Linux Professional Institute

LPIC-1

جلسه سوم: آشنایی نصب نرم‌افزار در لینوکس

در این جلسه:

ویدئو اول:

- آشنایی با sed
- آشنایی ابتدایی با تنظیمات کارت شبکه
- صحبت در مورد نصب نرم‌افزار در لینوکس
- آشنایی با مفهوم پکیج‌ها و پکیج منیجرها
- آشنایی با طریقه‌ی انتقال فایل با SFTP
- آشنایی با دستور rpm
- آشنایی با دستور yum

ویدئو دوم:

- دانلود فایل با استفاده از wget
- نصب برنامه از روی سورس کد
- صحبت در مورد مشکلات نصب از روی سورس کد
- صحبت کلی در مورد پکیج منیجر سیستم‌های Debian-based
- صحبت در مورد طریقه‌ی بررسی یکپارچگی فایل‌ها

فهرست مطالب

مقدمه

جلسه‌ی قبل با ابزارهای تغییر و مشاهده‌ی فایل‌های متنی کار کردیم، با ابزار grep آشنا شدیم و به صورت خیلی ابتدایی از regexها استفاده کردیم. در این جلسه، ابتدا در مورد ابزار sed صحبت می‌کنیم، سپس در مورد چگونگی تنظیم کارت شبکه برای اتصال به اینترنت صحبت می‌کنیم و چگونگی انتقال فایل به لینوکس با SFTP را یاد می‌گیریم و در نهایت به سراغ روش‌های متفاوت نصب نرم‌افزار در لینوکس می‌رویم.

ابزار sed

یکی از ابزارهای بسیار جالب برای تغییر فایل‌های متنی، ابزار sed می‌باشد. sed که مخفف stream editor می‌باشد، می‌تواند خروجی یک دستور یا یک فایل متنی را تغییر دهد یا edit کند. sed با توجه به آپشن‌ها و الگوهایی که برایش مشخص می‌کنیم، متن‌ها را تغییر می‌دهد. در کل sed فرآیندی نظیر زیر دارد:

۱) هر بار، یک خط از متن موجود در ورودی را می‌خواند.
۲) متن آن خط را با آپشن‌ها و الگوهای مشخص شده تطبیق می‌دهد.
۳) متن را طبق الگو و آپشن مشخص شده تغییر می‌دهد.
٤) متن تغییر داده شده را STDOUT نمایش می‌دهد.

پس از آن که sed تغییرات را در یک خط اعمال کرد، به سراغ خط بعدی می‌رود و مراحل قبل را تکرار می‌کند تا به پایان متن برسد. برای استفاده از sed، به صورت زیر عمل می‌کنیم:

```
[root@localhost ~]# sed [آپشن] [الگو] [نام فایل]
```

توجه کنید که اگر به sed نام فایل خاصی را ندهیم، به صورت پیش فرض، تغییرات را روی متن موجود در STDIN خود اعمال می‌کند.

جایگزین کردن یک رشته یا الگو

ما می‌توانیم با استفاده از sed، یک رشته یا الگو را با رشته‌ای دیگر جایگزین کنیم. به مثال زیر توجه کنید:

```
[root@localhost ~]# echo "My name is Behnam." | sed 's/Behnam/Earl/'
My name is Earl.
```

همانطور که می‌بینید، ما خروجی دستور echo را به ورودی دستور sed دادیم. کل نوشته‌ی موجود بین دو علامت ' ، الگوی ما محسوب می‌شود. ما ابتدا با نوشتن s، به sed می‌گوییم که به دنبال جایگزین کردن (replace) هستیم. سپس با قرار دادن یک علامت /، الگویی که دنبال جایگزینی آن هستیم را می‌نویسیم. پس از آن، یک / دیگر قرار داده و چیزی که می‌خواهیم جایگزین نتایج الگوی ذکر شده شود را می‌نویسیم و در نهایت کار را با یک / دیگر تمام می‌کنیم. ما در دستور بالا، به دنبال جایگزین کردن رشته‌ی Behnam با Earl بودیم و این کار را با sed انجام دادیم.

نکته: کلیه‌ی الگوهایی که به sed می‌دهیم باید بین دو علامت ' قرار گیرد.

همانطور که گفتیم sed می‌تواند محتویات یک فایل متنی را نیز تغییر دهد. برای مثال:

```
[root@localhost ~]# cat albatross
My name is Albatross. The Albatross.
```

```
[root@localhost ~]# sed 's/Albatross/Behnam/' albatross
My name is Behnam. The Albatross.
```

همانطور که می‌بینید با قرار دادن نام فایل پس از الگوی مورد نظر، sed اقدام به جایگزین کردن رشته‌ی Albatross با Behnam کرد.

اما اینجا مشکلی وجود دارد؛ چرا sed فقط اولین Albatross را تغییر داد؟

> دلیل این است که به صورت پیش‌فرض، sed فقط اولین مورد وقوع از یک الگو در یک خط را تغییر می‌دهد.

برای این که کاری کنیم که sed کلیه‌ی موارد وقوع از یک رشته در یک خط را تغییر دهد، باید دستور را به صورت زیر وارد کنیم:

```
[root@localhost ~]# sed 's/Albatross/Behnam/g' albatross
My name is Behnam. The Behnam.
```

همانطور که می‌بینید با قرار دادن g در پایان الگو (پس از آخرین /)، به sed گفتیم که کلیه‌ی موارد وقوع از رشته‌ی مشخص شده در یک خط را تغییر دهد.

همانطور که گفتیم، sed به صورت پیش‌فرض تغییراتی که روی یک متن اعمال می‌کند را روی STDOUT نشان می‌دهد. یا به عبارتی دیگر، محتویات موجود در فایل اصلی را تغییر نمی‌دهد. یعنی:

```
[root@localhost ~]# cat name
My name is Behnam.
My name is also The Albatross.
What's your name?
I don't care for your name...
[root@localhost ~]# sed 's/name/face/' name
My face is Behnam.
My face is also The Albatross.
What's your face?
I don't care for your face...
[root@localhost ~]# cat name
My name is Behnam.
My name is also The Albatross.
What's your name?
I don't care for your name...
```

همانطور که می‌بینید، فایل اصلی دچار تغییر نشده است و خروجی sed فقط روی STDOUT نمایش داده شده است. اگر بخواهیم خروجی sed درون یک فایل ریخته شود، باید از redirectorها استفاده کنیم.

پاک کردن خط دارای الگو یا رشته‌ی خاص

تا به اینجا فقط از sed برای جایگزینی استفاده کرده‌ایم. sed، می‌تواند خطی که در آن الگوی خاصی موجود باشد را نیز پاک کند. مثلا:

```
[root@localhost ~]# cat name
My name is Behnam.
My name is also The Albatross.
What's your name?
I don't care for your name...
[root@localhost ~]# sed '/My name/d' name
What's your name?
I don't care for your name...
```

همانطور که می‌بینید، ابتدا الگوی خود را بین دو علامت ' قرار دادیم و سپس با قرار دادن یک علامت /، رشته‌ای که میخواهیم دنبال آن بگردیم را مشخص کردیم و سپس یک علامت / دیگر قرار دادیم و پشت سر آن، از d استفاده کردیم. استفاده از d، به معنای پاک کردن خطی می‌باشد که رشته‌ی مشخص شده در آن وجود داشته است.

تغییر محتوای یک خط

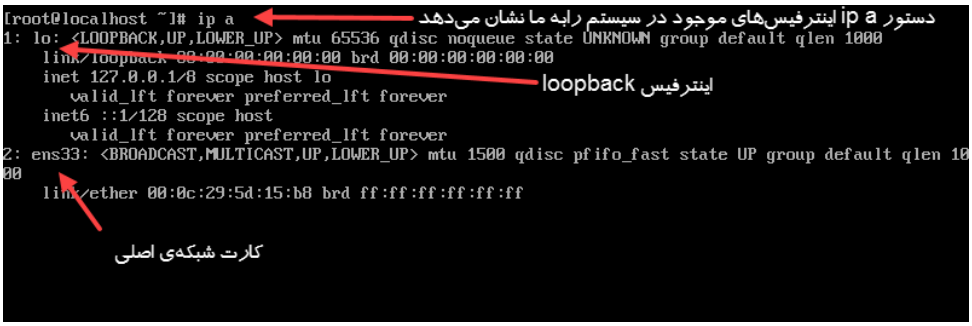
ما میتوانیم از sed بخواهیم که محتویات یک خط را به طور کامل تغییر دهد. مثلا فرض کنید می‌خواهیم خط چهارم موجود در فایل name را به صورت کامل تغییر دهیم. برای این کار:

```
[root@localhost ~]# sed '4cI changed this line.' name
My name is Behnam.
My name is also The Albatross.
What's your name?
I changed this line.
```

همانطور که می‌بینید، با قراردادن شماره‌ی خط مورد نظر (4) قبل از c و نوشتن متن جدید پس از c، به sed می‌گوییم که محتویات کدام خط را به چه چیزی تغییر دهد.

تنظیم کارت شبکه در لینوکس

ما تا به اینجا برای کار با لینوکس، نیازی به اتصال به شبکه نداشتیم. اما اتصال به یک شبکه و همچنین اینترنت، بسیار کار ما را راحت می‌کند و درب‌های زیادی را به روی ما باز می‌کند. بیایید ابتدا اطلاعاتی در مورد کارت‌های شبکه‌ی موجود در سیستم به دست آوریم. برای این کار، از دستور ip a استفاده می‌کنیم:



تصویر ۱- مشاهده‌ی اینترفیس‌های شبکه‌ی موجود در سیستم با دستور ip a

همانطور که در تصویر ۱ می‌بینید، در حال حاضر دو اینترفیس (کارت شبکه) در سیستم وجود دارد. یکی از آنها اینترفیس loopback می‌باشد که به آن کاری نداریم. اینترفیس بعدی (ens33)، اینترفیسی است که می‌توانیم از طریق آن به اینترنت برویم یا به سیستم SSH بزنیم. البته ممکن است نام این اینترفیس در سیستم شما متفاوت باشد.

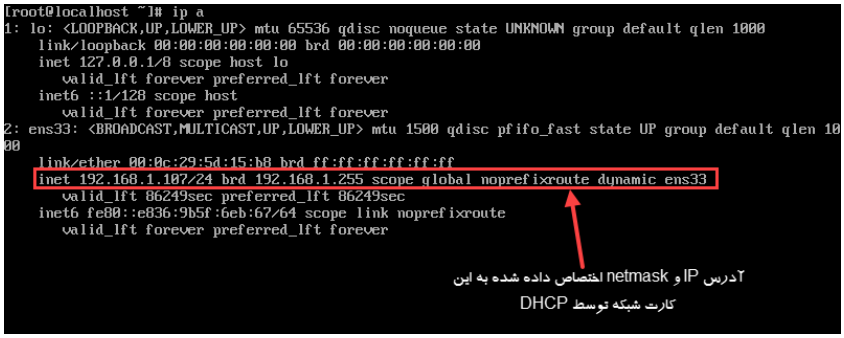
ما قصد نداریم در این بخش در مورد مفاهیم شبکه صحبت کنیم، اما برای این که به یک شبکه متصل شویم و به اینترنت برویم، باید به اینترفیس شبکه، یک آدرس IP و یک سری اطلاعات دیگر، اختصاص دهیم. ما می‌توانیم این اطلاعات را یا به صورت اتوماتیک (از طریق DHCP)، یا به صورت دستی (استاتیک) به اینترفیس اختصاص دهیم.

تنظیم اطلاعات کارت شبکه از طریق DHCP

DHCP پروتکلی است که به صورت اتوماتیک به هر کارت شبکه‌ای که در خواست کند، IP و سایر اطلاعات مربوط به یک شبکه را می‌دهد. برای این که به سیستم از این طریق IP بدهیم، باید ابتدا اینترفیس کارت شبکه را روشن کنیم:

```
[root@localhost ~]# ifup ens33
Connection successfully activated (D-Bus active path:
/org/freedesktop/NetworkManager/ActiveConnection/1)
```

ما با وارد کردن دستور `ifup` و نوشتن نام اینترفیس مورد نظر جلوی آن (نام اینترفیس در سیستم ما، ens33 بود، در سیستم شما ممکن است اینترفیس نام دیگری داشته باشد)، اینترفیس کارت شبکه را روشن کردیم. با روشن کردن کارت شبکه، بیایید بار دیگر دستور `ip a` را اجرا کنیم و وضعیت کارت شبکه را چک کنیم:



تصویر ۲- مشاهده‌ی وضعیت اینترفیس پس از روشن کردن آن

همانطور که می‌بینید با روشن کردن کارت شبکه، اینترفیس ما به صورت اتوماتیک صاحب یک IP شده است. این کار DHCP است. به محض روشن کردن کارت شبکه، این اینترفیس یک درخواست DHCP به سرور DHCP موجود در شبکه فرستاده و در جواب آن، یک آدرس IP، Subnet Mask، Default Gateway و اطلاعات DNS دریافت کرده است. دلیل این که اینترفیس به محض روشن شدن یک درخواست DHCP فرستاد را بعدا با هم بررسی می‌کنیم.

نکته: برای این که سیستم از DHCP اطلاعات شبکه را دریافت کند، باید حتما در شبکه‌ی شما یک DHCP Server وجود داشته باشد. مثلا مودم ADSL شما در منزل، درون خود یک DHCP Server دارد و شما از این طریق می‌توانید از DHCP آدرس بگیرید. اما در محیط کاری، باید حتما یک DHCP Server داشته باشید تا بتوانید از آن IP بگیرید.

اطلاعاتی که از طریق DHCP روی کارت شبکه تنظیم می‌شود، پس از هر بار درخواست (مثلا خاموش و روشن کردن اینترفیس یا سیستم) دچار تغییر شود. این امر هم می‌تواند یک مزیت باشد و هم یک عیب؛ مزیت آن در این است که به محض ایجاد تغییر در یکی از اطلاعات شبکه مثل آدرس Default Gateway (که برای اتصال به اینترنت به آن نیاز داریم)، DHCP می‌تواند این تغییر را برای همه‌ی سیستم‌های موجود در شبکه بفرستد. اما عیب آن در این است که IP یک سیستم خاص، می‌تواند دائما دچار تغییر شود، که این امر برای سرورها، اصلا مناسب نیست.

نکته‌ی دیگر که باید به آن توجه کنید این است که اگر ما سیستم خود را خاموش و روشن کنیم، اینترفیس شبکه‌ی ما نیز خاموش خواهد شد و باید وقتی وارد سیستم شدیم، بار دیگر با دستور `ifup` آن را روشن کنیم.

این امر ارتباطی به استفاده از DHCP ندارد و در بخش بعدی در مورد دلیل این امر و چگونگی حل آن صحبت خواهیم کرد.

تنظیم اطلاعات کارت شبکه به صورت دستی (Static)

برای این که به سیستم به صورت استاتیک IP بدهیم، باید مقداری تنظیمات دستی (😐) انجام دهیم. IP استاتیک ثابت است و دچار تغییر نمی‌شود؛ اما بر خلاف DHCP، هنگام تنظیم IP استاتیک باید در مورد شبکه اطلاعاتی داشته باشیم.

لینوکس اطلاعات مربوط به هر کارت شبکه را روی یک فایل متنی ذخیره می‌کند. جلسه قبل با ادیتور vi آشنا شدیم. بیایید فایل تنظیمات کارت شبکه را با این ادیتور باز کنیم و در مورد آن بحث کنیم. تنظیمات کارت شبکه در مسیر `/etc/sysconfig/network-scripts` قرار دارد. شما در این مسیر باید به دنبال فایلی باشید که با `ifcfg-` شروع می‌شود و پس از آن، نام اینترفیس کارت شبکه‌ی شما نوشته شده است. بیایید این فایل را با هم باز کنیم و آن را بررسی کنیم:

```
[root@localhost ~]# vi /etc/sysconfig/network-scripts/ifcfg-ens33
TYPE=Ethernet
PROXY_METHOD=none
BROWSER_ONLY=no
BOOTPROTO=dhcp
DEFROUTE=yes
IPV4_FAILURE_FATAL=no
IPV6INIT=yes
IPV6_AUTOCONF=yes
IPV6_DEFROUTE=yes
IPV6_FAILURE_FATAL=no
IPV6_ADDR_GEN_MODE=stable-privacy
NAME=ens33
UUID=56002cf0-9e73-4617-8e54-3556bdbeb508
DEVICE=ens33
ONBOOT=no
```

در حال حاضر به اکثر اطلاعات نوشته شده در این فایل کاری نداریم. اما دو خط در این فایل برای ما مهم هستند. در خط چهارم، عبارت BOOTPROTO را می‌بینید. این خط، مشخص می‌کند که اینترفیس هنگام روشن شدن، باید از چه طریق IP بگیرد. در حال حاضر جلوی این عبارت، dhcp نوشته شده است، که بدین معناست که اینترفیس هنگام روشن شدن از DHCP Server آی‌پی می‌گیرد. اگر بخواهیم این اینترفیس به صورت دستی از ما IP قبول کند، باید مقدار BOOTPROTO را برابر با static قرار دهیم.

سپس در خط آخر، عبارت ONBOOT را می‌بینید. این عبارت مشخص می‌کند که این اینترفیس هنگام روشن شدن سیستم باید روشن شود یا نه. دلیل این که در بخش قبل برای روشن کردن کارت شبکه مجبور به استفاده از ifup بودیم، وجود عبارات no جلوی این عبارت بود. اگر این عبارت را برابر با yes قرار دهیم، دیگر هنگام روشن کردن سیستم، نیازی به روشن کردن اینترفیس با دستور ifup نخواهیم داشت.

حال بیایید در مورد تنظیم IP و سایر اطلاعات به صورت دستی صحبت کنیم. برای تنظیم این اطلاعات، کافی است عبارات زیر را درون این فایل اضافه کنیم:

```
IPADDR=192.168.1.50
NETMASK=255.255.255.0
GATEWAY=192.168.1.1
```

```
DNS1=8.8.8.8
DNS2=4.2.2.4
```

مشخص است هر کدام از این عبارات چه کاری انجام می‌دهند. IPADDR نشان دهنده‌ی آدرس IP مورد نظر شما می‌باشد. NETMASK نشان دهنده‌ی Subnet Mask آدرس IP می‌باشد. GATEWAY نشان‌دهنده‌ی Default Gateway شبکه‌ی شما می‌باشد و DNS1 و DNS2 نشان‌دهنده‌ی سرورهای DNS مورد نظر شما می‌باشند. فراموش نکنید که پس از وارد کردن این مقادیر، باید فایل را ذخیره کنید.

پس الان فایل ifcfg-ens33 باید شبیه زیر باشد:

```
[root@localhost ~]# cat /etc/sysconfig/network-scripts/ifcfg-ens33
TYPE=Ethernet
PROXY_METHOD=none
BROWSER_ONLY=no
BOOTPROTO=static
DEFROUTE=yes
IPV4_FAILURE_FATAL=no
IPV6INIT=yes
IPV6_AUTOCONF=yes
IPV6_DEFROUTE=yes
IPV6_FAILURE_FATAL=no
IPV6_ADDR_GEN_MODE=stable-privacy
NAME=ens33
UUID=56002cf0-9e73-4617-8e54-3556bdbeb508
DEVICE=ens33
ONBOOT=yes
IPADDR=192.168.1.50
NETMASK=255.255.255.0
GATEWAY=192.168.1.1
DNS1=8.8.8.8
DNS2=4.2.2.4
```

خطوط قرمز، نشان دهنده‌ی خطوطی می‌باشند که تغییر دادیم و خطوط آبی نشان‌دهنده‌ی خطوطی هستند که به فایل اضافه کردیم.

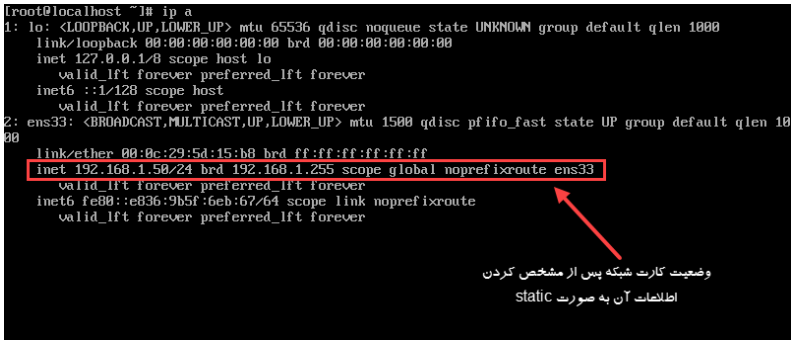
حال برای این که تنظیمات ما اعمال شوند، باید یک بار اینترفیس را خاموش و سپس آن را روشن کنیم:

```
[root@localhost ~]# ifdown ens33 && ifup ens33
```

دستور ifdown، اینترفیس مورد نظر را خاموش کرده و دستور ifup، اینترفیس مورد نظر را روشن می‌کند. همانطور که می‌بینید ما بین این دو دستور علامت && را قرار دادیم. این به سیستم می‌گوید که ابتدا دستور ifdown را اجرا کن و پس از اجرای موفق آن، دستور ifup را اجرا کن. یعنی با این کار، توانستیم در یک خط اینترفیس را خاموش و سپس روشن کنیم. البته می‌توانستیم این دستورها را به صورت جداگانه نیز اجرا کنیم.

نکته: استفاده از علامت && بین دو دستور ifdown و ifup، زمانی که با SSH به سیستم متصل شده باشیم و قصد ایجاد تغییر در تنظیمات کارت شبکه را داشته باشیم بسیار کاربردی خواهد بود؛ چون اگر با SSH به سیستم متصل باشیم و بخواهیم پس از اعمال تغییرات روی اینترفیس شبکه، آن را ابتدا خاموش و سپس روشن کنیم، به محض وارد کردن دستور ifdown، ارتباط ما با سیستم قطع می‌شود و دیگر فرصتی برای اجرای دستور ifup نخواهیم داشت.

حال بیایید وضعیت اینترفیس‌ها را بررسی کنیم:



تصویر ۳ - وضعیت کارت شبکه پس از از خاموش و روشن کردن اینترفیس

همانطور که می‌بینید، اینترفیس ما با IP که به صورت static مشخص کرده بودیم روشن شد. حال ما می‌توانیم با استفاده از این IP، به سیستم لینوکس خود SSH بزنیم و همچنین به دلیل تنظیم DNS و Default Gateway می‌توانیم به اینترنت متصل شویم.

نصب نرم‌افزار روی لینوکس

پرواضح است که یک سیستم لینوکسی بدون نرم‌افزارهای جانبی، به درد نمی‌خورد. برای نصب نرم‌افزار روی لینوکس چندین روش متفاوت وجود دارد. یکی از این روش‌ها، کاملپایل کردن نرم‌افزار از source code آن برنامه می‌باشد. اگر برنامه‌نویسی کرده باشید، می‌دانید که یک برنامه ممکن است از چندین library استفاده کند، یا برای اجرا، نیاز به یک نرم‌افزار دیگر داشته باشد. اگر بخواهیم برنامه‌ای را از روی سورس کد نصب کنیم، باید تک تک این لایبرری‌ها و نرم‌افزارها، یا به عبارتی دیگر، این dependencyها را نیز در سیستم نصب کنیم؛ که خود آنها نیز ممکن است به یک سری برنامه‌ی و لایبرری دیگر dependency داشته باشند.

اکثر کاربران لینوکس می‌خواهند یک نرم‌افزار را به راحتی دانلود و نصب کنند و خود را درگیر مسائلی مثل کامپایل کردن، نصب کردن dependencyها و... نکنند. برای حل این مشکل، سیستم‌های لینوکسی از برنامه‌ای به نام Package Manager استفاده می‌کنند. پکیج منیجرها، برنامه‌ها و لایبرری‌های متفاوت را به صورت کامپایل شده در کنار هم قرار می‌دهند و مدیریت و نصب برنامه‌ها را ساده‌تر می‌کنند.

توزیع‌های متفاوت لینوکس، پکیج منیجرهای متفاوتی برای خود ایجاد کرده‌اند. معروف‌ترین این سیستم‌ها که اکنون تبدیل به یک استاندارد شده اند، به شرح زیر می‌باشند:

- **Red Hat Package Management (RPM)**: اکثر توزیع‌های Red Hat-based نظیر CentOS، Fedora و... از این پکیج منیجر استفاده می‌کنند.
- **Debian Package Management (Apt)**: اکثر توزیع‌های Debian-based مثل Ubuntu، Mint و... از این پکیج منیجر استفاده می‌کنند.

این پکیج منیجرها از نظر مفاهیم اولیه، کاملا شبیه به هم می‌باشند، اما تفاوت‌های جزئی با هم دارند. ما ابتدا با مفاهیم مشترک بین این پکیج منیجرها آشنا می‌شویم و از آنجا که از توزیع CentOS استفاده می‌کنیم، تمرکز خود را بر روی پکیج منیجر RPM می‌گذاریم.

آشنایی با مفاهیم اولیه پکیج منیجرها

همانطور که گفتیم ابتدا می‌خواهیم با یک سری از مفاهیم که بین همه‌ی پکیج‌منیجرها مشترک است، آشنا شویم:

- **پکیج (Package)**

پکیج‌ها، کلکسیونی از فایل‌های متفاوت می‌باشند. می‌توانید به پکیج‌ها به عنوان یک فایل zip نگاه کنید که فایل‌ها و لایبرری‌های لازم برای نصب یک نرم‌افزار را درون خود دارند. هنگامی که یک پکیج را درون سیستم خود نصب کنید، چندین فایل متفاوت در سیستم قرار می‌گیرد و پکیج منیجر از موقعیت و عملکرد این فایل‌ها، مطلع خواهد شد.

- **دیتابیس فایل‌های نصب شده (Installed File Database)**

پکیج منیجرها یک دیتابیس از همه‌ی فایل‌های نصب شده نگه‌داری می‌کنند. این دیتابیس شامل اطلاعاتی در مورد همه‌ی فایل‌هایی که توسط این پکیج منیجر نصب شده می‌باشد.

- **Dependencies**

یکی از مهم‌ترین اطلاعاتی که یک پکیج منیجر درون خود نگه‌داری می‌کند، اطلاعات مربوط به dependencyهای هر پکیج می‌باشد. dependencyها نشان دهنده‌ی نیاز یک نرم‌افزار به لایبرری یا نرم‌افزاری دیگر برای اجرا می‌باشند. مثلا اگر پکیج A برای اجرا به پکیج B احتیاج داشته باشد، پکیج منیجر در دیتابیس خود این اطلاعات را خواهد داشت. اگر بخواهید پکیج A را بدون نصب پکیج B نصب کنید، پکیج منیجر شما را از نیاز پکیج A به پکیج B مطلع می‌سازد و اجازه نصب پکیج A را نمی‌دهد. همچنین اگر بخواهید پکیج B را حذف کنید، پکیج منیجر به شما اجازه نمی‌دهد، چون نرم‌افزار A برای اجرا به آن نیاز دارد.

- **Checksums**

پکیج منیجرها اطلاعات مربوط به تصدیق یکپارچگی یا Integrity یک پکیج را درون خود نگه‌داری می‌کنند که این امر به ما کمک می‌کند تا بتوانیم صحت فایل‌های نصب شده را تصدیق کنیم.

- **آپدیت و پاک کردن**

پکیج منیجرها به دلیل نگه‌داری دیتابیسی از dependencyها و فایل‌ها، کار پاک کردن برنامه‌ها را بسیار ساده می‌کنند و به دلیل نگه‌داری دیتابیسی از ورژن نرم‌افزارها، کار آپدیت کردن را نیز ساده می‌کنند.

پکیج منیجر RPM

سیستم‌عامل‌هایی مانند CentOS، Fedora و... که Red Hat-based هستند، از پکیج منیجر RPM استفاده می‌کنند. البته استفاده از این پکیج منیجر محدود به سیستم‌های Red Hat-based نیست، برخی از سیستم‌عامل‌های دیگر نظیر SUSE نیز از RPM استفاده می‌کنند.

فایل‌های RPM، پسوند rpm. دارند و نام آنها از الگوی زیر پیروی می‌کند:

```
PACKAGE-NAME-VERSION-RELEASE.ARCHITECTURE.rpm
```

PACKAGE-NAME: نام نرم‌افزار می‌باشد. توجه کنید که برخی از اوقات ممکن است اسم پکیج یک نرم‌افزار با نام خود نرم‌افزار تفاوت داشته باشد. (مثل apache web server که نام پکیج آن httpd web server است.)

VERSION: شماره‌ی نسخه‌ی کنونی نرم‌افزار را مشخص می‌کند. مثلا ۳ یا ۲,۱۰.

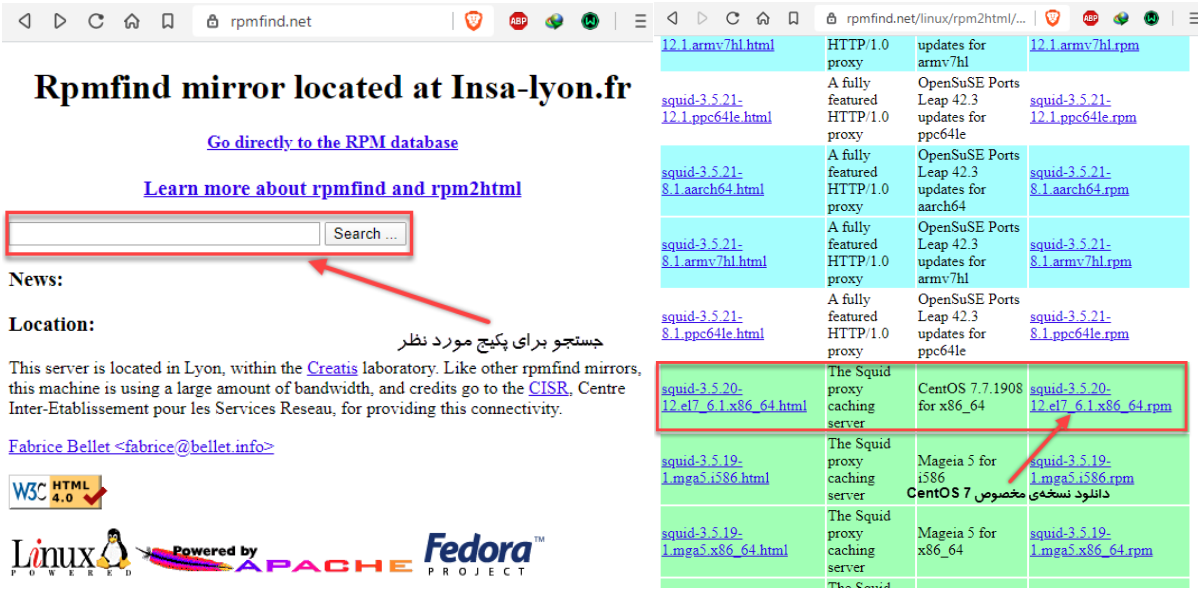
RELEASE: شماره‌ی build number را مشخص می‌کند. این شماره، مشخص‌کننده‌ی تغییراتی در نرم‌افزار است که به دلیل کوچک بودن، هنوز منجر به افزایش شماره‌ی ورژن نشده‌اند.

ARCHITECTURE: معماری CPU را مشخص می‌کند. مثلا سیستم‌های ۶۴ بیتی، معماری x86_64 دارند. برخی از پکیج‌ها، از کلمه‌ی noarch در این قسمت استفاده می‌کنند که نشان می‌دهد این پکیج، روی همه‌ی معماری‌ها اجرا می‌شود.

دانلود فایل‌های RPM

خوب، ما تا اینجا زیاد در مورد rpm صحبت کردیم. اما اگر بخواهیم یک فایل rpm را در سیستم نصب کنیم باید چه کنیم؟ اصلا فایل‌های rpm را از کجا بیاوریم؟

بیایید اول یک فایل rpm دانلود کنیم. وبسایت‌های زیادی برای دانلود فایل‌های rpm وجود دارد. یکی از این سایت‌ها، rpmfind.net می‌باشد. بیایید به این سایت برویم و پکیج squid را برای CentOS دانلود کنیم. برای این کار کافی است در وبسایت rpmfind، پکیج squid را جستجو کنیم و سپس نسخه‌ی مربوط به CentOS 7 را دانلود کنیم:



تصویر ٤- صفحه‌ی اصلی وبسایت rpmfind.net و دانلود squid برای CentOS 7

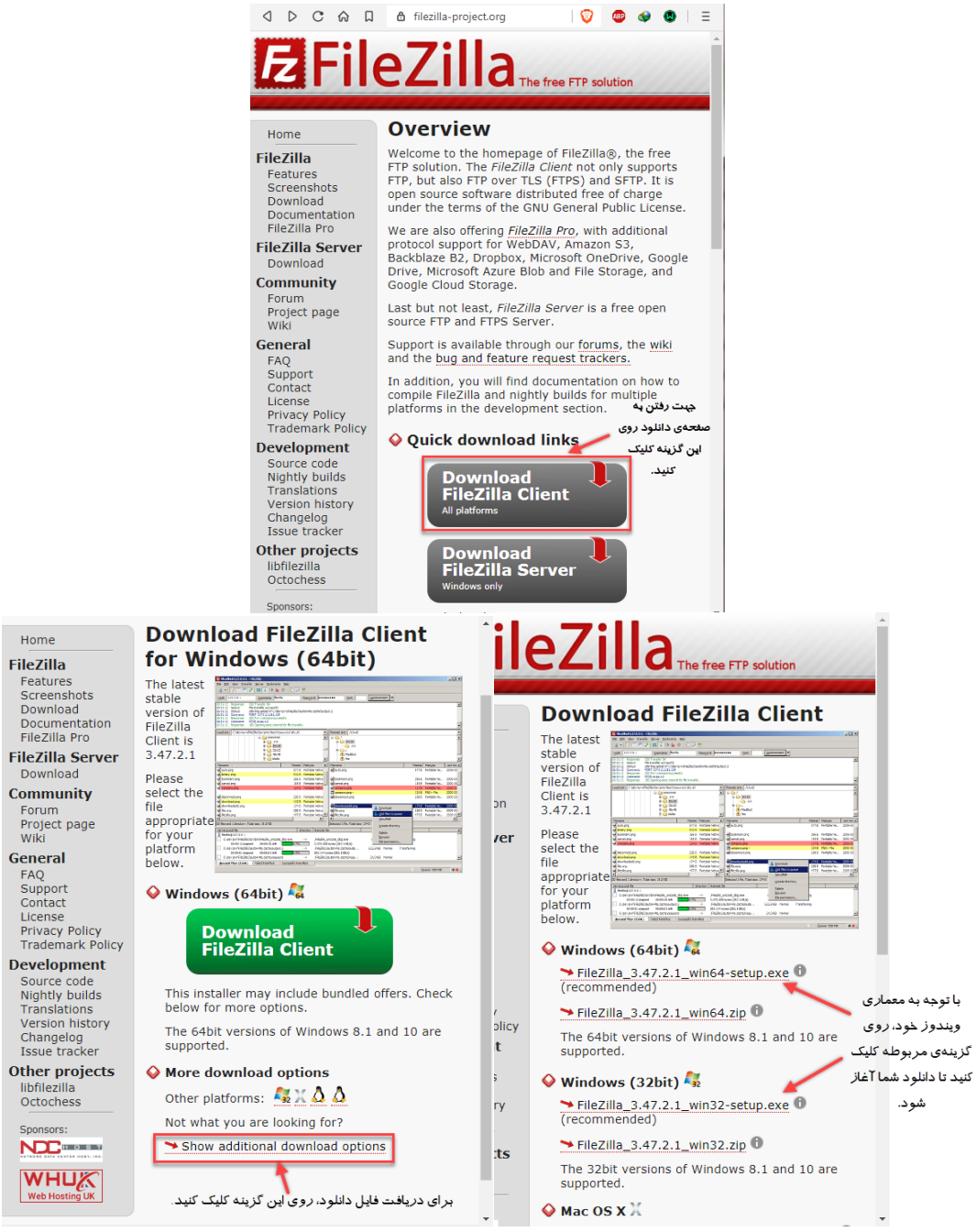
تمرین: فایل rpm برنامه‌ی zsh را برای CentOS 7 دانلود کنید.

حال یک مشکل بزرگ داریم؛ ما این فایل را دانلود کردیم، اما چگونه می‌توانیم آنرا داخل ماشین مجازی لینوکس خود بفرستیم؟

انتقال فایل به لینوکس با استفاده از پروتکل SFTP

ما با استفاده از پروتکل SFTP، میتوانیم فایل‌های متفاوت را درون سیستم لینوکس خود منتقل کنیم. اگر به خاطر داشته باشید، ما به لینوکس خود یک IP دادیم. حال که لینوکس ما IP دارد، می‌توانیم به این ماشین SSH بزنیم و همچنین از طریق SFTP، عمل انتقال فایل را انجام دهیم.

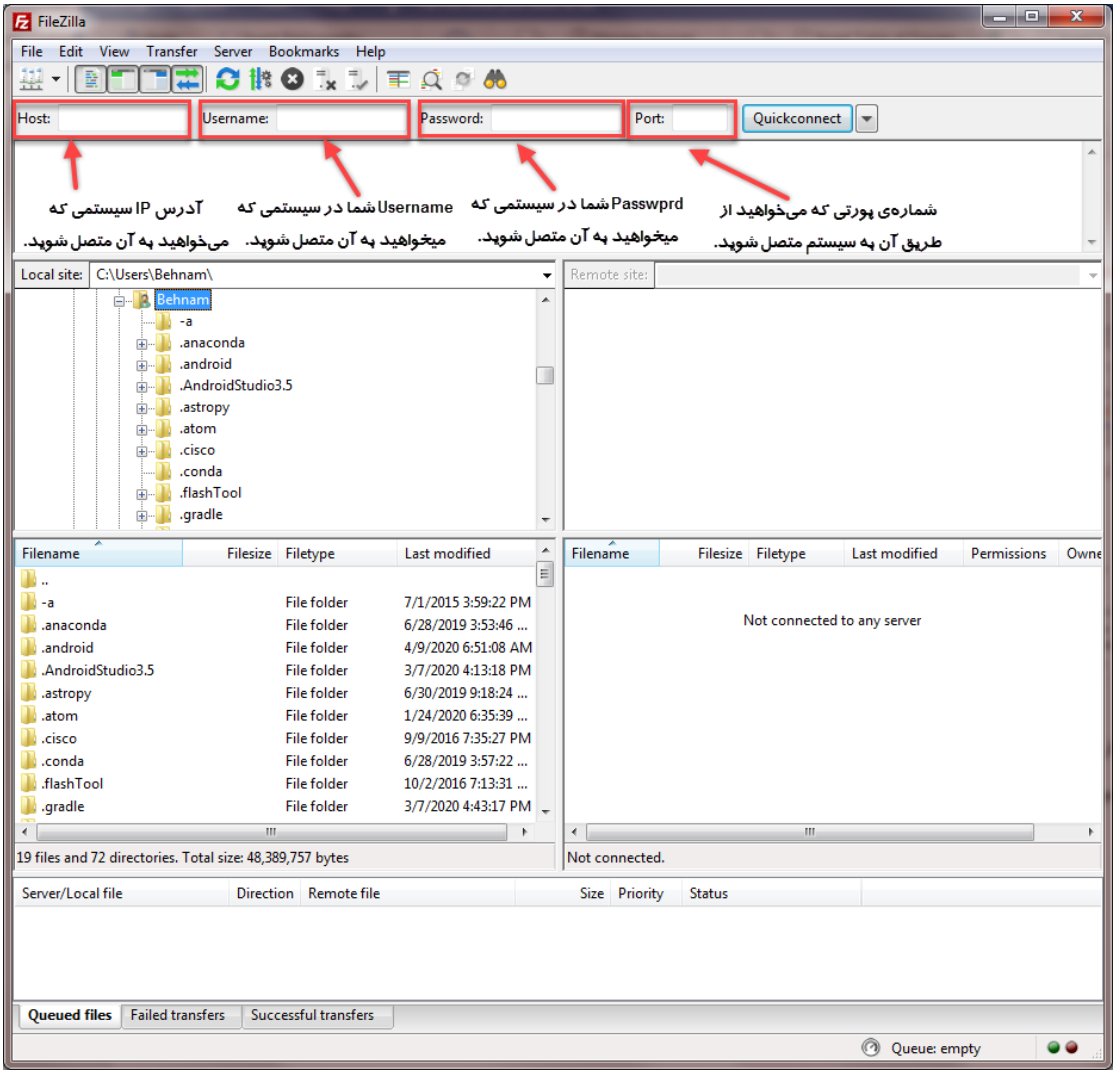
برای ایجاد ارتباط SFTP بین ویندوز و لینوکس، باید یک نرم‌افزار SFTP Client دانلود کنیم. ما از FileZilla که یک نرم‌افزار آزاد می‌باشد استفاده می‌کنیم. برای دانلود این نرم‌افزار، کافی است به وبسایت filezilla-project.org بروید و این برنامه را دانلود روی سیستم نصب کنید:



تصویر ۵ - صفحه‌ی اصلی سایت filezilla-project.org و دانلود این نرم‌افزار

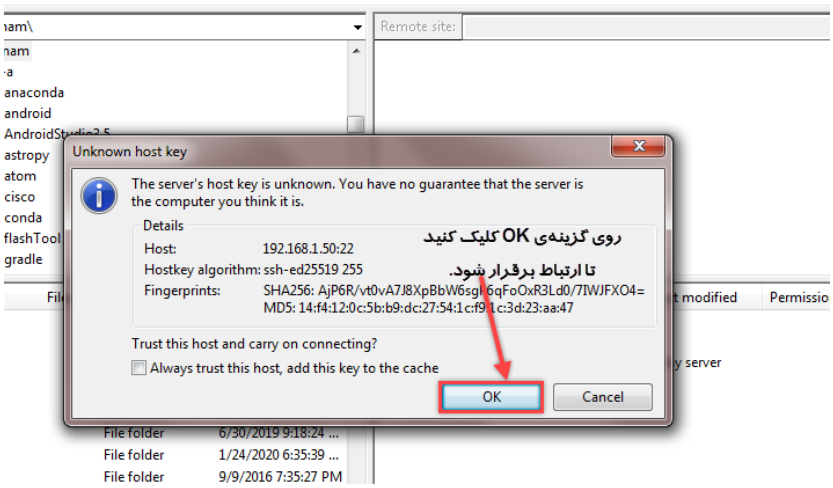
نصب این نرم‌افزار نکته‌ی خاصی ندارد، پس آن را توضیح نمی‌دهیم. پس از نصب نرم‌افزار و باز کردن آن، با صفحه‌ای نظیر تصویر ۶ مواجه می‌شوید. در این صفحه ما باید آدرس IP سیستمی که می‌خواهیم به آن متصل شویم، Username و Password خود در آن سیستم و همچنین شماره‌ی پورتی که در آن سیستم به SFTP اختصاص دارد را در قسمت‌های مشخص شده وارد کنیم.

در قسمت Host، باید IP که به ماشین مجازی خود اختصاص دادید را وارد کنید. در بخش «**تنظیمات کارت شبکه در لینوکس**» در مورد چگونگی بررسی IPهای هر کارت شبکه و همچنین چگونگی تنظیم IP روی سیستم صحبت کردیم. در قسمت Username و Password باید همان اطلاعاتی که هنگام وارد شدن به سیستم لینوکسی خود وارد می‌کنید را وارد کرده و در قسمت پورت، عدد ۲۲ را وارد کنید؛ چرا که SFTP به صورت پیش‌فرض از پورت ۲۲ استفاده می‌کند. در نهایت برای ارتباط با سیستم، دکمه‌ی Quickconnect را بزنید.

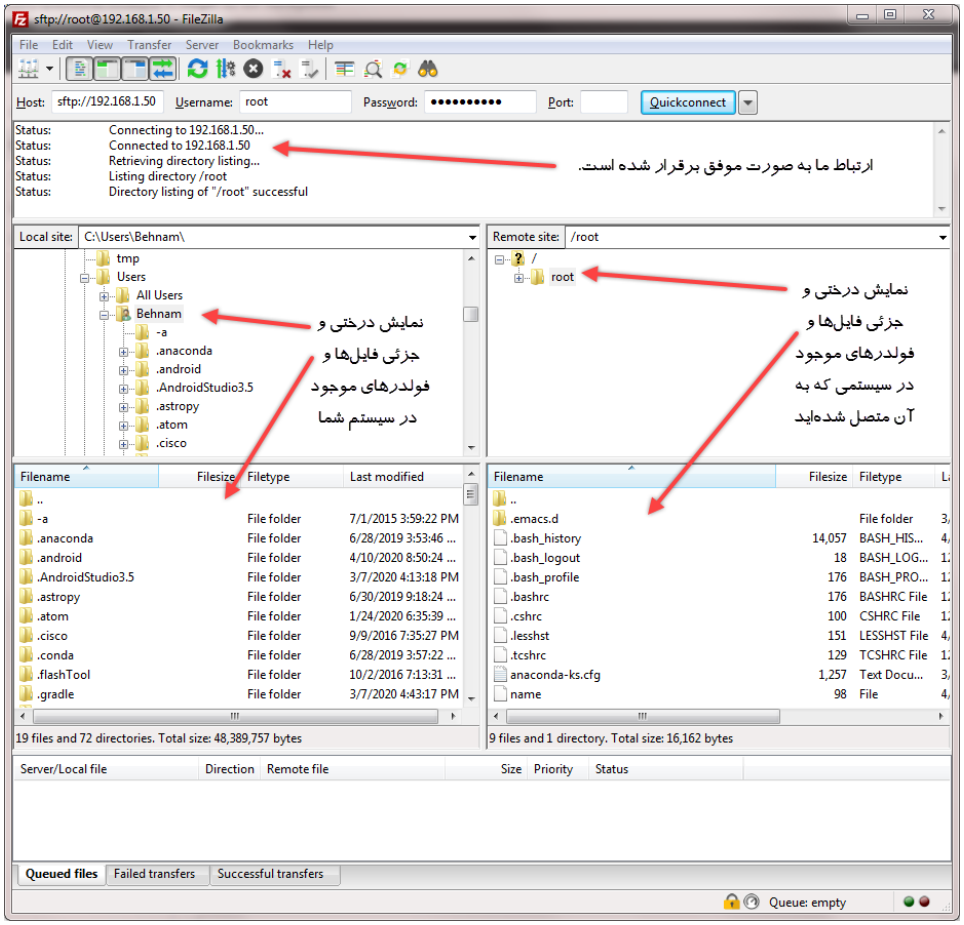


تصویر ۶ - صفحه‌ی اصلی نرم‌افزار FileZilla

پس از کلیک بر روی دکمه‌ی Quickconnect، در صورت صحیح بودن اطلاعات وارد شده، با صفحه‌ای نظیر تصویر ۷ مواجه می‌شوید. روی OK کلیک کنید تا ارتباط شما با ماشین مجازی برقرار شود.

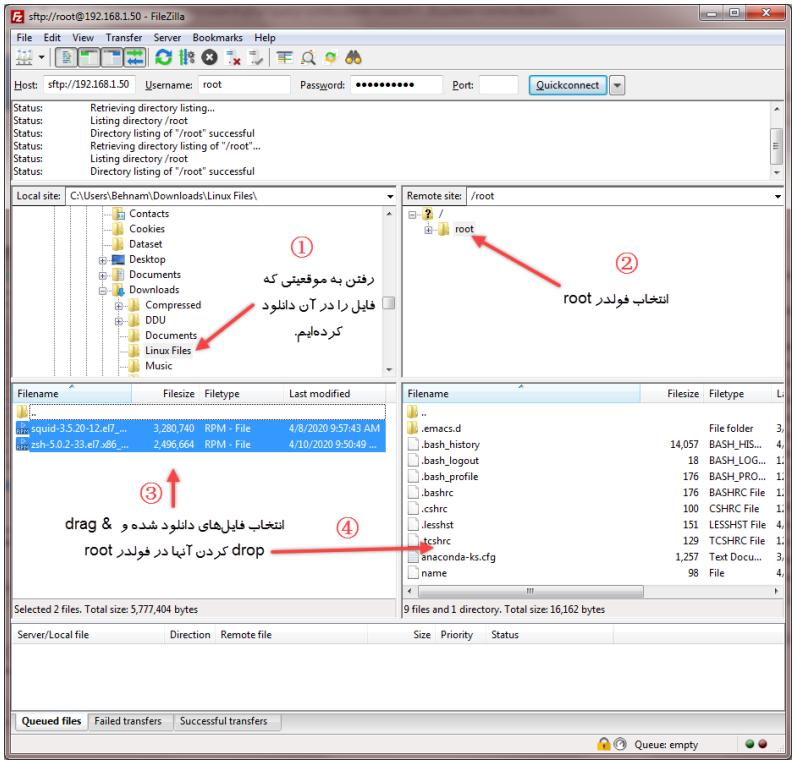


تصویر ۷- انتخاب گزینه‌ی OK جهت وصل شدن به ماشین مجازی

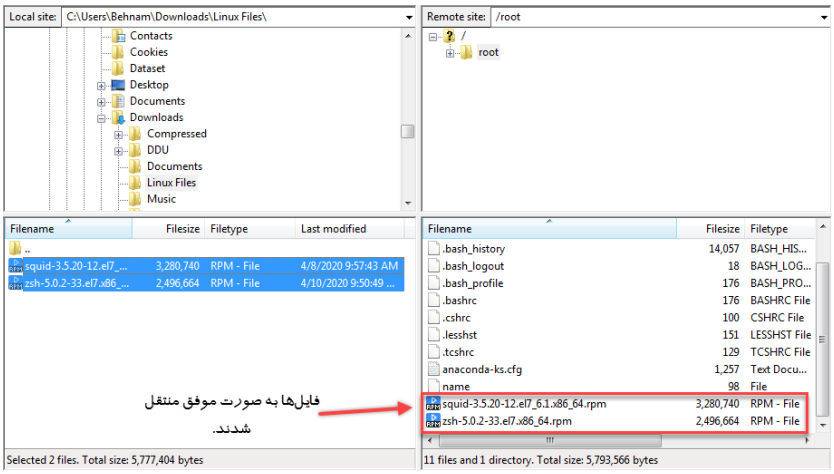


تصویر ۸- ارتباط ما با سیستم لینوکسی برقرار شده است.

همانطور که در تصویر ۸ می‌بینید، اکنون ارتباط ما به صورت موفق برقرار شده است. برای انتقال پکیج squid و zsh که در بخش قبل آن را دانلود کردیم، فایل را از طریق FileZilla روی سیستم خود پیدا کرده (فایل منیجر سمت چپ فایل‌های موجود در سیستم ما را نشان می‌دهد)؛ سپس فایل‌ها را انتخاب کرده و آن را در فولدر root سیستم لینوکسی که به آن متصل شدیم (فایل منیجر سمت راست نشان دهنده‌ی فایل‌های موجود در سیستم لینوکسی شما می‌باشد)، drag and drop کنیم. البته می‌توانید فایل‌ها را در هر فولدری که تمایل دارید منتقل کنید، اما ما فایل‌ها را به فولدر `root/` منتقل می‌کنیم.

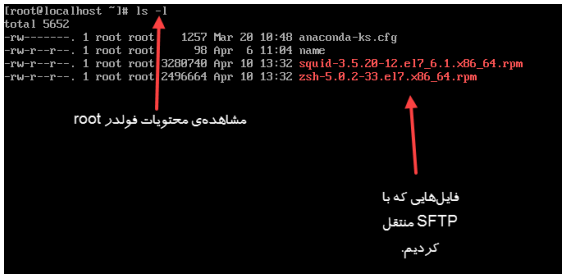


تصویر ۹- مراحل انتقال فایل به سیستم لینوکسی



تصویر ۱۰ - انتقال موفق فایل

حال بیایید از منتقل شدن فایل به سیستم لینوکسی خود، اطمینان حاصل کنیم. پس به سراغ سیستم لینوکسی خود می‌رویم و سپس با استفاده از دستور ls، محتویات این فولدر را بررسی می‌کنیم:



تصویر ۱۱- مشاهده‌ی فایل منتقل شده

همانطور که می‌بینید فایل به درستی منتقل شده و ما می‌توانیم از این فایل استفاده کنیم.

دستور RPM

ابزار اصلی برای کار کردن با فایل‌های RPM، دستور rpm می‌باشد. شما با استفاده از این ابزار می‌توانید فایل‌های rpm را روی سیستم نصب کنید، تغییر دهید یا پاک کنید. به صورت کلی، این دستور دارای syntax زیر می‌باشد:

```
rpm ACTION [OPTION] PACKAGE-FILE
```

یعنی شما باید دستور rpm را وارد کنید، سپس حتما یک action برای آن مشخص کنید. پس از آن، می‌توانید به صورت دلخواه یک آپشن برای آن مشخص کنید و در نهایت باید نام فایل پکیج مورد نظر را وارد کنید.

دریافت اطلاعات در مورد پکیج‌های نصب شده روی سیستم

برای مشاهده‌ی کلیه‌ی پکیج‌های rpm نصب شده روی سیستم، از دستور rpm -qa استفاده می‌کنیم:

```
[root@localhost ~]# rpm -qa
...
python-pyudev-0.15-9.el7.noarch
virt-what-1.18-4.el7.x86_64
rpm-build-libs-4.11.3-40.el7.x86_64
yum-plugin-fastestmirror-1.1.31-52.el7.noarch
rsyslog-8.24.0-38.el7.x86_64
selinux-policy-targeted-3.13.1-252.el7.noarch
```

همانطور که می‌بینید با استفاده از qa-، توانستیم کلیه‌ی پکیج‌های موجود در سیستم را مشاهده کنیم. اما بیایید کمی دقیق‌تر به این دستور نگاه کنیم. همانطور که گفتیم، دستور rpm از ما حتما یک اکشن می‌خواهد و پس از آن می‌توانیم یک آپشن نیز به آن بدهیم. اما در این دستور، اکشن ما کدام است و آپشن ما کدام؟

> به شکل ساده‌تر، این دستور، rpm -q -a می‌باشد. به طوری که q- نشان دهنده‌ی اکشن و a- نشان دهنده‌ی آپشن ما می‌باشد. ما در لینوکس مجبور نیستیم که برای قرار دادن اکشن، آپشن و... در کنار هم، از خط تیره‌های (-) جداگانه استفاده کنیم. به عبارت دیگر، دستور rpm -q -a دقیقا همان معنی rpm -qa را می‌دهد.

بیایید کمی بیشتر در مورد q- صحبت کنیم. با استفاده از این اکشن می‌توانیم درون پکیج منیجر جستجو کنیم یا به عبارت دیگر، پکیج منیجر را query کنیم و اطلاعاتی در مورد پکیج‌های نصب شده روی سیستم به دست آوریم. مثلا می‌توانیم با استفاده از q- و نوشتن نام یک نرم‌افزار جلوی آن، ببینیم که آن نرم‌افزار بر روی سیستم نصب شده یا نه:

```
[root@localhost ~]# rpm -q python
python-2.7.5-86.el7.x86_64
```

همانطور که می‌بینید با استفاده از q- و نوشتن نام پکیج python، دستور rpm به سراغ جستجو در دیتابیس برنامه‌های نصب شده رفته و در نهایت، به ما نام کامل پکیجی که روی سیستم نصب شده را نشان می‌دهد.

بیایید این کار را بار دیگر با نرم‌افزاری که میدانیم در سیستم وجود ندارد امتحان کنیم:

```
[root@localhost ~]# rpm -q squid
package squid is not installed
```

همانطور که می‌بینید، اگر نام پکیجی که به rpm -q می‌دهیم در سیستم نصب نشده باشد، rpm ما را از عدم وجود آن مطلع میسازد.

اما این تنها قابلیت q- نیست. با استفاده از i- پس از q-، می‌توانیم اطلاعات دقیق نظیر نسخه، معماری، زمان نصب و کلی اطلاعات دیگر در مورد یک پکیج دریافت کنیم. برای مثال:

```
[root@localhost ~]# rpm -qi bash
Name        : bash
Version     : 4.2.46
Release     : 33.el7
Architecture: x86_64
Install Date: Fri 20 Mar 2020 10:41:29 AM +0330
Group       : System Environment/Shells
Size        : 3667788
License     : GPLv3+
Signature   : RSA/SHA256, Fri 23 Aug 2019 01:50:37 AM +0430, Key ID 24c6a8a7f4a80eb5
Source RPM  : bash-4.2.46-33.el7.src.rpm
Build Date  : Thu 08 Aug 2019 04:39:21 PM +0430
Build Host  : x86-01.bsys.centos.org
Relocations : (not relocatable)
Packager    : CentOS BuildSystem <http://bugs.centos.org>
Vendor      : CentOS
URL         : http://www.gnu.org/software/bash
Summary     : The GNU Bourne Again shell
Description :
The GNU Bourne Again shell (Bash) is a shell or command language
interpreter that is compatible with the Bourne shell (sh). Bash
incorporates useful features from the Korn shell (ksh) and the C shell
(csh). Most sh scripts can be run by bash without modification.
```

همانطور که می‌بینید با استفاده از qi-، اطلاعات دقیقی در مورد پکیج bash به ما نمایش داده شد.

یکی دیگر از کارهایی که می‌توانیم با q- انجام دهیم، پیدا کردن dependencyهای هر پکیج می‌باشد. برای این کار، پس از q-، از R- استفاده می‌کنیم:

```
[root@localhost ~]# rpm -qR bash
...
libc.so.6(GLIBC_2.14)(64bit)
libc.so.6(GLIBC_2.15)(64bit)
libc.so.6(GLIBC_2.2.5)(64bit)
libc.so.6(GLIBC_2.3)(64bit)
...
```

همانطور که می‌بینید، qR- همه‌ی dependencyهای یک پکیج و نسخه‌ی آنها را به ما نشان می‌دهد.

اگر به خاطر داشته باشید، گفتیم که در لینوکس تنظیمات همه‌ی نرم‌افزارها و سرویس‌ها از طریق فایل‌های متنی انجام می‌شود. برای این که بدانیم فایل‌های متنی مربوط به تنظیمات هر پکیج چه فایل‌هایی هستند و در کجا قرار دارند، از q- و سپس c- استفاده می‌کنیم. به صورت زیر:

```
[root@localhost ~]# rpm -qc postfix
/etc/pam.d/smtp.postfix
/etc/postfix/access
/etc/postfix/canonical
/etc/postfix/generic
/etc/postfix/header_checks
/etc/postfix/main.cf
/etc/postfix/master.cf
/etc/postfix/relocated
/etc/postfix/transport
/etc/postfix/virtual
/etc/sasl2/smtpd.conf
```

همانطور که می‌بینید، با استفاده از qc-، کلیه‌ی فایل‌های مربوط به تنظیمات نرم‌افزار postfix به ما نمایش داده شد.

نصب کردن و پاک کردن پکیج‌ها

حال بیایید سراغ نصب یک پکیج rpm برویم. ما در بخش قبل، فایل rpm مربوط دو برنامه‌ی zsh و squid را دانلود کردیم و آن را داخل ماشین مجازی لینوکس منتقل کردیم. حال بیایید آن را نصب کنیم. ابتدا بیایید zsh را روی سیستم نصب کنیم. برای این کار، باید از i- استفاده کنیم. به صورت زیر:

```
[root@localhost ~]# rpm -i zsh-5.0.2-33.el7.x86_64.rpm
```

با وارد کردن این دستور، سیستم به مدت چند لحظه کنترل شل را از شما می‌گیرد و به سراغ نصب برنامه‌ی zsh می‌رود.

نکته: در bash، ما مفهومی به نام Tab Autocomplete داریم. یعنی کافیست ابتدای نام یک فایل، فولدر یا یک دستور را بنویسید و سپس دکمه‌ی Tab کیبورد را فشار دهید تا خود bash اقدام به کامل کردن ادامه‌ی نام آن فایل، فولدر یا دستور خاص کند. مثلا در مثال بالا، کافی‌است فقط zsh را نوشته و سپس دکمه‌ی Tab را بزنید تا خود bash نام این فایل را برایتان کامل کند.

بیایید از نصب این برنامه اطمینان حاصل کنیم:

```
[root@localhost ~]# rpm -q zsh
zsh-5.0.2-33.el7.x86_64
```

همانطور که می‌بینید، ما توانستیم با استفاده از دستور rpm و اکشن i-، پکیج برنامه‌ی zsh را روی سیستم نصب کنیم.

حال بیایید این پکیج را از روی سیستم حذف، یا uninstall کنیم. برای این کار، از اکشن e- استفاده می‌کنیم:

```
[root@localhost ~]# rpm -e zsh
[root@localhost ~]# rpm -q zsh
package zsh is not installed
```

همانطور که می‌بینید با استفاده از e-، توانستیم برنامه‌ی zsh را به سادگی از روی سیستم پاک کنیم.

نکته: توجه کنید که برای پاک‌کردن یک پکیج، نباید نام فایل rpm آن پکیج را به دستور rpm بدهید، بلکه فقط باید نام برنامه را به دستور rpm بدهید.

اما اینجا مشکلی هست. اگر کمی به نتیجه‌ی دستورهای نصب و حذف برنامه نگاه کنید، می‌بینید که ما هنگام نصب و هنگام پاک‌کردن برنامه، چیزی روی صفحه مشاهده نمی‌کنیم؛ بلکه سیستم فقط به مدت چند لحظه کنترل شِل را از دست ما می‌گیرد و پس از انجام کار درخواستی، کنترل را به ما باز می‌گرداند. اگر راهی بود که می‌توانستیم چگونگی نصب یک پکیج را ببینیم، خیلی بهتر میشد.

برای دیدن چگونگی نصب، آپگرید یا پاک‌کردن یک پکیج، باید vh- را به اکشن مورد نظر (i-، U- یا e-) اضافه کنید. بیایید بار دیگر پکیج zsh را روی سیستم نصب کنیم. این بار، از اکشن U- و آپشن‌های vh- برای نصب استفاده می‌کنیم. به صورت زیر:

```
[root@localhost ~]# rpm -Uvh zsh-5.0.2-33.el7.x86_64.rpm
Preparing...                          ################################# [100%]
Updating / installing...
   1:zsh-5.0.2-33.el7                 ################################# [100%]
```

همانطور که می‌بینید با اضافه کردن آپشن vh-، می‌توانیم چگونگی انجام عملیات را در خروجی ببینیم.

شاید از خود بپرسید که چرا برای نصب، به جای i- از U- استفاده کردیم. U- ابتدا بررسی می‌کند که پکیج برروی سیستم وجود دارد یا نه؛ اگر وجود داشت، به سراغ آپدیت آن پکیج می‌رود و اگر وجود نداشت، آن را نصب می‌کند. به همین دلیل، استفاده از U- معمول تر است.

> ما می‌توانیم با اضافه کردن آپشن vh- به e-، مراحل پاک کردن یا uninstall کردن یک پکیج از روی سیستم را نیز مشاهده کنیم، اما تست این کار را به شما می‌سپاریم.

خوب، حال که با چگونگی نصب یک برنامه از طریق rpm آشنا شدیم، بیایید به سراغ نصب برنامه‌ی squid برویم:

```
[root@localhost ~]# rpm -Uvh squid-3.5.20-12.el7_6.1.x86_64.rpm
error: Failed dependencies:
      libecap.so.3()(64bit) is needed by squid-7:3.5.20-12.el7_6.1.x86_64
      perl(DBI) is needed by squid-7:3.5.20-12.el7_6.1.x86_64
      perl(Data::Dumper) is needed by squid-7:3.5.20-12.el7_6.1.x86_64
      perl(Digest::MD5) is needed by squid-7:3.5.20-12.el7_6.1.x86_64
      squid-migration-script is needed by squid-7:3.5.20-12.el7_6.1.x86_64
```

همانطور که می‌بینید، ما موفق به نصب پکیج squid روی سیستم نشدیم. اگر به خروجی دستور نگاه کنید، می‌بینید که برای نصب برنامه‌ی squid، ابتدا باید dependencyهای آن را حل کنیم. یعنی باید تک تک مواردی که در خروجی از آن نام برده شده را روی سیستم نصب کنیم. این یک مشکل بزرگ است، چون ممکن است هر کدام از این برنامه‌ها نیز احتیاج به یک سری برنامه‌ی دیگر برای اجرا داشته باشند. با این که دستور rpm به ما می‌گوید که باید دقیقا چه پکیج‌هایی را برای رفع dependency دانلود کنیم، اما به ما کمکی در نصب این پکیج‌ها نمی‌کند.

دلایل به وجود آمدن مشکلات Dependency

دلایل به وجود آمدن مشکلات dependency و conflict را می‌توان به چند بخش تقسیم کرد:

- **عدم وجود لایبرری یا برنامه‌های جانبی**

اکثر برنامه‌ها برای این که به درستی کار کنند، احتیاج به یک سری لایبرری یا برنامه‌های جانبی دارند. اگر این لایبری‌ها یا نرم‌افزارها روی سیستم نصب نباشند، برنامه نمی‌تواند به درستی اجرا شود و ما به مشکل dependency بر می‌خوریم.

- **ناسازگاری لایبرری یا برنامه‌ی جانبی با پکیج مورد نظر**

گاهی اوقات ما لایبرری یا برنامه‌ی مورد نیاز را روی سیستم داریم، اما نسخه‌ی لایبرری ما با نسخه‌ای که برنامه به آن نیاز دارد همخوانی ندارد.

- **وجود فایل‌های تکراری**

گاهی اوقات یک پکیج شامل فایل‌هایی باشد که از قبل روی سیستم وجود دارند، اما متعلق یک پکیج دیگر می‌باشند. این امر باعث به وجود آمدن کانفلیکت می‌شود.

- **عدم تطبیق نام پکیج‌ها**

گفتیم که هر پکیج یک اسم دارد. پکیج منیجرها با توجه به این اسم‌ها، به دنبال dependencyها و کانفلیکت‌ها می‌گردند. اما گاهی اوقات در برخی از توزیع‌ها، یک پکیج دارای اسم متفاوتی از توزیع دیگر خواهد بود و این امر مشکلاتی را برای ما ایجاد می‌کند.

کار کردن با YUM

با این که دستور rpm بسیار کاربردی است، اما محدودیت‌های زیادی دارد. مثلا همانطور که دیدیم، با این که دستور rpm ما را از نام dependencyها مطلع می‌سازد، ولی به ما کمکی در نصب dependencyها نمی‌کند. برای حل این محدودیت‌ها، هر توزیع لینوکسی برای خود یک سری انبار یا repository که پر از پکیج‌های متفاوت می‌باشد ایجاد کرده است. پکیج‌های موجود در این ریپازیتوری‌ها تست شده می‌باشند و کاملا با هر نسخه از آن توزیع خاص، همخوانی دارند (هر نسخه از هر توزیع، یک ریپازیتوری جداگانه از نرم‌افزارهای سازگار با خود دارد).

برای اتصال به این ریپازتوری‌ها احتیاج به یک ابزار داریم. توزیع‌های Red Hat-based از ابزار yum جهت کارکردن با ریپازیتوری‌ها استفاده می‌کنند. ابزار yum به ما این امکان را می‌دهد که نرم‌افزارهای مورد نظر را با استفاده از ریپازیتوری‌های رسمی یک توزیع، نصب، آپدیت، پاک و... کنیم.

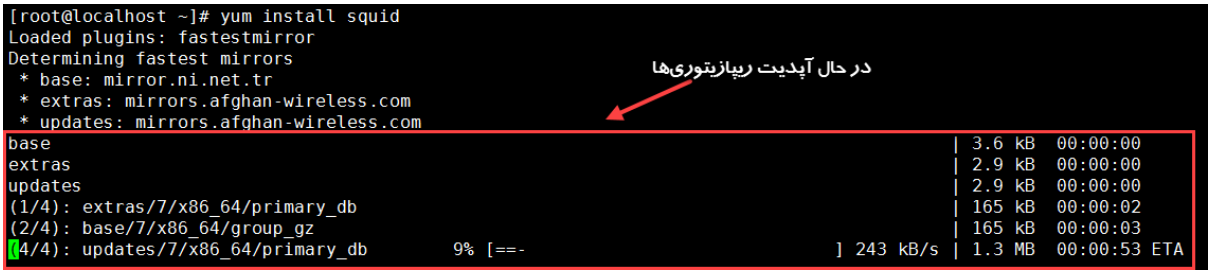
نکته: اتصال به این ریپازیتوری‌ها و استفاده از yum، نیازمند وجود ارتباط اینترنت می‌باشد.

نصب نرم‌افزار

فرض کنید می‌خواهیم نرم‌افزار squid را با کمک yum روی سیستم نصب کنیم. برای نصب یک نرم‌افزار توسط yum، کافی است دستور yum install و همچنین نام نرم‌افزار مورد نظر را بنویسیم:

```
[root@localhost ~]# yum install squid
```

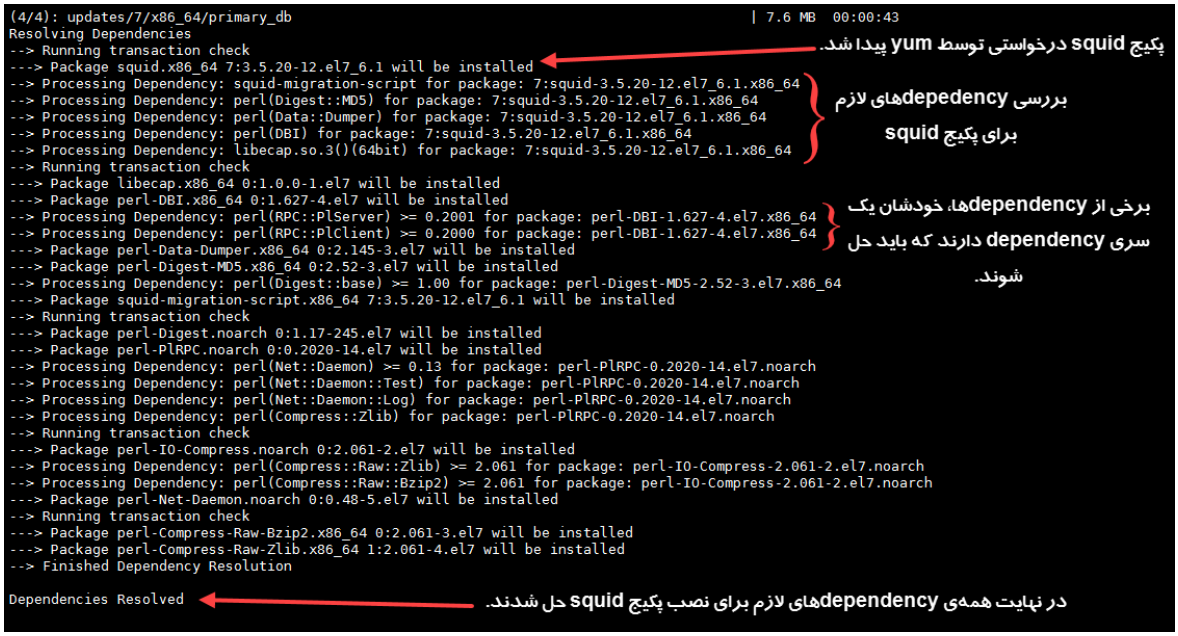
به محض وارد کردن این دستور، با نمایی نظیر تصویر ۱۲ مواجه می‌شویم. yum ابتدا به سراغ پیدا کردن نزدیک‌ترین سرورها به موقعیت شما می‌رود و سپس با اتصال به این سرورها، دیتابیس‌های ریپازتیوری‌های موجود روی سیستم شما را آپدیت می‌کند. این امر باعث می‌شود که برای هر بار جستجو برای یک پکیج، مجبور به اتصال مستقیم به این سرورها نباشیم.



تصویر ۱۲- دستور yum در حال آپدیت ریپازیتوری‌ها

همانطور که در تصویر ۱۳ می‌بینید، پس از آپدیت ریپازتوری‌ها، yum به سراغ پیدا کردن نرم‌افزار خواسته شده توسط ما و همچنین جستجو و پیدا کردن dependencyهای آن نرم‌افزار رفت و علاوه بر آن، dependencyهای مربوط به dependencyهای نرم‌افزار درخواستی را نیز برای ما پیدا کرد.

همانطور که در تصویر ۱۴ می‌بینید، پس از پیدا کردن نرم‌افزار و حل کردن همه‌ی dependencyها، yum لیستی از همه‌ی پکیج‌هایی که باید دانلود کند به ما نشان می‌دهد و حجم دانلود و همچنین حجم آنها پس از نصب را به ما می‌گوید.



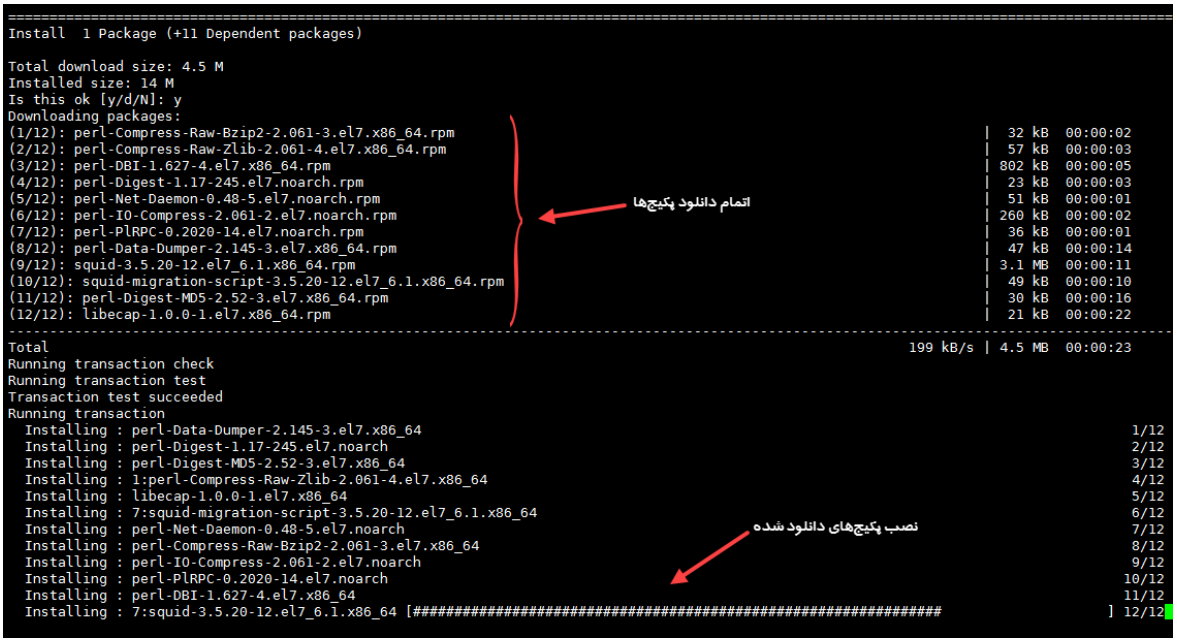
تصویر ۱۳- دستور yum هم نرم‌افزار squid را پیدا کرد و هم dependencyهای آن را حل کرد.

تصویر ۱۴- لیست پکیج‌هایی که باید دانلود شوند و انتظار سیستم برای تایید یا رد انجام این عملیات

همانطور که می‌بینید، سیستم از ما می‌خواهد که عملیات نصب را تایید کنیم، رد کنیم یا فقط پکیج‌ها را دانلود کنیم اما نصب نکنیم. برای تایید و رفتن به سمت دانلود و نصب پکیج‌های ذکر شده، حرف y را وارد کرده و سپس دکمه‌ی Enter را میزنیم. برای رد کردن دانلود و نصب، حرف n را وارد می‌کنیم و سپس دکمه‌ی Enter را می‌زنیم و اگر بخواهیم فقط پکیج‌های لیست شده را دانلود کنیم اما آنها را نصب نکنیم، حرف d را وارد کرده و سپس دکمه‌ی Enter را می‌زنیم.

از آنجایی که ما می‌خواهیم squid را روی سیستم نصب کنیم، عملیات را تایید میکنیم:

تصویر ۱۵- تایید عملیات نصب و شروع دانلود



تصویر ۱۶- نصب پکیج‌های دانلود شده

تصویر ۱۷- پایان موفق مراحل نصب

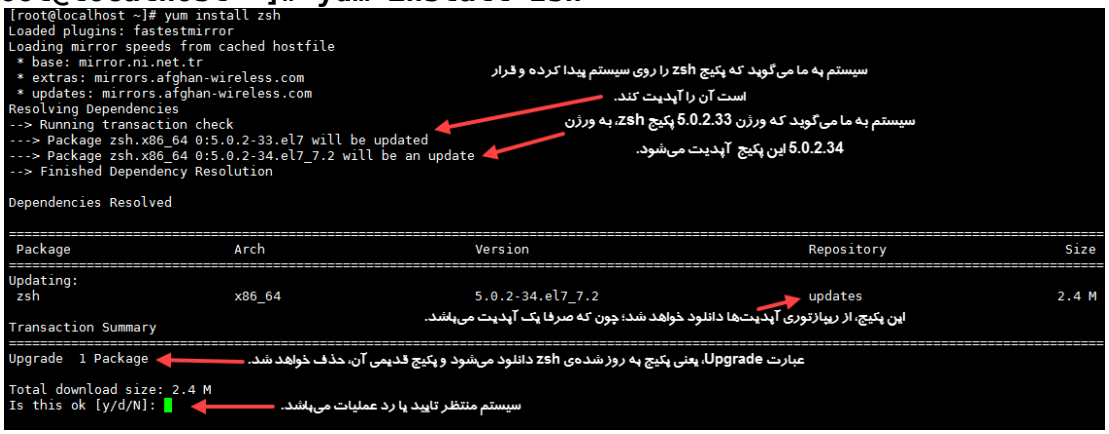
همانطور که می‌بینید، با استفاده از دستور yum، توانستیم به سادگی و بدون هیچ دردسری، پکیج squid را روی سیستم نصب کنیم.

نکته: ریپازیتوری‌های رسمی CentOS، معمولا نسخه‌های قدیمی‌تر از نرم‌افزارها را درون خود دارند. مثلا اگر به سایت squid بروید، می‌بینید که آخرین نسخه‌ی این نرم‌افزار، نسخه‌ی ٤,١٠ می‌باشد، اما نسخه‌ی موجود در ریپازیتوری CentOS 7، نسخه‌ی ٣,٥ می‌باشد. دلیل این امر، stability می‌باشد. توزیع‌هایی که مخصوص سرور هستند، معمولا نسخه‌ای از نرم‌افزار را که بیشترین میزان stability برای آن نسخه‌ی خاص از سیستم دارد را درون ریپازیتوری‌های خود قرار می‌دهند. اگر با یک دلیل مناسب، نیاز به جدیدترین نسخه‌ی یک نرم‌افزار دارید، باید آن را از طریق source code نصب کنید. کلمه‌ی کلیدی در اینجا، «دلیل مناسب» می‌باشد.

اما فرض کنید یک پکیج روی سیستم نصب باشد، اما ما از نصب آن اطلاعی نداشته باشیم و مستقیما به سراغ دستور `yum install` برویم. در این حالت، yum چه رفتاری از خود نشان خواهد داد؟

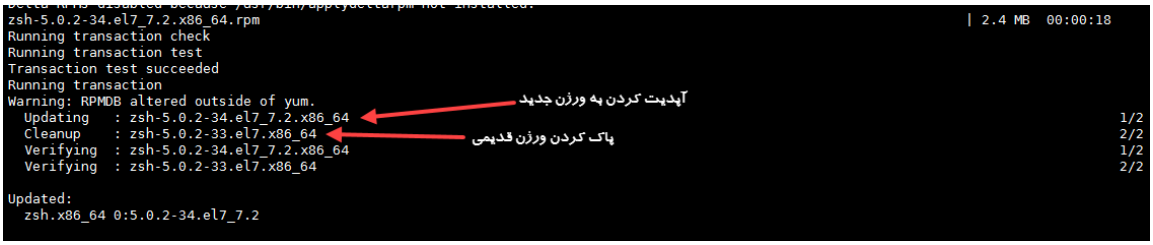
بیایید این کار را روی پکیج zsh که در بخش قبل با استفاده از دستور rpm نصب کردیم امتحان کنیم:

```
[root@localhost ~]# yum install zsh
```



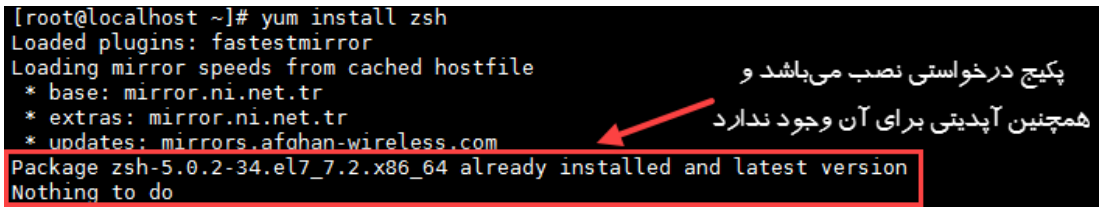
تصویر ۱۸- عملکرد `yum install` وقتی پکیج از قبل روی سیستم نصب باشد.

همانطور که می‌بینید، چون پکیج zsh از قبل روی سیستم نصب شده بود و همچنین در ریپازیتوری سیستم وجود داشت، سیستم به سراغ آپدیت کردن این پکیج، و پاک کردن نسخه‌ی قدیمی این پکیج رفت. اگر عملیات را تایید کنید، با نمایی نظیر تصویر ۱۹ مواجه می‌شوید:



تصویر ۱۹- نصب آپدیت پکیج و پاکسازی ورژن قبلی آن

اما اگر آپدیتی برای آن پکیج وجود نداشته باشد، yum چه رفتاری از خود نشان می‌دهد؟ در آن حالت، yum به ما می‌گوید پکیج مورد نظر روی سیستم نصب می‌باشد و هیچ آپدیتی برای آن وجود ندارد:



تصویر ۲۰- نتیجه‌ی دستور `yum install` وقتی پکیج درخواستی نصب و آپدیت باشد.

آپدیت نرم‌افزار و سیستم

برای آپدیت کردن یک پکیج نصب شده روی سیستم، از دستور `yum update` استفاده می‌کنیم. نحوه‌ی آپدیت کردن یک پکیج، دقیقا شبیه چیزی که در تصویر ۱۸ و ۱۹ می‌بینید می‌باشد. با این حال، بیایید با استفاده از این دستور، پکیج rsyslog را آپدیت کنیم:

```
[root@localhost ~]# yum update rsyslog
```

خروجی این دستور دقیقا شبیه تصویر ۱۸ می‌باشد و شما می‌توانید آپدیت پکیج را انجام دهید یا آن را کنسل

کنید.

اگر دستور `yum update` را بدون مشخص کردن نام یک نرم‌افزار اجرا کنیم، yum به سراغ جستجوی آپدیت برای کلیه‌ی پکیج‌های نصب شده روی سیستم، اعم از کرنل لینوکس، می‌رود. این امر بسیار کاربردی است، چون ما می‌توانیم با نوشتن یک دستور، همه‌ی پکیج‌های موجود در سیستم را آپدیت کنیم. برای این کار، کافی است دستور زیر را اجرا کنید:

```
[root@localhost ~]# yum update
```

در تصویر ۲۱، فقط بخشی از خروجی این دستور را می‌بینید و طبق معمول، می‌توانید درخواست آپدیت را تایید، رد، یا فقط فایل‌های آپدیت را دانلود کنید.

```
 selinux-policy                noarch  3.13.1-252.el7_7.6  updates  492 k
 selinux-policy-targeted       noarch  3.13.1-252.el7_7.6  updates  7.0 M
 sqlite                        x86_64  3.7.17-8.el7_7.1    updates  394 k
 sudo                          x86_64  1.8.23-4.el7_7.2    updates  842 k
 systemd                       x86_64  219-67.el7_7.4      updates  5.1 M
 systemd-libs                  x86_64  219-67.el7_7.4      updates  411 k
 systemd-sysv                  x86_64  219-67.el7_7.4      updates   89 k
 tuned                         noarch  2.11.0-5.el7_7.1    updates  268 k
 tzdata                        noarch  2019c-1.el7         updates  493 k
 util-linux                    x86_64  2.23.2-61.el7_7.1   updates  2.0 M

Transaction Summary
================================================================================
Install   1 Package
Upgrade  52 Packages

Total download size: 114 M
Is this ok [y/d/N]:
```

تصویر ۲۱- نتیجه‌ی اجرای دستور `yum update`

جستجو برای یک نرم‌افزار

ما با استفاده از yum، می‌توانیم در ریپازیتوری‌ها جستجو کنیم. مثلا فرض کنید دنبال پکیجی به نام httpd هستیم. برای این کار، کافی است از دستور `yum search` استفاده کنیم:

```
[root@localhost ~]# yum search httpd
```

```
[root@localhost ~]# yum search httpd
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
 * base: mirror.centos.jt.iq
 * extras: mirror.centos.jt.iq
 * updates: mirror.centos.jt.iq
============================================================ N/S matched: httpd ============================================================
keycloak-httpd-client-install.noarch : Tools to configure Apache HTTPD as Keycloak client
libmicrohttpd-devel.i686 : Development files for libmicrohttpd
libmicrohttpd-devel.x86_64 : Development files for libmicrohttpd
libmicrohttpd-doc.noarch : Documentation for libmicrohttpd
python2-keycloak-httpd-client-install.noarch : Tools to configure Apache HTTPD as Keycloak client
httpd.x86_64 : Apache HTTP Server
httpd-devel.x86_64 : Development interfaces for the Apache HTTP server
httpd-manual.noarch : Documentation for the Apache HTTP server
httpd-tools.x86_64 : Tools for use with the Apache HTTP Server
libmicrohttpd.i686 : Lightweight library for embedding a webserver in applications
libmicrohttpd.x86_64 : Lightweight library for embedding a webserver in applications
mod_auth_mellon.x86_64 : A SAML 2.0 authentication module for the Apache Httpd Server
mod_dav_svn.x86_64 : Apache httpd module for Subversion server

  Name and summary matches only, use "search all" for everything.
```

کلیه‌ی پکیج‌هایی که در آن نام httpd وجود دارد.

تصویر ۲۲- جستجو در ریپازیتوری با استفاده از دستور `yum search`

پاک‌کردن یک نرم‌افزار

ما با استفاده از yum، می‌توانیم پکیج‌های نصب شده را نیز به راحتی پاک کنیم. برای مثال:

```
[root@localhost ~]# yum remove zsh
```

با اجرای این دستور، yum به سراغ پاک کردن پکیج zsh می‌رود و طبق معمول از شما می‌خواهد که درخواست حذف را تایید یا رد کنید.

نکته‌ی قابل توجه در مورد دستور yum remove این است که این دستور موقع پاک کردن یک پکیج، هر پکیج دیگری که از آن پکیج استفاده کند را نیز پاک می‌کند؛ یا به عبارت دیگر، اگر پکیج A، dependency پکیج‌های B و C باشد، موقع درخواست پاک کردن پکیج A، پکیج B و C نیز پاک خواهند شد. دستور زیر را در نظر بگیرید:

```
[root@localhost ~]# yum remove perl
```

تصویر ۲۳- نتیجه‌ی اجرای دستور yum remove perl

همانطور که در تصویر ۲۳ می‌بینید، درخواست حذف پکیج perl، باعث می‌شود که yum به سراغ هر پکیجی که از perl استفاده می‌کند رفته و آنها را نیز پاک کند. مثلا می‌بینید که اگر perl را پاک کنیم، yum پکیج squid را نیز پاک خواهد کرد. این قابلیت بسیار کاربردی می‌باشد، چون به ما نشان می‌دهد که با حذف کردن یک پکیج، چه بلایی ممکن است سر سیستم بیاید.

نصب ریپازیتوری‌های جانبی

حال که با ویژگی‌های yum بیشتر آشنا شدیم، بیایید به سراغ نصب یک برنامه‌ی دیگر برویم. نرم‌افزار w3m، یک مرورگر وب می‌باشد که تحت command line کار می‌کند. ما می‌خواهیم این نرم‌افزار را روی سیستم نصب کنیم. برای این کار:

```
[root@localhost ~]# yum install w3m
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
 * base: centos.turhost.com
 * extras: centos.turhost.com
 * updates: ftp.linux.org.tr
No package w3m available.
Error: Nothing to do
```

همانطور که می‌بینید، yum نتوانست پکیج w3m را در ریپازیتوری‌های خود پیدا کند. این امر، یک نکته‌ی مهم را به ما نشان می‌دهد. با این که در ریپازیتوری‌های yum پکیج‌های زیاد و معروفی وجود دارند، اما ممکن است

نیاز به پکیجی داشته باشیم که در این ریپازیتوری‌ها موجود نمی‌باشد. اینجاست که باید به سراغ ریپازتوری‌های Third Party و جانبی برویم.

اگر به خاطر داشته باشید، گفتیم که yum برای دانلود نرم‌افزار به یک سری ریپازیتوری متصل می‌شود. دستور yum از فولدر /etc/yum.repos.d برای نگه داشتن فایل‌هایی که آدرس URL ریپازیتوری‌های متفاوت را درون خود دارد، استفاده می‌کند. بیایید محتویات این فولدر را ببینیم:

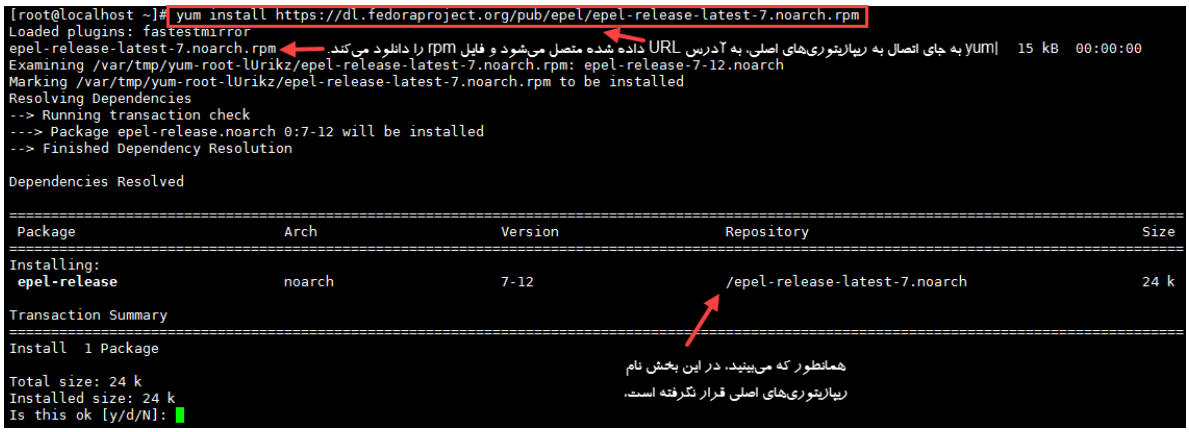
```
[root@localhost ~]# ls -l /etc/yum.repos.d/
total 32
-rw-r--r--. 1 root root 1664 Sep  5  2019 CentOS-Base.repo
-rw-r--r--. 1 root root 1309 Sep  5  2019 CentOS-CR.repo
-rw-r--r--. 1 root root  649 Sep  5  2019 CentOS-Debuginfo.repo
-rw-r--r--. 1 root root  314 Sep  5  2019 CentOS-fasttrack.repo
-rw-r--r--. 1 root root  630 Sep  5  2019 CentOS-Media.repo
-rw-r--r--. 1 root root 1331 Sep  5  2019 CentOS-Sources.repo
-rw-r--r--. 1 root root 6639 Sep  5  2019 CentOS-Vault.repo
```

اگر یکی از این فایل‌ها را باز کنید، خواهید دید که در این فایل‌ها، یک سری آدرس URL وجود دارد که شما را به لیستی از آدرس‌های URL که شامل لینک‌هایی به سرورهای ریپازیتوری مخصوص نسخه‌ی کنونی سیستم‌عامل شما می‌باشد، می‌برد. yum هنگام اجرا، با خواندن این فایل و وصل شدن به URL موجود در فایل، نزدیک‌ترین ریپازیتوری به موقعیت شما را پیدا کرده و عملیات مورد نظر شما را انجام می‌دهد.

همانطور که گفتیم، ما فقط محدود به ریپازیتوری‌های پیش‌فرض موجود در سیستم نیستیم و می‌توانیم ریپازیتوری‌های دیگری را نیز به yum معرفی کنیم. یکی از ریپازیتوری‌های معروف و کارآمد که میتوانیم به سیستم اضافه کنیم، ریپازتوری EPEL می‌باشد. اگر به سایت مربوط به ریپازیتوری EPEL مراجعه کنید، می‌بینید که برای نصب این ریپازیتوری، باید دستور زیر را در سیستم وارد کنید:

```
[root@localhost ~]# yum install https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm
```

پس از وارد کردن این دستور، با نمایی نظیر تصویر ۲۴ مواجه می‌شوید. همانطور که می‌بینید، با ارائه‌ی آدرس URL به دستور yum install، دستور yum به جای متصل شدن به ریپازیتوری‌های درونی خود، به آدرس URL ارائه شده متصل شده و فایل لازم جهت نصب ریپازیتوری EPEL را دانلود کرده و طبق معمول، از ما در خواست تایید یا رد نصب این برنامه را می‌کند. البته برای اضافه کردن ریپازیتوری به سیستم روش‌های دیگری نیز وجود دارد، اما ما به آنها نمی‌پردازیم.



تصویر ۲۴- نصب ریپازیتوری EPEL

حال که ریپازیتوری EPEL را نصب کردیم، بیایید بار دیگر نگاهی به فولدر /etc/yum.repos.d/ بیاندازیم:

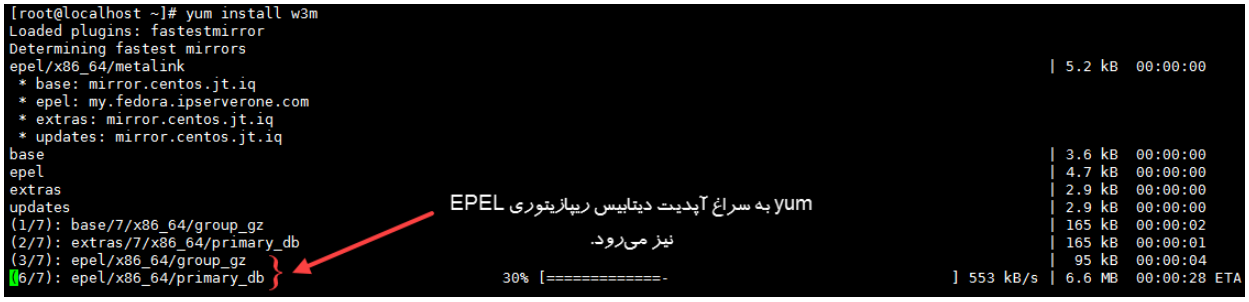
```
[root@localhost ~]# ls -l /etc/yum.repos.d/
total 40
-rw-r--r--. 1 root root 1664 Sep  5  2019 CentOS-Base.repo
-rw-r--r--. 1 root root 1309 Sep  5  2019 CentOS-CR.repo
-rw-r--r--. 1 root root  649 Sep  5  2019 CentOS-Debuginfo.repo
-rw-r--r--. 1 root root  314 Sep  5  2019 CentOS-fasttrack.repo
-rw-r--r--. 1 root root  630 Sep  5  2019 CentOS-Media.repo
-rw-r--r--. 1 root root 1331 Sep  5  2019 CentOS-Sources.repo
-rw-r--r--. 1 root root 6639 Sep  5  2019 CentOS-Vault.repo
-rw-r--r--. 1 root root 1050 Sep 18  2019 epel.repo
-rw-r--r--. 1 root root 1149 Sep 18  2019 epel-testing.repo
```

همانطور که می‌بینید، اکنون فایل‌های مربوط به ریپازیتوری EPEL نیز در این فولدر موجود می‌باشد.

حال که یک ریپازیتوری جدید اضافه کردیم، می‌توانیم باری دیگر اقدام به نصب w3m کنیم:

```
[root@localhost ~]# yum install w3m
```

همانطور که می‌بینید، این بار yum به سراغ آپدیت کردن دیتابیس ریپازیتوری EPEL نیز می‌رود:



تصویر ۲۵- استفاده از yum پس از نصب ریپازیتوری EPEL

در تصویر ۲۶، می‌بینید که yum، پکیج w3m را در ریپازیتوری EPEL پیدا کرد. نکته‌ی جالب این است که yum برای حل dependencyها، برخی از پکیج‌ها را از خود ریپازیتوری EPEL و برخی از پکیج‌ها را از ریپازیتوری‌های اصلی دانلود می‌کند:

تصویر ۲۶- دانلود پکیج w3m پس از نصب ریپازیتوری EPEL

نکته: در CentOS 8، برنامه‌ی dnf جایگزین yum شده است. این دو برنامه از همه نظر شبیه هم هستند، اما برنامه‌ی dnf سریع تر است.

دانلود از وب با استفاده از wget

در بخش‌های قبلی، ما پس از دانلود فایل rpm نرم‌افزار squid، آن را با استفاده از SFTP به لینوکس منتقل کردیم. پرواضح است که این روش در خیلی از اوقات، روش منطقی و مناسبی نیست، اگر قرار است فایلی را ابتدا دانلود کرده و سپس آن را به لینوکس منتقل کنیم، چرا فایل را مستقیما از خود لینوکس دانلود نکنیم؟

این کاری است که ابزار wget انجام می‌دهد. wget نرم‌افزاری است که می‌تواند با پروتکل‌های HTTP، HTTPS و FTP محتوای متفاوت را از وب سرورها دانلود کنند. بیایید بدون اتلاف وقت، به سراغ نصب نرم‌افزار wget برویم:

```
[root@localhost ~]# yum install wget
...
Installed:
  wget.x86_64 0:1.14-18.el7_6.1

Complete!
```

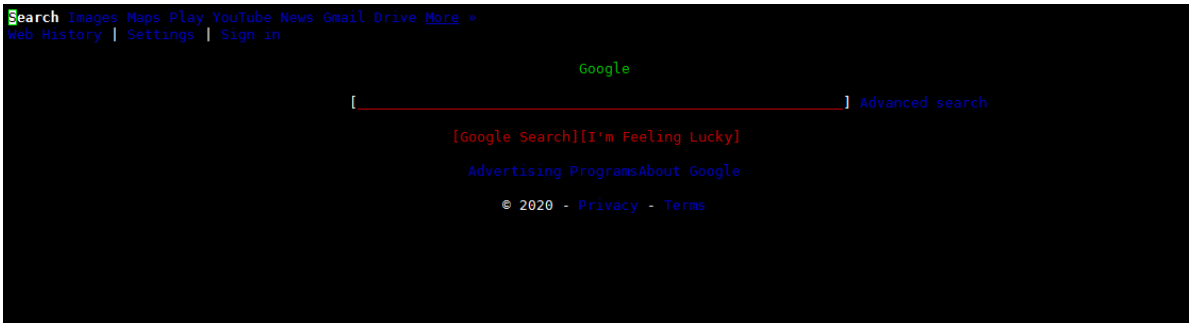
حال که این نرم‌افزار را نصب کردیم، بیایید با آن چیزی دانلود کنیم. مثلا اگر به wget بگوییم که google.com را دانلود کند چه می‌شود؟ آیا با این کار معنی واقعی زندگی را می‌فهمیم؟ بیایید امتحان کنیم:

```
[root@localhost ~]# wget google.com
...
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/html]
Saving to: ‘index.html’

    [ <=>                                                                 ]
11,878      --.-K/s   in 0s

2020-04-15 10:21:11 (107 MB/s) - ‘index.html’ saved [11878]
```

همانطور که می‌بینید، wget به google.com متصل شد و فایلی به نام index.html را دانلود کرد. بیایید این فایل را با استفاده از w3m که در بخش قبل دانلود کردیم، مشاهده کنیم:

```
[root@localhost ~]# w3m index.html
```

همانطور که در تصویر ۲۷ می‌بینید، wget فایل HTML صفحه‌ی اول google.com را برای ما دانلود کرده است. البته نمای این صفحه با چیزی که به آن عادت دارید متفاوت است، اما این به دلیل مشاهده‌ی فایل HTML در محیط شل می‌باشد.



تصویر ۲۷- مشاهده‌ی فایل HTML دانلود شده با استفاده از w3m

نکته: برای خروج از w3m، دکمه‌ی q و سپس دکمه‌ی y را روی کیبورد خود فشار دهید.

دانلود یک فایل نیز پروسه‌ای کاملا مشابه دارد. مثلا:

```
[root@localhost ~]# wget http://ipv4.download.thinkbroadband.com/5MB.zip
```

همانطور که می‌بینید به محض وارد کردن این دستور، wget به سراغ دانلود این فایل می‌رود. چیزی که در مثال قبل فرصت دیدن آن را نداشتیم این است که wget به صورت اتوماتیک، وضعیت دانلود فایل را به ما نشان می‌دهد، یعنی به ما می‌گوید که چند درصد از فایل دانلود شده و سرعت دانلود ما چقدر است:

```
[root@localhost ~]# wget http://ipv4.download.thinkbroadband.com/5MB.zip
--2020-04-15 10:39:48--  http://ipv4.download.thinkbroadband.com/5MB.zip
Resolving ipv4.download.thinkbroadband.com (ipv4.download.thinkbroadband.com)... 80.249.99.148
Connecting to ipv4.download.thinkbroadband.com (ipv4.download.thinkbroadband.com)|80.249.99.148|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 5242880 (5.0M) [application/zip]
Saving to: '5MB.zip'

22% [=====================>                                  ] 1,159,097    340KB/s  eta 13s
```

تصویر ۲۸- wget در خروجی، اطلاعاتی در مورد چگونگی دانلود به ما می‌دهد.

نصب نرم‌افزار از Source Code

در بخش‌های قبل به صورت کامل در مورد چگونگی نصب نرم‌افزار با استفاده از پکیج‌های آماده صحبت کردیم. همانطور که گفتیم، خیلی از اوقات پکیج‌های آماده، نرم‌افزارهای به‌روزی را در اختیار ما قرار نمی‌دهند. البته به دلیل این امر نیز اشاره کردیم؛ اما برخی از اوقات مجبوریم به دلایلی نظیر احتیاج به یک ویژگی جدید، به سراغ نصب آخرین نسخه‌ی نرم‌افزاری خاص برویم. در چنین شرایطی که یک پکیج آماده از آخرین نسخه‌ی نرم‌افزار وجود نداشته باشد، مجبوریم به سراغ نصب از سورس کد برویم.

فرض کنید ما می‌خواهیم آخرین نسخه نرم‌افزار squid را روی سیستم نصب کنیم. اگر به سایت squid بروید، می‌بینید که آخرین نسخه‌ی این نرم‌افزار، نسخه‌ی ٤,١٠ می‌باشد. با رفتن به صفحه‌ی دانلود این نسخه، می‌توانیم سورس کد این نرم‌افزار را دانلود کرده و آن را روی سیستم نصب کنیم. پس بیایید دقیقا همین کار را انجام دهیم. در بخش قبل با نرم‌افزار wget آشنا شدیم، پس سورس کد نرم‌افزار را درون خود لینوکس دانلود می‌کنیم:

```
[root@localhost ~]# wget http://www.squid-cache.org/Versions/v4/squid-4.10.tar.gz
...
2020-04-15 10:53:47 (418 KB/s) - 'squid-4.10.tar.gz' saved [5256312/5256312]
```

همانطور که می‌بینید، فایل سورس کد نرم‌افزار squid روی سیستم دانلود شد. در حال حاضر فایل‌های source code این نرم‌افزار درون یک فایل tar.gz قرار دارد. این که مفهوم tar و مفهوم gz چیست را در جلسات بعد به صورت کامل خواهیم دید، اما اگر بخواهیم آنها را به صورت ابتدایی توضیح دهیم، می‌گوییم که tar، ابزاری است که از طریق آن می‌توانیم چندین فایل را درون یک فایل قرار دهیم، یا به عبارتی، آرشیو کنیم و gz ابزاری است که با آن فایل‌ها را فشرده‌سازی یا compress می‌کنیم. برای درک بهتر، می‌توان گفت که ترکیب tar و gz، عملکردی مانند فایل‌های zip در ویندوز دارد.

حال که فایل سورس کد نرم‌افزار squid را دانلود کردیم، باید این فایل را از حالت آرشیو و فشرده شده درآوریم. برای این کار، از دستور زیر استفاده می‌کنیم:

```
[root@localhost ~]# tar xfvz squid-4.10.tar.gz
...
squid-4.10/SPONSORS
squid-4.10/CONTRIBUTORS
squid-4.10/RELEASENOTES.html
squid-4.10/COPYING
```

همانطور که می‌بینید، به محض وارد کردن این دستور، سیستم کلیه‌ی فایل‌ها و فولدرهای درون فایل tar.gz را extract می‌کند. همانطور که گفتیم، بعدا با دستور tar آشنا می‌شویم، اما با استفاده از آپشن x، به tar

می‌گوییم که عمل extract را انجام دهد، با آپشن f به tar می‌گوییم که می‌خواهیم یک فایل را به ورودی این دستور بدهیم، با آپشن v به tar می‌گوییم که به صورت verbose عمل کند (یعنی عملیاتی که انجام می‌دهد را در شل به ما نشان دهد) و در نهایت با آپشن z، به tar می‌گوییم که می‌خواهیم فایلی که با فرمت gz فشرده‌سازی شده است را decompress کند.

حال بیایید نگاهی به فولدرهای موجود بیاندازیم:

```
[root@localhost ~]# ls -l
total 5144
-rw-------.  1 root root    1257 Mar 20 10:48 anaconda-ks.cfg
drwxr-xr-x. 16 1000 1000    4096 Jan 20 06:37 squid-4.10
-rw-r--r--.  1 root root 5256312 Jan 20 08:25 squid-4.10.tar.gz
```

همانطور که می‌بینید، یک فولدر جدید به اسم squid-4.10 برای ما به وجود آمده که اگر به محتویات آن نگاهی بیاندازیم، می‌بینیم که سورس کد نرم‌افزار squid که همین الان آن را اکسترکت کردیم، می‌باشد:

```
[root@localhost ~]# cd squid-4.10
[root@localhost squid-4.10]# ls -l
...
-rw-r--r--.  1 1000 1000    3818 Jan 20 06:21 QUICKSTART
-rw-r--r--.  1 1000 1000    1513 Jan 20 06:21 README
-rw-r--r--.  1 1000 1000   30995 Jan 20 06:37 RELEASENOTES.html
drwxr-xr-x.  2 1000 1000    4096 Jan 20 06:22 scripts
-rw-r--r--.  1 1000 1000    4659 Jan 20 06:21 SPONSORS
drwxr-xr-x. 34 1000 1000   12288 Jan 20 06:37 src
drwxr-xr-x.  3 1000 1000    4096 Jan 20 06:37 test-suite
drwxr-xr-x.  8 1000 1000    4096 Jan 20 06:37 tools
```

اگر وارد فولدر src شوید، می‌توانید کلیه‌ی کدهای squid که اکثرا با زبان c نوشته شده‌اند را ببینید و تا دلتان می‌خواهد آنها را تغییر دهید. اما ما کاری با محتویات سورس کد نداریم و فقط می‌خواهیم آن را نصب کنیم.

برای این که یک نرم‌افزار را از طریق سورس کد نصب کنیم، اول باید محیط سیستم را برای نصب آن نرم‌افزار آماده کنیم. در فولدر اکثر نرم‌افزارهایی که به صورت سورس کد دانلود می‌کنیم، یک اسکریپت به نام configure وجود دارد. کاری که این اسکریپت انجام می‌دهد، آماده‌سازی سیستم برای کامپایل و نصب آن نرم‌افزار می‌باشد. مثلا این اسکریپت چک می‌کند که آیا dependencyهای مورد نیاز برنامه در سیستم نصب هست یا نه، کامپایلر c روی سیستم نصب هست یا نه و... .

خوب، بیایید این اسکریپت را اجرا کنیم. برای اجرای این اسکریپت، باید نام این اسکریپت را به صورت زیر وارد کنیم:

```
[root@localhost squid-4.10]# ./configure
```

شاید این امر کمی برایتان عجیب باشد، اما اگر ما نام configure را بدون ./ بنویسیم، bash فکر می‌کند که configure یک دستور است و به دنبال اجرای آن دستور می‌رود. اما ما با قرار دادن /.، به bash می‌گوییم که فایل configure موجود در همین فولدر را اجرا کند. علامت نقطه (.) نشان‌دهنده‌ی فولدر کنونی می‌باشد و با استفاده از / و نوشتن نام فایل، می‌گوییم که چه فایلی از فولدر کنونی باید اجرا شود. خوب وقتی این دستور را وارد کنیم، با پیغامی که در صفحه‌ی بعد می‌بینید مواجه می‌شویم:

```
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for a thread-safe mkdir -p... /usr/bin/mkdir -p
checking for gawk... gawk
checking whether make sets $(MAKE)... yes
checking whether make supports nested variables... yes
checking whether UID '0' is supported by ustar format... yes
checking whether GID '0' is supported by ustar format... yes
checking how to create a ustar tar archive... gnutar
checking whether to enable maintainer-specific portions of Makefiles... no
checking for gcc... no
checking for cc... no
checking for cl.exe... no
configure: error: in `/root/squid-4.10':
configure: error: no acceptable C compiler found in $PATH
See `config.log' for more details
```

همانطور که می‌بینید اسکریپت configure به ما گفته که روی سیستم ما، کامپایلر c را پیدا نکرده است. پس باید به سراغ نصب کامپایلر c برویم. با کمی جستجو، می‌بینیم که نسخه‌ی ٤ نرم‌افزار squid، احتیاج به کامپایلر c++ و همچنین پکیج binutils نیز دارد. ما در سیستم‌های لینوکسی، از کامپایلرهای gcc یا GNU Compiler Collection که کلکسیونی از کامپایلرهای C، C++ و برخی زبان‌های دیگر می‌باشند استفاده می‌کنیم. این کامپایلرها، هم نرم‌افزار آزاد و هم مجانی می‌باشند. برای نصب این کامپایلرها، از yum کمک می‌گیریم:

```
[root@localhost squid-4.10]# yum install gcc gcc-c++ binutils
```

نکته: همانطور که می‌بینید با نوشتن نام چند نرم‌افزار پشت سر هم، می‌توانیم با استفاده از yum چندین نرم‌افزار را با یک خط دستور، نصب کنیم.

پس از اتمام نصب این کامپایلرها، بار دیگر به سراغ اجرای اسکریپت configure می‌رویم:

```
[root@localhost squid-4.10]# ./configure
```

همانطور که می‌بینید، این بار پس از اجرای این اسکریپت، خطایی به ما نشان داده نمی‌شود و اسکریپت شروع به آماده‌سازی محیط سیستم برای نصب نرم‌افزار می‌کند. شاید برایتان سوال باشد که configure دقیقا چه فولدری از سیستم را آماده‌ی نصب squid می‌کند. به صورت پیش‌فرض، squid خود را در فولدر /usr/local/squid نصب می‌کند. اگر بخواهید موقعیت نصب را تغییر دهید، می‌توانید با اضافه کردن آپشن --prefix= و نوشتن موقعیت جدید نصب، برنامه را در جایی دیگر نصب کنید. اما ما با تنظیمات پیش‌فرض جلو می‌رویم.

پس از این که اسکریپت کار خود را تمام کرد، می‌توانیم به سراغ کامپایل (یا اگر بخواهیم دقیق‌تر بگوییم، build) squid برویم. برای این کار، از دستور make all استفاده می‌کنیم:

```
[root@localhost squid-4.10]# make all
```

پس از وارد کردن این دستور، باید مدت زمان زیادی منتظر کامپایل شدن برنامه بمانیم. تا برنامه کامپایل می‌شود، بیایید کمی در مورد دستور make all صحبت کنیم. وقتی که اسکریپت configure به درستی اجرا شود و کار خود را انجام دهد، برای ما یک فایل جدید به نام Makefile به وجود می‌آورد. فایل Makefile، فایلی است که در آن یک سری task برای کامپایل کردن سورس کد نوشته شده است. این فایل، از روی یک فایل دیگر، به نام Makefile.in ایجاد می‌شود.

پس از این که عملیات کامپایل پایان یافت، باید با استفاده از دستور make install، فایل‌های کامپایل شده را نصب کنیم:

```
[root@localhost squid-4.10]# make install
```

کاری که این دستور انجام می‌دهد، کپی کردن کلیه‌ی باینری‌های ایجاد شده (یا همان فایل‌های کامپایل شده) توسط make all در موقعیت‌های مربوطه در هارد دیسک سیستم می‌باشد. در واقع این دستور فایل‌های اجرایی، مستندات، لایبرری‌ها و... برنامه را در موقعیت‌های مربوطه، کپی می‌کند.

پس از اتمام کار دستور make install، می‌توانیم به محل نصب squid برویم و نگاهی به محتویات آن بیاندازیم:

```
[root@localhost share]# cd /usr/local/squid/
[root@localhost squid]# ls -l
total 4
drwxr-xr-x. 2 root root   38 Apr 16 13:48 bin
drwxr-xr-x. 2 root root  221 Apr 16 13:51 etc
drwxr-xr-x. 2 root root 4096 Apr 16 13:48 libexec
drwxr-xr-x. 2 root root   19 Apr 16 13:48 sbin
drwxr-xr-x. 5 root root   59 Apr 16 13:48 share
drwxr-xr-x. 5 root root   42 Apr 16 13:48 var
```

در هر کدام از این فولدرها، یک سری فایل مربوط به راه‌اندازی squid وجود دارد. مثلا در فولدر share، فایل manpage این برنامه وجود دارد، یا مثلا در فولدر sbin، فایل باینری اجرایی squid وجود دارد.

بیایید سعی کنیم squid را با استفاده از فایل باینری آن اجرا کنیم:

```
[root@localhost squid]# cd sbin/
[root@localhost sbin]# ls
squid
[root@localhost sbin]# ./squid
WARNING: Cannot write log file: /usr/local/squid/var/logs/cache.log
/usr/local/squid/var/logs/cache.log: Permission denied
        messages will be sent to 'stderr'.
```

همانطور که می‌بینید، فایل باینری squid، به ما یک warning می‌دهد، چرا که این فایل اجازه‌ی نوشتن logهای خود در فولدر ذکر شده را ندارد و به همین دلیل، logها را درون STDERR قرار می‌دهد. این یکی از مسائلی است که هنگام نصب برنامه از روی سورس کد باید به آن توجه کنیم. یعنی پس از نصب برنامه، باید permissionهای برنامه را نیز به درستی تنظیم کنیم. البته این مسئله‌ای نیست که در حال حاضر بخواهیم به آن بپردازیم.

نکته‌ای که باید به آن توجه کنید این است که ما هر برنامه‌ای را که از طریق سورس کد نصب کنیم، توسط rpm و yum قابل مشاهده یا حذف کردن نخواهند بود. مثلا:

```
[root@localhost squid]# rpm -q squid
squid-3.5.20-12.el7_6.1.x86_64
```

همانطور که می‌بینید، rpm به ما نسخه‌ی ۳,۵ نرم‌افزار squid را نشان می‌دهد، در حالی که ما نسخه‌ی ٤,١٠ را نیز نصب کرده‌ایم. دلیل این است که ما squid را با استفاده از یک پکیج rpm نصب نکردیم، پس هیچ راهی وجود ندارد که rpm بتواند آن را ببیند و از وجود آن اطلاع داشته باشد.

نرم‌افزار yum نیز دقیقا همین رفتار را خواهد داشت. مثلا:

```
[root@localhost ~]# yum remove squid
...
Removed:
  squid.x86_64 7:3.5.20-12.el7_6.1

Complete!
```

همانطور که می‌بینید ما با استفاده از این دستور، نرم‌افزار squid را پاک کردیم، اما اگر به فولدر `/usr/local/squid` برویم، می‌بینیم که نسخه‌ای که خودمان کامپایل کردیم هنوز سرجای خود وجود دارد؛ چرا که yum نسخه‌ی ۳٫۵ نرم‌افزار squid را پاک کرده است.

نصب کردن نرم‌افزار از طریق source code مشکلات دیگری نیز برای ما ایجاد می‌کند. برای مثال، اگر در جایی از سیستم سعی کنید دستور `squid` را اجرا کنید، می‌بینید که bash به شما ارور می‌دهد:

```
[root@localhost ~]# squid
-bash: /usr/sbin/squid: No such file or directory
```

دلیل این است که موقعیتی که در آن باینری squid قرار گرفته، درون متغیر $PATH شل وجود ندارد. علاوه بر این، حتی manpage دستور squid را نیز نمی‌توانید مشاهده کنید، چون manpage آن در فولدر استاندارد مربوط به manpageها قرار نگرفته است.

این فقط بخشی از مشکلاتی بود که با نصب کردن نرم‌افزار از روی سورس کد به وجود می‌آید. البته کلیه‌ی این مشکلات را می‌توان حل کرد، ولی این کار زمان‌بر خواهد بود. پس همانطور که می‌بینید، بهتر است تا زمانی که مجبور نشده‌ایم، به سراغ نصب برنامه از سورس کد نرویم و از ابزارهایی مثل yum و rpm برای نصب برنامه‌ها استفاده کنیم، چون آنها کار ما را بسیار راحت می‌کنند.

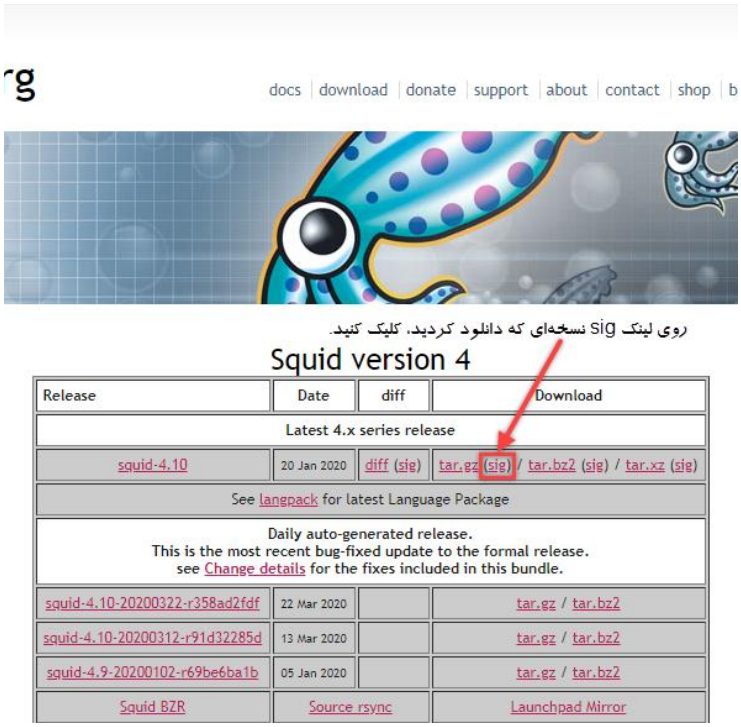
نکته: همانطور که گفتیم، rpm و yum پکیج‌منیجرهای توزیع‌های Red Hat-based می‌باشند. در سیستم‌هایی که از SUSE یا openSUSE استفاده می‌کنند، دستور rpm موجود می‌باشد و دقیقا مانند rpm در سیستم‌های Red Hat-based کار می‌کند. اما در آن، پکیج منیجر zypper جایگزین yum می‌باشد. در سیستم‌های Debian-based، نرم‌افزار dpkg معادل rpm، و نرم‌افزارهای apt-cache و apt-get معادل نرم‌افزار yum می‌باشند. این نرم‌افزارها بسیار شبیه هم می‌باشند اما برخی از آپشن‌های آنها با هم متفاوت است. البته ما فعلا به پکیج‌منیجرهای سایر توزیع‌ها نمی‌پردازیم.

بررسی Checksum برنامه‌ی دانلود شده

ما در بخش قبل یک سری کد را دانلود کردیم، آنها را کامپایل کردیم و روی سیستم نصب کردیم. معمولا برای دانلود سورس کد هر برنامه، به وبسایت اصلی آن برنامه وصل می‌شویم. خیلی از اوقات آن وبسایت‌ها ما را به یک سری mirror ارجاع می‌دهند. سوال این است که ما از کجا می‌توانیم مطئن باشیم که کسی یکی از mirrorها را hijack نکرده و مثلا یک قطعه کد به سورس کد برنامه اضافه نکرده باشد؟ آن قطعه کد می‌تواند یک backdoor باشد که از طریق آن می‌توانند سیستم شما را کنترل کنند. اینجاست که باید در مورد روش‌های بررسی یکپارچگی فایل صحبت کنیم.

ما می‌توانیم با استفاده از الگوریتم‌های Hash، نظیر SHA (Secure Hash Algorithm)، یکپارچگی یا integrity یک فایل دانلود شده را بررسی کنیم. فرض کنید می‌خواهیم یکپارچگی فایل سورس کد برنامه‌ی squid را که در چند بخش قبل دانلود کردیم، بررسی کنیم. اگر به بخش دانلود وبسایت squid برویم، لینکی به نام sig

می‌بینیم:



تصویر ۲۹- لینک مشاهده‌ی اطلاعات Checksum برنامه‌ی squid

با کلیک بر روی این لینک، یک فایل asc. روی سیستم شما دانلود می‌شود. روش‌های زیادی برای بررسی Checksum بسته با توجه به فایل asc. وجود دارد، اما ما ساده‌ترین روش را یاد می‌گیریم. برای مشاهده‌ی محتویات این فایل، کافی است فایل asc. دانلود شده را توسط مرورگر خود باز کنید:

تصویر ۳۰- مشاهده‌ی محتویات فایل asc.

همانطور که می‌بینید اطلاعات متفاوتی در مورد پکیج squid-4.10.tar.gz در این فایل وجود دارد. ما به دنبال مقدار SHA1 این پکیج هستیم. حال که این مقدار را داریم، به سیستم لینوکسی خود می‌رویم و با استفاده از دستور sha1sum، مقدار SHA1 فایلی که روی لینوکس دانلود کردیم را بررسی می‌کنیم:

```
[root@localhost ~]# sha1sum squid-4.10.tar.gz
e55a6dd5a1fe2f6867cba122fa8bbc17550f9e11  squid-4.10.tar.gz
```

حال باید این مقدار را با مقدار SHA1 موجود در فایل asc. مقایسه کنیم. همانطور که می‌بینید این مقادیر با هم برابر هستند، پس پکیج ما سالم می‌باشد.

تحقیق: لایسنس GPL چیست؟

لایسنس GPL، یک لایسنس برای نرم‌افزارهای آزاد یا Free Software می‌باشد. این لایسنس به کاربر اجازه می‌دهد که از نرم‌افزار استفاده کند، آن را تغییر دهد و همچنین آن را کپی کند. به علاوه، کاربران می‌توانند نسخه‌هایی از برنامه را با اعمال تغییرات یا بدون اعمال تغییرات بفروشند یا اهدا کنند.

البته فروش و اهدای نرم‌افزار ملزم به قبول دو شرط زیر می‌باشد:

- همراه نرم‌افزار، حتما باید یک کپی از سورس کد برنامه، یا اطلاعاتی در مورد چگونگی دسترسی به سورس کد برنامه وجود داشته باشد.
- لایسنس نرم‌افزار نمی‌تواند حذف شود یا تغییر یابد. به عبارت دیگر لایسنس نرم‌افزار باید همیشه GPL باشد.

اگر کاربر این دو شرط را قبول نکند، همچنان می‌تواند نرم‌افزار را برای خود تغییر دهد و از آن استفاده کند، اما نمی‌تواند آن را بفروشد یا اهدا کند.

GPL چندین نسخه‌ی متفاوت دارد؛ معروف‌ترین آنها GPLv2 و GPLv3 می‌باشند. کرنل لینوکس، از GPLv2 استفاده می‌کند.

(اطلاعات بیشتر در مورد GPLv2، GPLv3)

تحقیق: FHS چیست؟

خیلی از کاربران لینوکس، بخش زیادی از وقت خود را در سیستم‌های ویندوزی گذرانده‌اند و به همین دلیل، فولدرهای مهم و ساختارهای آن در لینوکس برای آنها ناآشنا می‌باشد. مثلا ما می‌دانیم که به صورت پیش‌فرض، ویندوز در درایو C ریخته می‌شود و فولدر C:\Windows\System32 لایبرری‌ها و فایل‌های اجرایی مهم ویندوز را درون خود دارد. اما در لینوکس چه؟

ما در لینوکس مفهومی به نام درایو C، D و... یا به عبارت دیگر، Driver Letter نداریم. به جای همه‌ی اینها، یک /، و چندین دایرکتوری (فولدر) با نام‌های عجیب داریم. FHS یا Filesystem Hierarchy Structure، ساختار فایل‌سیستم سیستم‌های لینوکسی را تعریف می‌کند. در واقع، FHS قصد دارد چگونگی سازماندهی فولدرها در لینوکس و این که چه فایل‌هایی باید در چه فولدرهایی قرار گیرند را استانداردسازی کند. اکثر توزیع‌های لینوکسی ازساختارهای ذکر شده در FHS پیروی می‌کنند؛ اما بعضا ممکن است کمی از آن طفره بروند.

ساختار دایرکتوری‌های فایل‌سیستم در FHS، به شرح زیر می‌باشد:

- /

همه چیز در سیستم‌های لینوکس، در دایرکتوری /، که به آن دایرکتوری روت می‌گویند، قرار دارد. می‌توانید / را مانند درایو C:\ در ویندوز ببینید؛ البته همانطور که گفتیم در لینوکس مفهومی به نام Driver Letter وجود ندارد. میدانیم که در سیستم‌های ویندوزی، پارتیشن‌های متفاوت از هارد شما با Driver Letterهای متفاوت شناخته می‌شوند، اما در لینوکس، پارتیشن‌ها به عنوان یک دایرکتوری درون / شناخته می‌شود.

- **/bin و /sbin**

در این دو دایرکتوری، فایل‌های باینری (فایل‌های اجرایی، یا نرم‌افزارها) سیستم قرار می‌گیرند. در هر دوی این دایرکتوری‌ها، فایل‌های اجرایی لازم برای بوت کردن سیستم (مثل دستور mount) قرار می‌گیرد.

تفاوت اصلی بین /sbin و /bin در این است که /sbin، فایل‌های اجرایی ضروری سیستم را درون خود دارد.

- **/boot**

در این دارکتوری، فایل‌های لازم برای بوت کردن سیستم وجود دارد. مثلا فایل‌های بوت‌لودر و فایل‌های کرنل لینوکس در این دایرکتوری قرار دارند. معمولا این دایرکتوری در یک پارتیشن مجزا در سیستم شما قرار می‌گیرد.

- **/etc**

در این دایرکتوری، کلیه‌ی فایل‌های لازم برای پیکربندی (configuration) نرم‌افزارهای متفاوت و برنامه‌های سیستمی وجود دارد. اگر بخواهیم یک سرویس نظیر squid را پیکربندی کنیم یا تنظیمات شبکه‌ی سیستم را عوض کنیم، باید به سراغ فایل‌های موجود در این دایرکتوری بیاییم.

- **/home**

دایرکتوری /home، مکانی است که به هر کدام از کاربران سیستم، یک دایرکتوری جهت ذخیره‌ی فایل‌های شخصی اختصاص داده می‌شود. دایرکتوری‌های موجود در /home، همنام username کاربر می‌باشند. مثلا اگر در سیستم کاربری به نام Behnam وجود داشته باشد، این کاربر یک دایرکتوری اختصاصی به آدرس /home/Behnam خواهد داشت. هر دایرکتوری درون /home، می‌تواند تنظیمات شخصی کاربر، کلیدهای SSH کاربر و... درون خود داشته باشد.

- **/lib**

این دایرکتوری کلیه‌ی لایبرری‌های مورد نیاز برای عملکرد صحیح فایل‌های اجرایی موجود در /bin و /sbin را درون خود دارد. ماژول‌های کرنل نیز در این دایرکتوری قرار دارند.

- **/usr، /usr/bin، /usr/lib و /usr/sbin**

دایرکتوری‌های /usr، برای ذخیره‌ی فایل‌هایی که وجود آنها برای بوت‌کردن سیستم واجب نیست، استفاده می‌شود. به عبارت دیگر، هر وقت یک نرم‌افزار جانبی نظیر squid را روی سیستم خود نصب کنید، فایل‌های اجرایی و لایبرری‌های آن در /usr/bin، /usr/sbin و /usr/lib قرار خواهد گرفت.

- **/usr/local**

این دایرکتوری یک نسخه‌ی ویژه از /usr می‌باشد که درون خود دایرکتوری‌های bin، sbin و lib را دارد. این دایرکتوری مکانی است که به کاربران اجازه می‌دهد نرم‌افزارهای دلخواه خود را که از ریپازیتوری‌های خود توزیع نیامده، بدون نگرانی در مورد دستکاری کردن فایل‌های اصلی توزیع، نصب کنند.

- **/opt**

دایرکتوری /opt و /usr/local عملکردی شبیه به هم دارند. بر خلاف /usr/local که در آن دایرکتوری‌های bin، sbin و lib بین هر نرم‌افزاری که نصب می‌کنیم به اشتراک گذاشته شود، در /opt، هر نرم‌افزار در یک دایرکتوری جداگانه قرار می‌گیرد و درون آن دایرکتوری، دایرکتوری‌های /bin، /sbin و /lib وجود خواهد داشت.

یعنی اگر squid را در /usr/local نصب کنیم، فایل‌های باینری و لایبرری آن در /usr/local/bin، /usr/local/sbin و /usr/local/lib قرار می‌گیرد. اما اگر squid را در /opt نصب کنیم، فایل‌های باینری و لایبرری آن در /opt/squid/bin، /opt/squid/sbin و /opt/squid/lib قرار می‌گیرد.

البته نرم‌افزارهایی که در /opt نصب می‌کنیم مجبورنیستند دایرکتوری‌های bin و... را درون خود داشته باشند و به همین دلیل، این دایرکتوری معمولا توسط نرم‌افزارهایی که از ساختار FHS پیروی نمی‌کنند (مثل نرم‌افزارهای غیر آزاد یا غیر اُپن سورس) استفاده می‌شود. منطق پشت ساختار /opt این است که اگر کاربر بخواهد نرم‌افزاری را پاک کند، می‌تواند به سادگی دایرکتوری آن نرم‌افزار را در /opt پاک کند، بدون این که نگران پاک شدن فایل‌های مربوط به سایر نرم‌افزارها باشد.

- **/root**

این دایرکتوری، دایرکتوری مخصوص کاربر root می‌باشد که فقط کاربر root به آن دسترسی دارد. به عبارتی دیگر، این فولدر، فولدر home کاربر root می‌باشد.

- **/var**

این دایرکتوری، مخصوص ذخیره‌ی فایل‌هایی هست که حجم متغیری دارند. بیشتر از این دایرکتوری برای ذخیره‌ی logهایی که توسط سرویس‌های متفاوت سیستم ایجاد می‌شود، استفاده می‌شود.

- **/dev**

سیستم‌های لینوکسی همه چیز را به عنوان یک فایل می‌بینند؛ یعنی حتی سخت افزارهای سیستم نیز به عنوان یک فایل در نظر گرفته می‌شوند. در دایرکتوری /dev، یک سری فایل ویژه وجود دارد هر کدام، نمایان کننده‌ی یک سخت‌افزار می‌باشند. این فایل‌ها، فایل‌های همیشگی که به آنها عادت داریم نیستند، اما نمای یک فایل را دارند. مثلا فایل /dev/sda نمایان کننده‌ی اولین درایو SATA در سیستم می‌باشد. اگر بخواهیم آن درایو را پاریتیشن کنیم، باید به نرم‌افزار پارتیشن‌بندی بگوییم که /dev/sda را برایمان پارتیشن‌بندی کند.

علاوه بر سخت‌افزارها، این دایرکتوری یک سری شبه-سخت‌افزار را نیز درون خود دارد که نمایان کننده‌ی سخت‌افزارهای واقعی نمی‌باشند. مثلا /dev/random اعداد تصادفی ایجاد می‌کند و /dev/null فقط یک ورودی دریافت می‌کنند و آن را دور می‌ریزد. مثلا اگر خروجی یک دستور را درون /dev/null پایپ کنیم، آن خروجی کاملا دور ریخته می‌شود.

- **/proc و /sys**

دایرکتوری /proc و /sys نیز دارای شبه‌فایل‌هایی هستند که نمایانگر چیزی فراتر از یک فایل می‌باشند (دقیقا مثل فایل‌های موجود در /dev).

دایرکتوری proc/، فایل‌هایی را که نمایانگر اطلاعات در مورد همه‌ی پراسس‌های روی سیستم می‌باشد را درون خود نگه‌داری می‌کنند و دایرکتوری sys/ شامل کلیه‌ی فایل‌هایی که به شما اجازه‌ی تعامل با کرنل را می‌دهند، می‌باشد. با این حال، برخی از فایل‌های مربوط به کرنل درون proc/sys ذخیره می‌شوند.

- **srv/**

این دایرکتوری برای ذخیره‌سازی فایل‌هایی که یک سرور با بیرون به اشتراک می‌گذارد، می‌باشد. مثلا فایل‌های یک وب سرور، درون srv/www قرار می‌گیرند، چون فایل‌های مربوط به یک وبسایت باید با بیرون به اشتراک گذاشته شود.

- **mnt/ و media/**

اگر به سیستم خود یک دیسک USB (فلش مموری، هارد اکسترنال و...) یا یک دیسک NFS، یا به عبارت دیگر، هر چیزی که دارای یک فایل‌سیستم باشد متصل کنید، این فایل‌سیستم‌ها در mnt/ و media/ قابل مشاهده خواهند بود.

قبلا دایرکتوری mnt/ برای هر دیسکی که به سیستم متصل می‌کردیم استفاده می‌شد، اما امروزه باید از mnt/ برای درایوهای NFS و سایر دیسک‌هایی که بلند مدت به سیستم متصل خواهند بود و از media/ برای فلش مموری، سی‌دی و... استفاده کنیم.

- **tmp/، var/tmp و dev/shm**

دایرکتوری tmp/، برای ذخیره‌ی فایل‌های موقتی که پس از ریبوت سیستم به آنها احتیاج نداریم، استفاده می‌شود. لینوکس هنگام بوت‌شدن، کلیه‌ی محتویات tmp/ را پاک می‌کند.

محتویات دایرکتوری var/tmp/، پس از ریبوت پاک نمی‌شوند، پس این دایرکتوری به درد ذخیره‌ی فایل‌های cache که به آنها احتیاج داریم می‌خورد.

دایرکتوری dev/shm/ یک RAM کوچک می‌باشد و هر فایلی که در این قسمت ذخیره شود، پس از خاموش شدن سیستم پاک می‌شود. این دایرکتوری، بهترین بخش سیستم برای نگهداری فایل‌های محرمانه و مهم نظیر پسوردها می‌باشد، چون این اطلاعات هیچ وقت روی هارد دیسک ریخته نمی‌شوند، بلکه روی RAM قرار می‌گیرند و پس از خاموش شدن سیستم از بین می‌روند. البته باید حواستان به permission فایل‌هایی که در این دایرکتوری قرار می‌دهید باشد، چون اگر فایل‌های موجود در این دایرکتوری را همه بتوانند بخوانند، خاک بر سرتان خواهد شد.