛ یعنی بین حرف بزرگ و حرف کوچک در جستجوی خود فرق قائل می‌شود. با استفاده از آپشن i-، می‌توانیم به grep بگوییم که کاری به بزرگ یا کوچک بودن حروف نداشته باشد و هر حرف و کلمه‌ی را در هر حالتی به ما نشان دهد. مثلا فرض کنید در همان فایل COPYING، به دنبال کلیه‌ی خط‌هایی باشیم که در آن کلمه‌ی license آمده باشد. اگر به خروجی این فایل در مثال بالا نگاه کنید، می‌بینید که در خط اول، کلمه‌ی license به صورت LICENSE و در خطوط بعدی این کلمه به صورت License نوشته شده است. اینجاست که ما از آپشن i- دستور استفاده می‌کنیم:

```
[root@localhost gmp-6.0.0]# grep -i "license" COPYING
                    GNU GENERAL PUBLIC LICENSE
 of this license document, but changing it is not allowed.
  The GNU General Public License is a free, copyleft license for
  The licenses for most software and other practical works are designed
the GNU General Public License is intended to guarantee your freedom to
GNU General Public License for most of our software; it applies also to
price.  Our General Public Licenses are designed to make sure that you
(1) assert copyright on the software, and (2) offer you this License
...
```

همانطور که می‌بینید، آپشن i- باعث شد که grep بدون توجه به بزرگی یا کوچکی حروف، درون فایل جستجو کند و هم کلمه‌ی LICENSE، license و License را به ما نشان دهد.

ما می‌توانیم از grep بخواهیم که خطوطی را به ما نشان دهد که یک کلمه یا حرف خاص درون آن وجود **نداشته باشد**. برای این کار، از آپشن v- استفاده می‌کنیم. مثلا فرض کنید در همان فایل COPYING، دنبال خطوطی باشیم که کلمه‌ی the در آن وجود **نداشته باشد**:

```
[root@localhost gmp-6.0.0]# grep -v "the" COPYING
                    GNU GENERAL PUBLIC LICENSE
                       Version 3, 29 June 2007

 Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/>
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.
```

```
                    Preamble

  The GNU General Public License is a free, copyleft license for

share and change all versions of a program--to make sure it remains free
GNU General Public License for most of our software; it applies also to
your programs, too.
...
```

همانطور که می‌بینید ما در خروجی هنوز خطوطی که شامل The هستند را داریم. این بدین دلیل است که ما از آپشن i- استفاده نکردیم. با این حال اگر خروجی این متن را با خروجی‌های قبلی مقایسه کنید، خواهید دید که خطوطی که در آن دقیقا کلمه‌ی the آمده بوده، حذف شده است. البته لازم نیست این کار را بکنید. کافی است از grep بخواهیم که خطوط خروجی را شماره گذاری کند. برای این کار، از آپشن n- استفاده می‌کنیم:

```
[root@localhost gmp-6.0.0]# grep -vn "the" COPYING
...
8:                              Preamble
9:
10:  The GNU General Public License is a free, copyleft license for
12:
16:share and change all versions of a program--to make sure it remains free
18:GNU General Public License for most of our software; it applies also to
20:your programs, too.
21:
22:  When we speak of free software, we are referring to freedom, not
23:price.  Our General Public Licenses are designed to make sure that you
28:
33:
39:
42:giving you legal permission to copy, distribute and/or modify it.
43:
...
```

همانطور که می‌بینید، خط ۱۲ به صورت غیر منطقی خالی است و درچند خط پایین تر نیز همین وضعیت وجود دارد. این یعنی در جایی در این خطوط، کلمه‌ی the وجود داشته است.

نکته: آپشن n- بسیار کاربردی است، چون خیلی از اوقات دانستن شماره خطی که یک کلمه‌ی خاص در آن قرار گرفته بسیار به کار می‌آید.

با استفاده از آپشن r-، می‌توانیم به grep بگوییم که کلمه‌ی مشخص شده را داخل همه‌ی فایل‌ها و فولدرهای موجود در یک موقعیت، جستجو کند. مثلا فرض کنید می‌خواهیم grep در همه‌ی فایل‌ها یا فولدرهای موجود در موقعیت /usr/share/licenses، به دنبال کلمه‌ی misrepresentation بگردد. برای این کار:

```
[root@localhost ~]# grep -r "misrepresentation" /usr/share/licenses/
/usr/share/licenses/grub2-common-2.02/COPYING:    c) Prohibiting
misrepresentation of the origin of that material, or
/usr/share/licenses/gmp-6.0.0/COPYING:    c) Prohibiting misrepresentation
of the origin of that material, or
/usr/share/licenses/gmp-6.0.0/COPYINGv3:    c) Prohibiting
misrepresentation of the origin of that material, or
/usr/share/licenses/binutils-2.27/COPYING3:    c) Prohibiting
misrepresentation of the origin of that material, or
```

همانطور که می‌بینید، در خروجی، مسیر فایل و خطی که در آن کلمه‌ی misrepresentation آمده مشخص شده است. پرواضح است که این ویژگی grep، بسیار کاربردی می‌باشد.

ما می‌توانیم خروجی دستورهای دیگر را نیز درون grep، پایپ کنیم. برای مثال، فرض کنید میخواهیم ببینیم که آیا در فولدر /usr/share، فولدری به نام licenses وجود دارد یا نه:

```
[root@localhost ~]# ls -l /usr/share/ | grep licenses
drwxr-xr-x.  66 root root  4096 Mar 24 12:29 licenses
```

همانطور که می‌بینید، ما خروجی دستور ls را درون grep پایپ کردیم و بدین طریق، در بین فایل‌ها و فولدرهای موجود در /usr/share، دنبال فولدرها یا فایل‌هایی که در نام آنها کلمه‌ی licenses وجود داشت، گشتیم.

Regular Expressions یا regexها

regexها الگوهایی هستند که به یک برنامه، جهت استخراج اطلاعات خاص از متن می‌دهیم. به عبارت دیگر، regex یک سری الگو برای جستجو در متن می‌باشند. regexها فقط محدود به لینوکس نیستند، بسیاری از زبان‌های برنامه‌نویسی نظیر Python، Perl، Java و... نیز از regex استفاده می‌کنند. اما regex چیست؟

regex در ساده‌ترین حالت خود، همان کلمه یا حرف‌هایی می‌باشد که درون یک فایل جستجو می‌کنیم. مثلا در بخش قبل، دیدید که چگونه با استفاده از grep، توانستیم خطوطی که یک کلمه‌ی خاص را درون خود داشتند، بیرون بکشیم. اما برای این که regex را بهتر درک کنیم، باید به حروف یا کلماتی که به grep می‌دهیم، به عنوان رشته‌ای از حروف نگاه کنیم.

الگوهایی که دقیقا می‌گویند دنبال چه حرف یا کلمه‌ای هستند، literal نامیده می‌شوند، چون دقیقا به سراغ حروف و کلمه‌ی مشخص شده می‌روند. همه‌ی حروف الفبا و اعداد به صورت literal در نظر گرفته می‌شوند، مگر این که آنها را با مکانیزم‌های دیگری اصلاح کنیم.

علامت ^ و $

^ و $ علامت‌های ویژه‌ای هستند که مشخص می‌کنند یک الگو باید در کجای خط باشد تا خروجی به ما نشان داده شود. ما با استفاده از علامت ^ می‌توانیم به سیستم بگوییم که دنبال خطهایی بگردد با یک کاراکتر یا رشته‌ی خاص، **شروع می‌شوند**.

مثلا فرض کنید در فایل COPYING فقط به دنبال خطهایی باشیم که با کلمه‌ی GNU شروع می‌شوند:

```
[root@localhost gmp-6.0.0]# grep "^GNU" COPYING
GNU General Public License for most of our software; it applies also to
GNU General Public License, you may choose any version ever published
```

همانطور که می‌بینید، با قرار دادن کاراکتر ^ قبل از کلمه‌ی GNU، فقط خطوطی که با کلمه‌ی GNU شروع شده‌اند را در خروجی می‌بینیم.

> **نکته:** اگر به خود فایل COPYING نگاه کنید، می‌بینید که جمله‌های دیگری هم هستند که با GNU شروع شده باشند. دلیل این که الگوی بالا آنها را در خروجی نشان نداد این است که آن خطوط، در ابتدای خود خط فاصله داشتند. پس در regex، خود خط فاصله نیز به عنوان یک کاراکتر در نظر گرفته می‌شود.

با استفاده از کاراکتر $، می‌توانیم به سیستم بگوییم که دنبال خط‌هایی بگردد که با یک کاراکتر یا یک رشته‌ی خاص، **تمام می‌شوند**.

مثلا فرض کنید در فایل COPYING فقط به دنبال خط‌هایی باشیم که با کلمه‌ی and تمام می‌شود:

```
[root@localhost gmp-6.0.0]# grep "and$" COPYING
that there is no warranty for this free software.  For both users' and
  The precise terms and conditions for copying, distribution and
License.  Each licensee is addressed as "you".  "Licensees" and
receive it, in any medium, provided that you conspicuously and
    alternative is allowed only occasionally and noncommercially, and
network may be denied when the modification itself materially and
adversely affects the operation of the network or violates the rules and
provisionally, unless and until the copyright holder explicitly and
receives a license from the original licensors, to run, modify and
make, use, sell, offer for sale, import and otherwise run, modify and
```

همانطور که می‌بینید با قرار دادن علامت $ پس از کلمه‌ی and، فقط خطوطی که با and تمام شده‌اند را در خروجی می‌بینیم.

جستجو برای هر تک کاراکتر

با استفاده از علامت نقطه (.)، می‌توانیم به سیستم بگوییم که در این موقعیت، هرکاراکتری به صورت تکی می‌تواند وجود داشته باشد. فرض کنید می‌خواهیم به سیستم بگوییم که خطوطی را برایمان پیدا کند که با هر دو کاراکتری می‌تواند شروع شود، اما پس از دو کاراکتر، باید حروف cept وجود داشته باشد:

```
[root@localhost gmp-6.0.0]# grep "..cept" COPYING
use, which is precisely where it is most unacceptable.  Therefore, we
infringement under applicable copyright law, except executing it on a
tells the user that there is no warranty for the work (except to the
License by making exceptions from one or more of its conditions.
form of a separately written license, or stated as exceptions;
  You may not propagate or modify a covered work except as expressly
  9. Acceptance Not Required for Having Copies.
  You are not required to accept this License in order to receive or
to receive a copy likewise does not require acceptance.  However,
not accept this License.  Therefore, by modifying or propagating a
covered work, you indicate your acceptance of this License to do so.
public statement of acceptance of a version permanently authorizes you
```

همانطور که می‌بینید، با استفاده از علامت نقطه، به سیستم گفتیم که دنبال رشته کاراکترهایی بگردد که دو کاراکتر اول آنها **هرچیزی** می‌تواند باشد، اما آن دو کاراکتر، حتما باید حروف cept را در ادامه‌ی خود داشته باشند. همانطور که می‌بینید با توجه به شرایط جستجوی ما، سیستم هر خطی که در آن رشته‌های accept، Accept و except وجود داشته را به ما نشان داده است. این رشته‌ها، دو کاراکتر اولشان با هم متفاوت است، اما حروف cept، در ادامه‌ی این دو کاراکتر آمده‌اند. مثلا اگر در متن 3Pcept هم داشتیم، در خروجی به ما نمایش داده می‌شد.

کروشه‌ها (براکت‌ها)

با قرار دادن گروهی از کاراکترها درون یک کروشه، ما به سیستم می‌گوییم که کاراکتر موجود در این موقعیت، می‌تواند **یکی** از کاراکترهای موجود در کروشه باشد. مثلا فرض کنید می‌خواهیم به سیستم بگوییم که خطوطی که در آن کلمه‌ی too، یا two وجود دارد را به ما نشان دهد:

```
[root@localhost gmp-6.0.0]# grep "t[wo]o" COPYING
your programs, too.
freedoms that you received.  You must make sure that they, too, receive
  Developers that use the GNU GPL protect your rights with two steps:
a computer network, with no transfer of a copy, is not conveying.
System Libraries, or general-purpose tools or generally available free
    Corresponding Source from a network server at no charge.
    copy the object code is a network server, the Corresponding Source
network may be denied when the modification itself materially and
adversely affects the operation of the network or violates the rules and
protocols for communication across the network.
publicly available network server or other readily accessible means,
section 13, concerning interaction through a network will apply to the
```

همانطور که می‌بینید، با استفاده از الگوی t[wo]o به سیستم گفتیم که هر رشته کاراکتری که با t شروع می‌شود، کاراکتر دوم آن w **یا** o می‌باشد و کاراکتر سوم آن، o می‌باشد را به ما نشان دهد. به همین دلیل سیستم هم رشته‌ی two، و هم رشته‌ی too را به ما نشان داد.

یکی دیگر از کارهایی که با کروشه می‌توان انجام داد، استفاده از علامت ^ **درون کروشه** می‌باشد. با استفاده از ^ درون کروشه، به سیستم می‌گوییم که در این موقعیت هر کاراکتری **به جز** کاراکتر موجود در کروشه می‌تواند قرار گیرد. برای مثال:

```
[root@localhost gmp-6.0.0]# grep "[^c]ode" COPYING
  1. Source Code.
    model, to give anyone who possesses the object code either (1) a
the only significant mode of use of the product.
notice like this when it starts in an interactive mode:
```

همانطور که می‌بینید با قرار دادن ^ در ابتدای کاراکتر درون کروشه، به سیستم گفتیم که هر رشته کاراکتری که با هر کاراکتری **به جز** c شروع می‌شود و کاراکترهای بعدی آن ode می‌باشند را به ما نشان دهد. چند نکته در مورد خروجی بالا ممکن است شما را سردرگم کند:

- در خط اول، می‌بینید که سیستم رشته‌ی Code را به ما نشان داده است. همانطور که قبلا گفتیم، regexها و grep، case sensitive می‌باشند. به عبارت دیگر، c و C دو کاراکتر متفاوت می‌باشند. اگر از آپشن i- دستور grep استفاده می‌کردیم، رشته‌ی Code نیز به ما نشان داده نمی‌شد.
- در خط دوم، رشته‌ی code قابل مشاهده است. این مشکل regex یا grep نیست؛ بلکه دلیل آن این است که از قضای روزگار، در همان خطی که رشته‌ی mode وجود داشته، رشته‌ی code نیز وجود داشته است. همانطور که می‌بینید، بر خلاف mode، رشته‌ی code توسط grep، مشخص نشده است.

یکی دیگر از ویژگی‌های خوب کروشه این است که به ما اجازه می‌دهد محدوده‌ای از کاراکترها را مشخص کنیم. مثلا فرض کنید میخواهیم به سیستم بگوییم که تمام خطوطی که با حرف A، B، C و... تا Z شروع می‌شوند را به ما نشان دهد. به جای این که تک تک این کاراکترها را درون کروشه تایپ کنیم، می‌توانیم آنها را به صورت زیر مشخص کنیم:

```
[root@localhost gmp-6.0.0]# grep  "^[A-Z]" COPYING
GNU General Public License for most of our software; it applies also to
States should not allow patents to restrict development and use of
License.  Each licensee is addressed as "you".  "Licensees" and
Component, and (b) serves only to enable use of the work with that
Major Component, or to implement a Standard Interface for which an
System Libraries, or general-purpose tools or generally available free
Source.
User Product is transferred to the recipient in perpetuity or for a
...
```

همانطور که می‌بینید، با استفاده از علامت - درون کروشه، به سیستم گفتیم که در این موقعیت، هر کاراکتری که بین A تا Z باشد را به ما نشان دهد. شاید استفاده از ^، با توجه به چیزی که در مثال قبل دیدید، برایتان عجیب باشد.

> همانطور که گفتیم، استفاده از ^ **درون کروشه**، به سیستم می‌گوید که دنبال هر کاراکتری به جز کاراکتر درون کروشه باشد. اما استفاده از ^ **بیرون کروشه**، معنی وجود در ابتدای خط را دارد.

بهتر است زمانی که می‌خواهیم محدوده‌های مشخصی از کاراکترها، نظیر کلیه‌ی حروف الفبا، کلیه‌ی حروف بزرگ یا کوچک و... را مشخص کنیم، از کلاس‌های کاراکتری پازیکس (POSIX Character Classes) استفاده کنیم. صحبت در مورد این کلاس‌های کاراکتری از حوصله‌ی ما خارج است، اما مثال بالا با استفاده از این کلاس کاراکتری، به شکل زیر می‌شود:

```
[root@localhost gmp-6.0.0]# grep "^[[:upper:]]" COPYING
GNU General Public License for most of our software; it applies also to
States should not allow patents to restrict development and use of
License.  Each licensee is addressed as "you".  "Licensees" and
Component, and (b) serves only to enable use of the work with that
Major Component, or to implement a Standard Interface for which an
System Libraries, or general-purpose tools or generally available free
Source.
User Product is transferred to the recipient in perpetuity or for a
...
```

همانطور که می‌بینید، ما درون کروشه، کلاس کاراکتری [:upper:] را قرار دادیم که همان معنی A-Z را می‌دهد. برای اطلاعات بیشتر در مورد POSIX Character Classes، می‌توانید به این لینک مراجعه کنید.

علامت *

یکی دیگر از علامت‌هایی که در regex می‌توان از آن استفاده کرد، علامت * می‌باشد. ما با استفاده از این علامت به سیستم می‌گوییم که به دنبال خط‌هایی بگردد که در آن، کاراکتر موجود در **قبل** از علامت *، **صفر** یا **هر چند بار** تکرار شده باشد.

> منظور از صفر بار تکرار، یعنی اگر کاراکتری که قبل از علامت * آمده اصلا در رشته وجود نداشته باشد، باز هم سیستم به ما خروجی خواهد داد.

شاید درک عملکرد این علامت کمی برایتان دشوار باشد. پس بیایید یک مثال بزنیم. فرض کنید ما از سیستم می‌خواهیم که به دنبال الگوی زیر بگردد:

```
[root@localhost gmp-6.0.0]# grep "ab*c" test.txt
```

این الگو، رشته‌هایی را که با a آغاز می‌شوند، اما پس از کاراکتر a، **صفر**، یا **هر تعداد** کاراکتر b دارند و پس از آن تعداد b، حتما کاراکتر c دارند را به ما نشان می‌دهد.

این یعنی الگوی بالا، رشته‌های زیر را به ما نشان می‌دهد:

```
ac
abc
abbc
a[b هرتعداد]c
....
```

حال بیایید کمی کار را پیچیده‌تر کنیم. فرض کنید در فایل COPYING، به دنبال رشته‌هایی هستیم که با پرانتز شروع و تمام می‌شوند؛ به طوری که درون آن پرانتز هر کدام از حروف الفبا و هر تعداد فاصله خالی (space) می‌تواند وجود داشته باشد:

```
[root@localhost gmp-6.0.0]# grep "([A-Za-z ]*)" COPYING
 Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/>
distribution (with or without modification), making available to the
than the work as a whole, that (a) is included in the normal form of
Component, and (b) serves only to enable use of the work with that
(if any) on which the executable work runs, or a compiler used to
    (including a physical distribution medium), accompanied by the
    (including a physical distribution medium), accompanied by a
    place (gratis or for a charge), and offer equivalent access to the
    may be on a different server (operated by you or a third party)
...
```

خوب، بیایید الگوی بالا را بررسی کنیم. همانطور که می‌بینید، طبق صورت مسئله، ابتدا با قرار دادن یک پرانتز باز در ابتدای الگو و قرار دادن یک پرانتز در انتهای الگو، به صورت literal مشخص می‌کنیم که دنبال پرانتز در شروع و پایان رشته هستیم. سپس درون پرانتز، یک کروشه داریم. درون کروشه، ما به سیستم گفتیم که دنبال هر رشته‌ای که شامل یک کاراکتر، که می‌تواند از بین A-Z **یا** a-z **یا** فاصله خالی (space) باشد، بگردد. سپس خارج از کروشه، علامت * را قرار دادیم. علامت *، به سیستم می‌گوید که به دنبال رشته‌هایی بگردد که الگوی درون کروشه در آن صفر یا هر چند بار تکرار شده باشد.

همانطور که می‌بینید، الگوی ما اول رشته‌ی (C) را به ما نشان داد؛ چرا که این رشته درون پرانتز است، و حرف C درون آن قرار دارد. اما هیچ فاصله خالی (space) ندارد. همانطور که در بالا گفتیم، علامت * به دنبال رشته‌هایی می‌رود که کاراکتر قبلی (یا در اینجا، الگوی قبلی) در آن صفر یا چند بار تکرار شده باشد. در اینجا الگوی قبلی یک بار تکرار شده است. فراموش نکنید که الگوی درون براکت به دنبال A-Z یا a-z یا فاصله خالی می‌گردد، که در اینجا، فقط کاراکتری بین A-Z پیدا کرده است و در نتیجه آن را به ما باز گردانده است.

در خط بعدی نیز رشته‌ی (with or without modifcation) به ما نشان داده شده است، اما کشف دلیل نشان دادن این رشته را به شما می‌سپاریم.

Escape Characterها

تا به اینجا با علامت‌هایی نظیر *، ^، . و... آشنا شدیم. حال فرض کنید می‌خواهیم در متن، دنبال رشته‌ای بگردیم که درون خود از علامت نقطه (.) یا سایر علامت‌ها استفاده می‌کند. از آنجا که سیستم در حالت نرمال با این علامت‌ها به صورت خاص برخورد می‌کند، جهت جستجو برای این کاراکترها، باید آنها را اصطلاحا escape کنیم. یعنی باید به سیستم بگوییم که با این علامت، به عنوان یک علامت خاص رفتار نکند.

برای escape کردن یکی از این علامت‌های خاص، باید قبل از آن علامت، یک \ قرار دهیم. فرض کنید می‌خواهیم دنبال هر خطی بگردیم که با یکی از حروف بزرگ الفبا شروع می‌شود و با یک نقطه تمام می‌شود:

```
[root@localhost gmp-6.0.0]# grep "^[A-Z].*\.$" COPYING
Source.
License by making exceptions from one or more of its conditions.
License would be to refrain entirely from conveying the Program.
ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
SUCH DAMAGES.
Also add information on how to contact you by electronic and paper mail.
```

همانطور که می‌بینید، با استفاده از علامت ^ و [A-Z]، به سیستم گفتیم که دنبال خطوطی باشد که در ابتدای آن یک حرف بزرگ قرار دارد. سپس از کروشه خارج شدیم و با علامت نقطه (.) به سیستم گفتیم که در این موقعیت، هر کاراکتری می‌تواند قرار گیرد. سپس با استفاده از *، به سیستم گفتیم که ویژگی علامت نقطه، می‌تواند صفر یا هر چند بار تکرار شود. به عبارت دیگر، نقطه ستاره (*.) در کنار یکدیگر، یعنی هر تعداد علامت و کاراکتر که پشت هم بیایند. سپس با استفاده از $.\، به سیستم گفتیم که فقط خطوطی را به ما نشان دهد که کاراکتر نقطه در پایان آن قرار دارد، و همانطور که می‌بینید جهت این که به سیستم بگوییم که با نقطه به عنوان یک علامت خاص برخورد نکند، از \ استفاده کردیم.

Extended Regular Expressions

با استفاده از آپشن E- در grep، می‌توانیم از یک زبان گسترده‌تر regex به نام extended regex استفاده کنیم. extended regex علاوه بر قابلیت regex معمولی، یک سری قابلیت پرقدرت و جدیدتر دارد که به ما کمک می‌کند تا بتوانیم الگوهای پیچیده‌تری را جستجو کنیم.

گروه‌بندی

یکی از ساده‌ترین و کاربردی‌ترین قابلیت‌های extended regex، قابلیت گروه‌بندی الگوهای regex می‌باشد. کاربرد گروه‌بندی در این است که به ما امکان می‌دهد تا روی یک الگوی خاص، کنترل بیشتری داشته باشیم و بتوانیم آنها را تکرار کنیم یا به الگوهای موجود در یک گروه، رفرنس بزنیم.

برای گروه‌بندی الگو، کافی است آن را درون پرانتز قرار دهیم. یعنی:

```
[root@localhost ~]# grep -E "(الگوی مورد نظر)" whyamidoingthis
```

نکته: به جای استفاده از آپشن E-، می‌توانیم از egrep استفاده کنیم. egrep بر خلاف grep، به صورت پیش‌فرض با extended regex کار می‌کند. به عبارت دیگر، egrep و grep -E عملکرد یکسانی دارند. توجه کنید که آپشن E- را باید حتما با حرف بزرگ بزنید.

نکته: در regex معمولی نیز می‌توانیم الگوها را با استفاده از پرانتز گروه‌بندی کنیم، ولی برای این کار، باید قبل از پرانتزها از \ استفاده کنیم؛ یا به عبارت دیگر، باید پرانتزها را escape کنیم. یعنی دستور زیر با دستور بالا دقیقا یک کار را انجام می‌دهند:

```
[root@localhost ~]# grep -E "\(الگوی مورد نظر\)" whyamidoingthis
```

تناوب یا alternation

اگر به خاطر داشته باشید، ما از کروشه برای مشخص کردن حالت‌های متفاوت برای یک تک‌کاراکتر استفاده می‌کردیم. با استفاده از تناوب یا altercation، می‌توانیم حالت‌های متفاوت برای یک رشته یا یک الگو را تعریف کنیم.

برای مشخص کردن تناوب یا alternation، از علامت پایپ (|) استفاده می‌کنیم. از این علامت معمولا درون گروه‌بندی پرانتزی استفاده می‌کنند تا مشخص کنند یکی از دو یا چند رشته یا الگوی موجود در پرانتز، باید در خروجی به ما نشان داده شود.

مثلا فرض کنید می‌خواهیم به سیستم بگوییم که خطوطی را به ما بازگرداند که در آن یا رشته‌ی GPL یا رشته‌ی General Public License وجود داشته باشد:

```
[root@localhost gmp-6.0.0]# grep -E "(GPL|General Public License)" COPYING
  The GNU General Public License is a free, copyleft license for
the GNU General Public License is intended to guarantee your freedom to
GNU General Public License for most of our software; it applies also to
price.  Our General Public Licenses are designed to make sure that you
  Developers that use the GNU GPL protect your rights with two steps:
  For the developers' and authors' protection, the GPL clearly explains
authors' sake, the GPL requires that modified versions be marked as
have designed this version of the GPL to prohibit the practice for those
of the GPL, as needed to protect the freedom of users.
make it effectively proprietary.  To prevent this, the GPL assures that
...
```

همانطور که می‌بینید، با استفاده از آپشن E- در grep، یک گروه پرانتزی ایجاد کردیم و درون آن با استفاده از علامت |، به سیستم گفتیم که خطوطی که در آن رشته‌ی GPL یا رشته‌ی General Public License وجود دارد را به ما بازگرداند.

نکته: علامت | می‌تواند بین بیش از دو الگو یا رشته نیز قرار گیرد.

علامت ?

اگر به خاطر داشته باشید، گفتیم که علامت *، به دنبال رشته‌هایی می‌رود که در آن کاراکتر یا الگوی قبل از علامت *، صفر یا هر چند بار تکرار شده باشد. علامت ? که قبلا نیز با آن کار کردیم، عملکردی شبیه علامت * دارد؛ با این فرق که به دنبال رشته‌هایی می‌گردد که کاراکتر قبلی در آن **صفر** یا **یک بار** تکرار شده باشد.

این یعنی دستور grep -E “colou?r”، هم رشته‌ی colour، و هم رشته‌ی color را به ما باز می‌گرداند؛ اما colouur را به ما باز نمی‌گرداند.

حال بیایید یک مثال پیچیده‌تر بزنیم. فرض کنید می‌خواهیم به سیستم بگوییم که خطوطی که در آن رشته‌ی right یا رشته‌ی copyright وجود دارد را به ما نشان دهد:

```
[root@localhost gmp-6.0.0]# grep -E "(copy)?right" COPYING
```

```
 Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/>
  To protect your rights, we need to prevent others from denying you
these rights or asking you to surrender the rights.  Therefore, you have
know their rights.
  Developers that use the GNU GPL protect your rights with two steps:
(1) assert copyright on the software, and (2) offer you this License
  "Copyright" also means copyright-like laws that apply to other kinds of
  "The Program" refers to any copyrightable work licensed under this
...
```

همانطور که می‌بینید، به علت وجود علامت سوال پس از علامت پرانتز، به سیستم می‌گوییم که به دنبال رشته‌هایی برود که یا رشته‌ی copy در آن فقط **یک بار** وجود داشته باشد، یا **اصلا** رشته‌ی copy در آن وجود **نداشته باشد**. اما سیستم حتما باید کاراکترهای right را به ما نشان دهد.

اگر در دستور بالا، رشته‌ی کپی را درون پرانتز قرار نمی‌دادیم، چه اتفاقی می‌افتاد؟

> اگر از پرانتز استفاده نمی‌کردیم، علامت ?، به دنبال رشته‌هایی می‌رفت که در آن کاراکتر y صفر یا یک‌بار تکرار شده باشد.

علامت +

علامت +، به دنبال رشته‌هایی می‌رود که الگو یا کاراکتر قبل در آن **یک** یا **هر چند بار** تکرار شده باشد. این عملکرد تقریبا شبیه عملکرد علامت * می‌باشد؛ اما هنگام استفاده از +، الگو یا کاراکتر قبل **حتما باید یک‌بار** تکرار شده باشد.

> این یعنی دستور `grep -E “ab+c”`، رشته‌های abc، abbc، abbbc و... را به ما باز می‌گرداند، اما بر خلاف *، رشته‌ی ac را به ما باز نمی‌گرداند.

فرض کنید می‌خواهیم به سیستم بگوییم که خط‌هایی را به ما بازگرداند که رشته‌ی free، به اضافه‌ی یک یا هر چند کاراکتر، به جز کاراکتر فاصله‌ی خالی (space)، در آن وجود داشته باشد:

```
[root@localhost gmp-6.0.0]# grep -E "free[^[:space:]]+" COPYING
  The GNU General Public License is a free, copyleft license for
to take away your freedom to share and change the works.  By contrast,
the GNU General Public License is intended to guarantee your freedom to
  When we speak of free software, we are referring to freedom, not
have the freedom to distribute copies of free software (and charge for
you modify it: responsibilities to respect the freedom of others.
freedoms that you received.  You must make sure that they, too, receive
protecting users' freedom to change the software.  The systematic
of the GPL, as needed to protect the freedom of users.
patents cannot be used to render the program non-free.
```

همانطور که می‌بینید، ما ابتدا به سیستم گفتیم که دنبال خط‌هایی بگردد که در آن رشته‌ی free وجود دارد. سپس چون می‌خواستیم بعد از free، هر کاراکتری به جز فاصله خالی داشته باشیم، یک کروشه قرار دادیم و پس از کروشه، علامت + را قرار دادیم. این یعنی الگوی داخل کروشه حتما **باید یک بار**، یا **هر چند بار** تکرار شود. اگر یادتان باشد، کروشه نشان دهنده‌ی امکان وجود یکی از کاراکترهای موجود درون خود، بود. اما ما گفتیم که هیچ گونه فاصله خالی پس از رشته‌ی free نمی‌خواهیم، پس از علامت ^ درون کروشه استفاده می‌کنیم. این مشخص می‌کند که در این موقعیت هر کاراکتری به جز فاصله خالی می‌تواند وجود داشته باشد. پس بدین صورت، با استفاده از کروشه، کاراکتری که نمیخواهیم پس از رشته‌ی free قرار گیرد را مشخص

کردیم. ما برای مشخص کردن این کاراکتر، از کلاس کاراکتری [:space:] استفاده کردیم، اما می‌توانستیم از فاصله‌ی خالی هم استفاده کنیم.

پس بدین شکل، سیستم کلیه خطوطی که در آن رشته‌ی free، به اضافه‌ی یک یا هر چند کاراکتر دیگر که فاصله‌ی خالی نیستند، وجود داشت را به ما باز گرداند. همانطور که می‌بینید، در خط اول، رشته‌ی free, به ما نشان داده شده است، در واقع کاراکتر , نیز در خروجی آمده، اما در خط چهارم، free به ما نشان داده نشده است، چون پس از رشته‌ی free، یک فاصله‌ی خالی وجود داشته است.

مشخص‌کردن تعداد تکرار یک الگو

اگر بخواهم تعداد دفعات تکرار یک الگو را مشخص کنیم، از علامت آکولاد ({}) استفاده می‌کنیم. تعداد تکرار می‌تواند یک عدد، یا محدوده‌ای از اعداد باشد.

برای مثال، `grep -E "abc{3}"` رشته‌ی abccc را به ما باز می‌گرداند. اگر یک محدوده تعریف کنیم، مثلا `grep -E "abc{2,5}"` رشته‌های abcc، abccc، abcccc و abccccc را به ما باز می‌گرداند.

حال بیایید یک مثال پیچیده‌تر بزنیم. فرض کنید می‌خواهیم به سیستم بگوییم که تنها خطوطی را به ما نشان دهد که در آن حروف صدا دار (a، e، o، u، i) سه بار پشت سر هم آمده باشند. برای این کار:

```
[root@localhost gmp-6.0.0]# grep -Ei "[aeoui]{3}" COPYING
changed, so that their problems will not be attributed erroneously to
authors of previous versions.
receive it, in any medium, provided that you conspicuously and
give under the previous paragraph, plus a right to possession of the
covered work so as to satisfy simultaneously your obligations under this
```

همانطور که می‌بینید، ما ابتدا حروف صدا دار را درون یک کروشه قرار دادیم. همانطور که می‌دانید، قرار دادن کاراکترها درون کروشه، به معنی این است در این موقعیت، فقط یکی از کاراکترهای درون کروشه می‌تواند وجود داشته باشد. سپس با استفاده از علامت آکولاد و قرار دادن عدد ۳ درون آکولاد، مشخص کردیم که می‌خواهیم الگوی درون کروشه، سه بار تکرار شود. یعنی به عبارت دیگر، داریم به سیستم می‌گوییم که دنبال هر رشته‌ای بگردد که ۳ تا از کاراکترهای آن به صورت پشت سر هم، یکی از حروف a، e، o، u یا i، باشد. همانطور که می‌بینید، در خروجی کلیه عبارت‌هایی که در آن یکی از ترکیب‌های کاراکترهای aeoui سه بار پشت سر هم تکرار شده بود به ما نمایش داده شد. اگر همچنان درک این قضیه برایتان دشوار است، خودتان را بُکُشید.

> همانطور که می‌بینید ما در دستور بالا از آپشن `i-` استفاده کردیم. آیا به خاطر دارید که این آپشن در `grep` چه عملی انجام می‌دهد؟

حال بیایید مثال دیگری بزنیم. فرض کنید می‌خواهیم به سیستم بگوییم که دنبال رشته‌هایی بگردد که بین ۱۶ تا ۲۰ حرف (حروف الفبا) دارند. برای این کار:

```
[root@localhost gmp-6.0.0]# grep -E  "[[:alpha:]]{16,20}" COPYING
certain responsibilities if you distribute copies of the software, or if
you modify it: responsibilities to respect the freedom of others.
    c) Prohibiting misrepresentation of the origin of that material, or
```

همانطور که می‌بینید، سیستم هر رشته‌ای که از ۱۶ الی ۲۰ کاراکتر موجود در حروف الفبا تشکیل شده بود را به ما نشان داد. الگوی به کار رفته نیاز به توضیح خاصی ندارد، چون دقیقا مانند الگوی مثال قبل کار می‌کند.

همانطور که گفتیم، در علامت آکولاد، می‌توانیم یک عدد یا محدوده‌ای از اعداد قرار دهیم؛ که اینجا ما محدوده را بین ۱۶ تا ۲۰ قرار دادیم. همچنین درون کروشه، به جای استفاده از A-Za-z، از کلاس کاراکتری [:alpha:] استفاده کردیم که یکی از POSIX Character Classes می‌باشد.