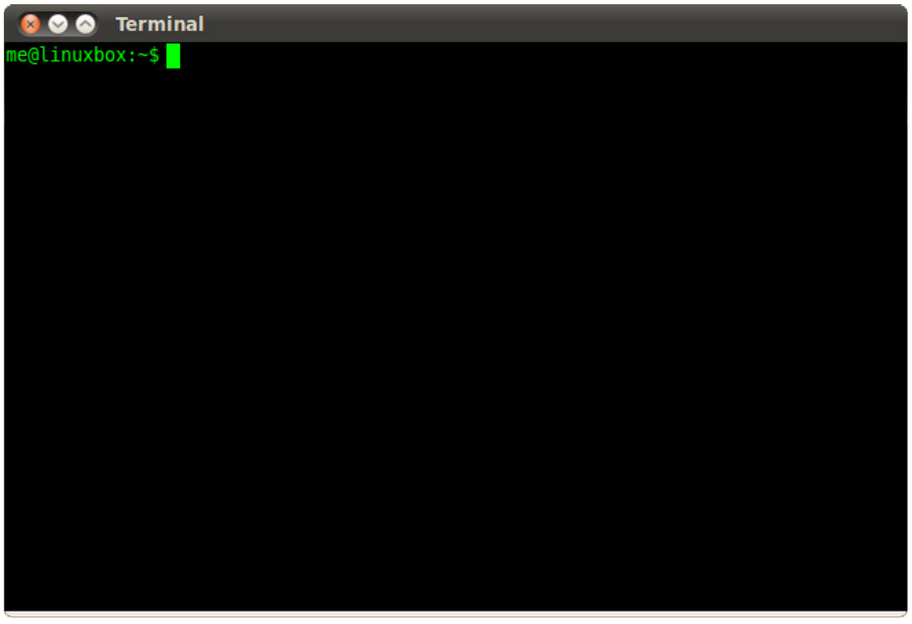
LinuxCommand.org



این پیوندها به سایت اصلی مرتبط هستند.

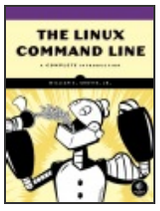
- Home
- Learning The Shell
- Writing Shell Scripts
- Resources
- The Book

- Blog

اکنون چطور؟

شما لینوکس را نصب کرده‌اید و در حال اجرا می‌باشد. رابط گرافیکی کاربر به خوبی کار می‌کند، لیکن تعویض تِم‌های میزکار برای شما خالی از لطف گردیده، و مرتباً با این شئ «ترمینال» هم مواجه می‌شوید.

نگران نباشید، ما به شما نشان خواهیم داد که چه کار بکنید .



مشابه این سایت؟ این کتاب را بخوانید!

The Linux Command Line توسط William Shotts

Learning the Shell

آموزش پوسته

زحمت برای چه؟

چرا لازم است شما در هر صورت خط فرمان را بیاموزید؟ خُب، اجازه دهید ماجرایی را برایتان نقل کنم. چند سال قبل در جایی که کار می‌کردم مشکلی داشتیم. یک درایو به اشتراک گذاشته شده روی یکی از سرویس‌دهنده‌های فایل ما وجود داشت که مرتب پُر می‌شد. از این صحبت نمی‌کنم که این سیستم‌عامل منسوخ از سهمیه‌های کاربر پشتیبانی نمی‌کرد، که آن نیز داستان دیگری است. اما فضای دیسک سرویس‌دهنده پُر می‌شد و مانع کار کردن افراد می‌گردید. یکی از مهندسین نرم‌افزار ما بخش نسبتاً زیادی از روز را صرف نوشتن یک برنامه C++ نمود که تمام دایرکتوریهای کاربر را به دقت بررسی می‌کرد و فضایی را که آنها مصرف می‌کردند جمع کرده و لیستی از نتایج تهیه می‌نمود. چون من هنگامی که در آن شغل بودم، مجبور شده بودم از آن سیستم‌عامل مهجور استفاده کنم، یک محیط خط فرمان شِبه لینوکس برای آن را نصب کردم. موقعی که در باره مشکل شنیدم، نشان دادم تمام کاری را که این مهندس انجام داده بود، من می‌توانستم با این سطر فرمان واحد انجام بدهم [1]:

```
du -s * | sort -nr > $HOME/user_space_report.txt
```

رابط‌های گرافیکی کاربر (GUIها) برای بسیاری از وظایف سودمند هستند، اما برای تمام وظایف مناسب نیستند. من مدتی است احساس کرده‌ام که امروز اکثر کامپیوترها توسط الکتریسیته به حرکت در نمی‌آیند. در عوض به نظر می‌رسد آنها بوسیله حرکت فریبنده ماوس نیرو می‌گیرند! کامپیوترها برای رهایی ما از کار دستی در نظر گرفته شده بودند، اما چند بار مقداری از وظیفه‌ای را انجام داده‌اید که یقین داشتید کامپیوتر باید قادر به انجام آن باشد لیکن خودتان آن را به طور کسل کننده با کاربرد ماوس به پایان رسانده‌اید؟ اشاره کردن و کلیک کردن، اشاره کردن و کلیک کردن.

سابقاً از یک نویسنده شنیدم که می‌گوید وقتی شما بچه هستید یک کامپیوتر را توسط نگاه کردن به عکس‌ها به کار می‌برید. موقعی که بالغ می‌شوید شما یاد می‌گیرید که بخوانید و بنویسید. Welcome to Computer Literacy 101. اکنون بیایید به کار بپردازیم.

فهرست مطالب

1. **مترجم:** در تایید این مطلب، از رادیو گیک شماره ۳۲ جادی می‌شنویم که اریک ریموند(Eric Steven Raymond) از پایه‌گذاران و توسعه دهندگان مبانی فکری و فلسفیopen source که بیشتر به عنوان سخنگوی اجتماع توسعه دهندگان منبع باز شناخته می‌شود و نویسنده کتاب ارزشمند هنر برنامه‌نویسی لینوکس است، می‌گوید: **طبیعت یونیکسی که در یک خط شل نهفته است بیشتر از ده هزار خط کُد C است.** (شنیدن رادیو گیک مخصوصاً این شماره و اولین شماره آنرا به شما پیشنهاد می‌دهم).(برگشت)

«شل» چیست؟

به سادگی چنین تعبیر کنید، شل برنامه‌ای است که فرمانها را از صفحه کلید می‌گیرد و آنها را برای انجام دادن به سیستم عامل ارایه می‌کند. در گذشته تنها رابط کاربر معتبر در یک سیستم شبه‌ـیونیکس همچون لینوکس، شل بود. امروزه، ما علاوه بر ***رابط‌های خط فرمانی کاربر (CLIها)*** از قبیل شل، دارای ***رابط‌های گرافیکی کاربر (GUIها)*** هستیم.

در اکثر سیستم‌های لینوکس برنامه‌ای به نام **bash** (که اختصاری برای **B**ourne **A**gain **SH**ell است، یک نگارش تقویت یافته از برنامه شل یونیکس، **sh**، نوشته شده توسط Steve Bourne) به عنوان برنامه شل عمل می‌کند. در کنار **bash**، برنامه‌های شل دیگری وجود دارند که می‌توانند در یک سیستم لینوکس نصب باشند. اینها **ksh**، **tcsh** و **zsh** را شامل می‌شوند.

یک «Terminal» چیست؟

برنامه‌ای است که ***شبیه‌ساز ترمینال*** نامیده می‌شود. این یک برنامه است که پنجره‌ای را باز می‌کند و به شما اجازه می‌دهد که با شل عمل متقابل داشته باشید. یک گروه مختلف از شبیه‌سازهای ترمینال وجود دارند که شما می‌توانید به کار ببرید. اکثر توزیع‌های لینوکس تعدادی از آنها را فراهم می‌کنند، از قبیل: **gnome-terminal**، **konsole**، **xterm**، **rxvt**، **kvt**، **nxterm**، و **eterm**.

راه‌اندازی یک Terminal

احتمالاً مدیر پنجره شما دارای روشی برای راه انداختن یک ترمینال از منو می‌باشد. برای دیدن موردی که مانند شبیه‌ساز ترمینال به نظر برسد، به دقت لیست برنامه‌ها را نگاه کنید. اگر شما یک کاربر KDE هستید، برنامه ترمینال «konsole» نامیده می‌شود، در گنوم این برنامه «gnome-terminal» نامیده می‌شود. شما می‌توانید هر تعداد از اینها را که می‌خواهید راه‌اندازی نمایید و با آنها کار کنید. در حالیکه تعدادی شبیه‌ساز ترمینال مختلف وجود دارد، تمام آنها کار همانندی انجام می‌دهند. آنها دستیابی به یک نشست پوسته را به شما می‌دهند. احتمالاً شما بر اساس ویژگی‌های نمایش و گرافیک مختلفی که هر یک فراهم می‌کنند، اولویتی را برای یکی از آنها ایجاد خواهید نمود.

آزمایش صفحه کلید

خُب، بیاید مقداری تایپ کنیم. یک پنجره ترمینال را بیاورید. باید یک ***اعلان پوسته*** که شامل نام کاربری شما و نام ماشین که با یک علامت دلار دنبال می‌شود را ببینید. موردی مشابه این:

```
[me@linuxbox me]$
```

بسیار خوب! حالا چند کاراکتر بی‌معنی تایپ کنید و کلید اینتر را بزنید.

```
[me@linuxbox me]$ kdkjflajfks
```

اگر همه چیز درست باشد، شما باید پیغام خطای شکایت کننده‌ای بیانگر اینکه نمی‌تواند شما را بفهمد دریافت کرده باشید:

```
[me@linuxbox me]$ kdkjflajfks

bash: kdkjflajfks: command not found
```

شگفتا! اکنون کلید جهت نمای بالا (up-arrow) را بزنید. ببینید چطور فرمان «kdkjflajfks» قبلی ما را برمی‌گرداند. بلی، دارای ***تاریخچه فرمان*** هستیم. کلید جهت‌نمای پایین را فشار دهید و دوباره ما سطر خالی را به دست می‌آوریم.

اگر لازم بود فرمان «kdkjflajfks» را با استفاده از کلید جهت بالا فراخوانی کنید. اکنون، کلیدهای جهت چپ و راست را امتحان کنید. شما می‌توانید اشاره‌گر متن را هر جایی از سطر فرمان قرار بدهید. این مطلب به شما امکان می‌دهد اشتباهات را به آسانی اصلاح کنید.

شما به عنوان کاربر ارشد(root) وارد نشده‌اید، شده‌اید؟

اگر آخرین کاراکتر اعلان فرمان شما به جای $ کاراکتر # است، شما به عنوان ***کاربر ارشد*** عمل می‌کنید. این به معنای آن است که شما از مزایای مدیریتی برخوردار هستید. این امر به طور بالقوه می‌تواند خطرناک باشد، چون شما قادر به حذف کردن یا رونویسی هر فایلی درسیستم هستید. جز اینکه به طور یقین به مزایای مدیریتی نیاز داشته باشید، به عنوان کاربر ارشد عمل نکنید.

استفاده از ماوس

هر چند که پوسته یک رابط خط فرمان است، بازهم ماوس آماده است.

در کنار استفاده از موشواره برای مرور کردن محتویات پنجره ترمینال، شما می‌توانید با موشواره کپی متن انجام بدهید. ماوس خود را درحالیکه دکمه سمت چپ را پایین نگاه داشته‌اید روی مقداری متن (برای مثال، «kdkjflajfks» درست اینجا روی پنجره مرورگر) بکشید. متن باید نشان شده(highlight) بشود. دکمه سمت چپ را رها کنید و اشاره‌گر موشواره خود را به پنجره ترمینال برده و دکمه میانی ماوس را (و اگر در حال کار با صفحه لمسی--touch pad-- هستید، به طور جایگزین می‌توانید هر دو دکمه چپ و راست را به طور همزمان فشار بدهید). متنی که شما در مرورگر نشان دار کردید، باید به سطر فرمان کپی بشود.

چند کلمه در مورد focus...

موقعی که شما سیستم لینوکس خود و مدیر پنجره‌اش (به احتمال قوی Gnome یا KDE) را نصب کردید، سیستم شمابرای رفتار کردن مطابق برخی روشهای آن سیستم عامل متروک پیکربندی شده است.

مخصوصاً، شاید دارای ***رویه تمرکز*** تنظیم شده خودش جهت «کلیک برای تمرکز» باشد. این به معنای آن است که برای تمرکز به دست آوردن یک پنجره(فعال شدن آن) شما باید در پنجره کلیک کنید. این با رفتار سنتی پنجره X Window مغایر است. باید تنظیمات سیاستِ تمرکزِ «focus follows mouse»(تمرکز در پیروی از ماوس) را وارسی کنید. ممکن است ابتدا برای شما عجیب باشد که پنجره‌ها موقعی که تمرکز می‌یابند در روی دیگران قرار نمی‌گیرند(برای انجام این کار شما باید روی پنجره کلیک کنید)، اما شما از توانایی کار همزمان با بیش از

یک پنجره بدون آنکه پنجره فعال سایر پنجره‌ها را بپوشاند برخوردار خواهید شد. آن را امتحان کنید و با تعصب به آن برخورد نکنید، تصور می‌کنم آن را خواهید پسندید. شما می‌توانید این تنظیمات را در ابزارهای پیکربندی مدیر پنجره‌تان پیدا کنید.

« نگاهی به پیرامون
فهرست مطالب
پوسته چیست »

راهبری

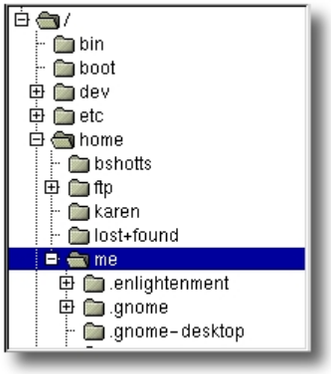
در این درس، من نخست سه فرمان را معرفی خواهم نمود: **pwd** (**p**rint **w**orking **d**irectory یا چاپ دایرکتوری کاری) و **cd** (**c**hange **d**irectory یا تعویض دایرکتوری)، و **ls** (**l**i**s**t files and directories یا لیست فایلها و دایرکتوریها).

اگر شما قبلاً با رابط خط فرمان کار نکرده‌اید، لازم است توجه دقیق‌تری به این درس داشته باشید، چون مفاهیم با کمی بردباری آسان خواهند شد.

سازمان‌دهی سیستم فایل

مانند آن سیستم عامل منسوخ، در یک سیستم لینوکس فایلها به طریقی مرتب می‌گردند که یک ***ساختار دایرکتوری سلسله مراتبی*** نامیده می‌شود. این به معنای آن است که آنها در یک الگوی درخت مانند از ***دایرکتوریها*** (در سایر سیستم‌ها folderها نامیده شده‌اند)،سازمان‌دهی می‌گردند،که ممکن است محتوی فایلها و دایرکتوریهای دیگری باشند. اولین دایرکتوری در این سیستم فایل ***دایرکتوری ریشه*** نامیده می‌شود. دایرکتوری ریشه شامل فایلها و دایرکتوری‌های فرعی است، که آنها محتوی فایلها و دایرکتوریهای فرعی دیگری هستند و همین طور و همین طور.

اکثر محیط‌های گرافیکی امروزه دارای برنامه مدیریت فایل برای نمایش و کار با سیستم فایل هستند. شما اغلب فایل سیستم نمایش داده شده به این شکل را خواهید دید:



یک تفاوت مهم میان سیستم عامل منسوخ و سیستم‌عامل‌های یونیکس-مانندی از قبیل لینوکس آن است که لینوکس مفهوم حروف درایوها را به کار نمی‌گیرد. در حالیکه حروف درایو سیستم فایل را به یک سری درخت‌های مختلف (یکی برای هر درایو) تفکیک می‌کند، لینوکس همیشه دارای یک درخت دایرکتوری است. دستگاه‌های ذخیره مختلف می‌توانند شامل انشعاب‌ها مختلفی از درخت باشند، اما درخت همواره یک درخت منفرد است.

pwd

چون یک رابط خط فرمان نمی‌تواند تصاویر گرافیکی از ساختار سیستم فایل ارایه کند، باید یک روش متفاوت برای نشان دادن آن داشته باشد. تصور کنید درخت سیستم فایل مانند یک مکان پر پیچ و خم (maze) است، و شما در آن ایستاده‌اید. در هر لحظه مفروض، شما در یک

دایرکتوری واحد مستقر شده‌اید. در داخل آن دایرکتوری، شما می‌توانید فایلهایش و خط سیر ***دایرکتوری والد*** آن و خط سیرهای منتهی به دایرکتوری‌های فرعی موجود در آن دایرکتوری را که در آن قرار دارید ببینید.

آن دایرکتوری که شما در آن قرار دارید ***دایرکتوری کاری*** نامیده می‌شود. برای پی بردن به نام دایرکتوری کاری، فرمان **pwd** را به کار ببرید.

```
[me@linuxbox me]$ pwd
/home/me
```

موقعی که نخست به یک سیستم لینوکس وارد می‌شوید، دایرکتوری کاری به ***دایرکتوری خانه*** شما تنظیم می‌گردد. اینجا مکانی است که شما فایلهایتان را قرار می‌دهید. روی اکثر سیستم‌ها، دایرکتوری خانه شما /home/your_user_name (نام کاربری شما) خواهد بود، اما بنا بر تمایل مدیر سیستم می‌تواند هر مورد دیگری هم باشد.

برای لیست کردن فایلهای داخل دایرکتوری کاری، فرمان **ls** را استفاده کنید.

```
[me@linuxbox me]$ ls

Desktop     Xrootenv.0    linuxcmd
GNUstep     bin           nedit.rpm
GUILG00.GZ  hitni123.jpg  nsmail
```

من در درس بعدی به **ls** باز خواهم گردید. موارد جالب بسیاری وجود دارد که می‌توان با آن انجام داد، اما ابتدا من باید یک مقدار در باره نام مسیرها و دایرکتوریها صحبت کنم.

cd

برای تعویض دایرکتوری کاری‌تان (آن دایرکتوری که داخل maze در آن ایستاده‌اید) از فرمان **cd** استفاده می‌کنید. برای انجام این کار، **cd** و به دنبال آن ***نام مسیر*** دایرکتوری مورد نظر را تایپ کنید. یک نام مسیر خط سیری است که شما را از میان انشعاب‌های درخت به دایرکتوری که می‌خواهید می‌رساند. نام مسیرها به دو روش می‌توانند مشخص گردند، ***نام مسیرهای مطلق*** یا ***نام مسیرهای نسبی***. بیایید ابتدا به نام مسیرهای مطلق نگاه کنیم.

یک نام مسیر مطلق همیشه با دایرکتوری ریشه شروع می‌شود و درخت را شاخه به شاخه ادامه می‌دهد تا مسیر دایرکتوری یا فایل مورد نظر تکمیل گردد. برای مثال، در سیستم شما یک دایرکتوری وجود دارد که اکثر برنامه‌ها در آنجا نصب می‌شوند. نام مسیر این دایرکتوری /usr/bin است. این یعنی در دایرکتوری ریشه (نمایش داده شده توسط یک اسلاش مقدم در نام مسیر) یک دایرکتوری به نام usr وجود دارد که شامل یک دایرکتوری به نام bin است.

اجازه بدهید این را عملاً آزمایش کنیم:

```
[me@linuxbox me]$ cd /usr/bin
[me@linuxbox bin]$ pwd
/usr/bin
```

```
[me@linuxbox bin]$ ls

[                       lwp-request
2to3                    lwp-rget
2to3-2.6                lxterm
a2p                     lz
aalib-config            lzcat
aconnect                lzma
acpi_fakekey            lzmadec
acpi_listen             lzmainfo
add-apt-repository      m17n-db
addpart                 magnifier

                و بسیار بیشتر...
```

اکنون می‌توانیم ببینیم که دایرکتوری کاری را به /usr/bin تعویض کرده‌ایم و آن دایرکتوری پُر از فایل است. توجه نمایید که چطور اعلان شما تغییر کرده است؟ به عنوان یک مزیت، اعلان فرمان معمولاً برای نمایش نام دایرکتوری کاری تنظیم می‌گردد.

در حالیکه یک نام مسیر مطلق از دایرکتوری ریشه آغاز می‌شود، نام مسیر نسبی از دایرکتوری کاری شروع می‌گردد. برای این منظور یک جفت نماد را جهت نشان دادن موقعیت‌های نسبی در درخت سیستم فایل به کار می‌برد. این نماد های ویژه "." (نقطه) و ".." (نقطه نقطه) می‌باشند.

نماد "." به خود دایرکتوری کاری اشاره می‌کند و نماد ".." به دایرکتوری والد دایرکتوری کاری اشاره می‌کند. اکنون چگونگی کارکرد آن. بیایید دایرکتوری کاری را دوباره به /usr/bin تعویض نماییم:

```
[me@linuxbox me]$ cd /usr/bin
[me@linuxbox bin]$ pwd
/usr/bin
```

بسیار خوب، اکنون اجازه بدهید فرض کنیم که ما می‌خواستیم دایرکتوری کاری را به والد /usr/bin که /usr است تعویض کنیم. حال این کار را به دو روش متفاوت می‌توانیم انجام بدهیم. اول، با یک نام مسیر مطلق:

```
[me@linuxbox bin]$ cd /usr
[me@linuxbox usr]$ pwd
/usr
```

یا، با یک نام مسیر نسبی:

```
[me@linuxbox bin]$ cd ..
[me@linuxbox usr]$ pwd
/usr
```

دو شیوه متفاوت با نتایج یکسان. شما کدام را باید به کار ببرید؟ آن روشی را که به حداقل تایپ کردن نیاز دارد!

به همچنین، به دو طریق مختلف می‌توانیم دایرکتوری کاری را از /usr به /usr/bin تعویض نماییم. اول با کاربرد نام مسیر مطلق:

```
[me@linuxbox usr]$ cd /usr/bin
[me@linuxbox bin]$ pwd
/usr/bin
```

یا، با یک نام مسیر نسبی:

```
[me@linuxbox usr]$ cd ./bin
[me@linuxbox bin]$ pwd
/usr/bin
```

اکنون، مطلب مهمی وجود دارد که باید من در اینجا اشاره کنم. تقریباً در تمام موقعیت‌ها، شما می‌توانید " **./** " را ذکر نکنید. به طور ضمنی به آن اشاره می‌شود. تایپ:

```
[me@linuxbox usr]$ cd bin
```

همان کار را انجام می‌داد. به طور کلی، اگر در موردی نام مسیر را معین نکنید، دایرکتوری کاری در نظر گرفته خواهد شد. یک استثنای مهم برای این مورد وجود دارد، اما فعلاً به آن نمی‌پردازم.

تعدادی میانبر

اگر شما **cd** را بدون چیزی در ادامه آن تایپ کنید، **cd** دایرکتوری کاری را به دایرکتوری خانگی شما تعویض می‌کند.

یک میانبر وابسته به آن تایپ **cd *~user_name*** است. در این حالت، **cd** دایرکتوری کاری را به دایرکتوری خانگی کاربر مشخص شده(م: به جای user_name) تعویض می‌کند.

تایپ **cd -** دایرکتوری کاری را به دایرکتوری قبلی تعویض می‌کند.(مترجم: یعنی آن دایرکتوری که قبل از آمدن به اینجا در آن مستقر بوده‌اید.)

نکات مهمی در مورد نام فایلها

1. نام فایلهایی که با کاراکتر نقطه شروع می‌شوند پنهان هستند. این تنها به معنای آن است که **ls** آنها را لیست نخواهد کرد مگر اینکه شما **ls -a** را به کار ببرید. وقتی حساب کاربری شما ایجاد شده، جهت ایجاد پیکربندی اقلام برای حساب شما چندین فایل پنهان

در دایرکتوری خانگی شما قرار داده شده است. بعد از این برای دیدن اینکه چطور می‌توانید **محیط**تان را سفارشی نمایید، ما نگاه نزدیک‌تری به برخی از این فایلها خواهیم داشت. علاوه بر این، برخی برنامه‌های کاربردی فایلهای پیکربندی و تنظیماتشان را به صورت فایلهای پنهان در دایرکتوری خانگی شما قرار می‌دهند.

2. در لینوکس همانند یونیکس، نام فایلها به حالت حروف حساس هستند. نام‌های فایل "File1" و "file1" به فایلهای متفاوتی اشاره می‌کنند.

3. لینوکس مانند سیستم‌های کهنه دارای مفهوم «پسوند فایل» نیست. شما می‌توانید به هر طریقی که مایل هستید فایلها را بنامید. اگر چه، در حالیکه لینوکس خودش به پسوند فایلها توجه ندارد، بسیاری برنامه‌های کاربردی توجه دارند.

4. با آنکه لینوکس از نام فایلهای طولانی دارای کاراکتر فاصله و با کاراکترهای نشانه‌گذاری تعبیه شده در آنها پشتیبانی می‌کند، کاراکترهای نشانه‌گذاری را به نقطه، خط تیره، و خط زیر(underscore) محدود کنید. **بالاتر از تمام ملاحظات، کاراکتر فاصله در نام فایلها تعبیه نکنید.** اگر خواهان فاصله بین کلمات در نام فایلها هستید، کاراکترهای خطزیر را به کار ببرید. بعداً از خودتان قدردانی خواهید نمود.

نگاهی به اطراف

اکنون که می‌دانید چگونه از دایرکتوری کاری به دایرکتوری دیگر حرکت کنید، می‌خواهیم در سیستم لینوکس شما گشتی بزنیم و در طول راه، مواردی در باره چگونگی رفتار آن بیاموزیم. اما قبل از اینکه شروع کنیم، من باید برخی ابزارها را که در جریان این سیاحت به کار می‌آیند، به شما بیاموزم. اینها عبارتند از:

- `ls` (لیست فایلها و دایرکتوریها)
- `less` (نمایش فایلهای متنی)
- `file` (رده‌بندی محتویات فایل)

ls

فرمان `ls` برای لیست کردن محتویات یک دایرکتوری استفاده می‌شود. احتمالاً این فرمان لینوکس دارای بیشترین مورد استفاده است. این فرمان به روشهای مختلفی می‌تواند به کار برود. در اینجا چند مثال آورده‌ایم:

مثالهای فرمان ls

فرمان	نتیجه
`ls`	لیست فایلهای دایرکتوری کاری
`ls /bin`	لیست فایلها در دایرکتوری bin/ (یا هر دایرکتوری دیگری که شمامایل به مشخص کردن آن باشید)

`ls -l`	لیست فایلهای دایرکتوری کاری در قالب بلند
`ls -l /etc /bin`	لیست فایلها در دایرکتوری bin/ و دایرکتوری etc/ در قالب طویل
`ls -la ..`	لیست تمام فایلها (حتی آنها که نامشان با کاراکتر نقطه شروع می‌شود، که به طور معمول پنهان هستند) در دایرکتوری والد دایرکتوری کاری با قالب طویل

این مثالها همچنین یک روش با اهمیت در مورد فرمانها را نمایان می‌سازند. اکثر فرمانها به این شکل عمل می‌کنند:

```
command -options arguments
```

که در اینجا *command* نام فرمان است، *options*- یک یا چند وسیله تنظیم رفتار فرمان است، و *arguments* (شناسه) یک یا چند «چیز» است که فرمان روی آن عمل می‌کند.

در مورد **ls**، می‌بینیم که **ls** نام فرمان است، و می‌تواند یک یا چند گزینه (option) داشته باشد، همچون **-a** و **-l**، و می‌تواند بر یک یا چند فایل یا دایرکتوری عمل کند.

یک نگاه دقیق‌تر به قالب طویل

اگر شما گزینه **-l** را با فرمان **ls** به کار ببرید، یک لیست فایل به دست می‌آورید که شامل اطلاعات فراوانی در باره فایلهایی است که لیست می‌شوند. این هم یک مثال:

```
-rw-------   1 bshotts  bshotts       576 Apr 17  1998 weather.txt
drwxr-xr-x   6 bshotts  bshotts      1024 Oct  9  1999 web_page
-rw-rw-r--   1 bshotts  bshotts    276480 Feb 11 20:41 web_site.tar
-rw-------   1 bshotts  bshotts      5743 Dec 16  1998 xmas_file.txt

----------    -------  -------  -------- ------------ -------------
    |            |        |         |         |            |
    |            |        |         |         |        نام فایل
    |            |        |         |         |
    |            |        |         |         +------  زمان ویرایش
    |            |        |         |
    |            |        |         +----------------  اندازه بر حسب بایت
    |            |        |
    |            |        +--------------------------  گروه
    |            |
```

```
    |           +----------------------------------   مالک
    |
    +-----------------------------------------------   مجوزهای فایل
```

نام فایل

نام فایل یا دایرکتوری.

زمان ویرایش

آخرین زمانی که فایل ویرایش گردیده است. در صورتیکه از آخرین ویرایش بیش از شش ماه گذشته باشد، تاریخ و سال نمایش داده می‌شود. در غیر اینصورت زمان روز نشان داده می‌شود.

اندازه

اندازه فایل بر حسب بایت.

گروه

نام گروهی که علاوه بر مالک فایل دارای مجوزهای فایل است.

مالک

نام کاربری که مالک فایل است.

مجوزهای فایل

یک نمایش از مجوزهای دسترسی فایل. کاراکتر اول نوع فایل است. یک "-" نشان دهنده یک فایل منظم (معمولی) می‌باشد. یک "d" نشان دهنده دایرکتوری است. مجموعه سه کاراکتر بعدی حق خواندن، نوشتن، و اجرا توسط مالک فایل را نشان می‌دهند. سه کاراکتر بعد از آن حقوق گروه فایل و سه کاراکتر آخر حقوق اعطا شده به هر شخص دیگر را نمایش می‌دهند. من این مطلب را در یکی از درسهای بعد با تفصیل بیشتر مطرح خواهم نمود.

less

`less` برنامه‌ای است که به شما امکان می‌دهد فایلهای متنی را نمایش بدهید. این خیلی مفید است چون بسیاری از فایلهای استفاده شده برای کنترل و پیکربندی لینوکس به صورت قابل خواندن توسط انسان(م: یعنی متنی) هستند.

«متن» چیست؟

روشهای بسیاری برای نمایندگی اطلاعات در یک کامپیوتر وجود دارد. تمام این شیوه‌ها مستلزم مشخص کردن یک ارتباط میان این اطلاعات و اعدادی است که برای نمایندگی آن به کار خواهند رفت. در هر صورت، کامپیوترها فقط اعداد را می‌فهمند و تمام داده‌ها به نماینده عددی تبدیل می‌شوند.

برخی از این سیستم‌های نمایندگی (از قبیل فایلهای فشرده شده چند رسانه‌ای) بسیار پیچیده‌اند، در حالیکه سایرین نسبتاً ساده هستند. یکی از قدیمی‌ترین و ساده‌ترین آنها *ASCII text* نامیده می‌شود. ASCII (اسکی تلفظ شده است) کوته نوشتی از **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange (کد استاندارد امریکایی برای تبادل اطلاعات) است. این یک طرح رمزی کردن ساده بود که نخست در ماشین‌های دورنویس (Teletype) برای نگاشت (بازنمایی) کاراکترهای صفحه‌کلید به وسیله اعداد، استفاده شد.

متن، یک نگاشتِ سادهِ یک به یکِ کاراکترها با اعداد است. بسیار خلاصه است. پنجاه کاراکتر متن را به پنجاه بایت داده ترجمه می‌کند. در سراسر یک سیستم لینوکس، فایلهای بسیاری در قالب متن ذخیره می‌شوند و ابزارهای لینوکس فراوانی وجود دارد که با فایلهای متنی کار می‌کنند. حتی سیستم‌عامل‌های کهنه اهمیت این قالب را به رسمیت می‌شناسند. برنامه شناخته شده NOTEPAD.EXE، ویرایشگری برای

فایلهای متن اسکی ساده است.

برنامه **less** به این شکل فراخوانی می‌گردد، با تایپ:

```
less text_file
```

این فرمان فایل را نمایش خواهد داد.

کنترل نمودن less

وقتی **less** شروع می‌شود، در هر زمان یک صفحه از فایل متنی را نمایش خواهد داد. شما می‌توانید کلیدهای Page Up و Page Down را برای حرکت درفایل متن به کار ببرید. برای خروج از **less**، کاراکتر "q" را تایپ کنید. در اینجا برخی از فرمانهایی که **less** قبول می‌کند آمده است:

فرمانهای صفحه کلید برای برنامه less

فرمان	اقدام
b یا Page Up	یک صفحه به عقب رفتن
space یا Page Down	یک صفحه به پیش رفتن
G	رفتن به انتهای فایل متن
1G	رفتن به ابتدای فایل
/*characters*	جستجو به طرف جلو در فایل متن جهت یک مورد حضور *کاراکترهای* مشخص شده
n	تکرار جستجوی قبلی
h	نمایش لیست کامل فرمانها و گزینه‌های less
q	خروج

file

هنگامیکه در هر طرفِ سیستمِ لینوکس‌تان گردش می‌کنید، قبل از کوشش برای نمایش داده‌های یک فایل، تعیین نوع آن سودمند است. این جایی است که فرمان **file** به کار می‌آید. **file** یک فایل را بازدید می‌کند و به شما می‌گوید که کدام نوع از فایل است.

برای استفاده از برنامه **file**، فقط تایپ کنید:

```
file name_of_file
```

برنامه **file** اکثر انواع فایلها را شناسایی می‌کند، از جمله:

انواع مختلف فایلها

نوع فایل	شرح	قابل نمایش به عنوان متن؟
متن ASCII	همه چیز از نامش پیداست	بله
اسکریپت Bourne-Again shell	یک اسکریپت **bash**	بله
فایل ELF 32-bit LSB core	یک فایل رونوشت core (یک برنامه موقعی که crash می‌کند آن را تولید خواهد کرد)	خیر
ELF 32-bit LSB قابل اجرا	یک برنامه قابل اجرای باینری	خیر
ELF 32-bit LSB shared object	یک کتابخانه به اشتراک گذاشته شده	خیر
آرشیو tar گنو	یک فایل **t**ape **ar**chive. روش متداول برای ذخیره گروهی از فایلها.	خیر، برای نمایش لیست فایلها **tar tvf** را به کار ببرید.
داده‌های فشرده شده gzip	یک آرشیو فشرده شده با برنامه **gzip**	خیر
متن سند HTML	یک صفحه وب	بله
داده تصویری JPEG	یک تصویر فشرده شده JPEG	خیر
متن سند PostScript	یک فایل PostScript	بله
RPM	یک آرشیو مدیر بسته ردهت	خیر، برای بررسی محتویاتش از **rpm -q** استفاده کنید.

داده‌های بایگانی Zip	یک بایگانی فشرده شده با **zip**	خیر

در حالیکه ممکن است به نظر برسد اکثر فایلها نمی‌توانند به عنوان متن نمایش داده شوند، شما از اینکه چه تعدادی می‌توانند نمایش داده شوند شگفت‌زده می‌شوید. این مطلب مخصوصاً برای فایلهای پیکربندی مهم صحیح است. شما همچنین در جریان سیاحت ما متوجه خواهید گردید که بسیاری از ویژگی‌های سیستم عامل به وسیله اسکریپت‌های پوسته کنترل می‌شوند. **در لینوکس، اسراری وجود ندارد!**

« مدیریت فایلها | فهرست مطالب | نگاهی به پیرامون »

یک تور آموزشی

اکنون هنگام سیاحت کردن است. جدول پایین برخی محل‌های جالب برای اکتشاف را لیست می‌کند. این به هیچ وجه یک لیست کامل نیست، اما باید جالب بودن سیاحت را آشکارکند. برای هر دایرکتوری لیست شده در پایین این موارد را انجام بدهید:

- **cd** به داخل هر دایرکتوری.
- کاربرد **ls** برای لیست کردن محتویات دایرکتوری.
- اگر یک فایل جالب توجه می‌بینید، استفاده از فرمان **file** برای تعیین نوع محتویات آن.
- برای فایلهای متن، کاربرد **less** به منظور نمایش آنها.

دایرکتوری‌های جالب توجه و محتویات آنها

دایرکتوری	شرح
/	دایرکتوری ریشه، جایی که سیستم فایل شروع می‌شود. در اکثر موارد دایرکتوری ریشه فقط محتوی دایرکتوری‌های فرعی است.
/boot	در اینجا هسته لینوکس و فایلهای بار کننده سیستم‌عامل نگهداری می‌شوند. هسته لینوکس فایلی به نام **vmlinuz** است.
/etc	دایرکتوری **/etc** شامل فایلهای پیکربندی سیستم است. تمام فایلهای داخل **/etc** باید فایلهای متن باشند. محل‌های جالب توجه: **/etc/passwd** فایل **passwd** شامل اطلاعات ضروری برای هر کاربر است. جایی است که کاربران تعریف می‌شوند. **/etc/fstab** فایل **fstab** شامل جدولی از دستگاه‌هایی است که موقع بالا آمدن سیستم شما متصل می‌گردند. این فایل درایوهای دیسک شما را تعریف می‌کند. **/etc/hosts** این فایل نام میزبان‌ها و آدرس‌های IP است که به طور درونی برای سیستم شناخته شده‌اند.

	/etc/init.d این دایرکتوری شامل اسکریپت‌هایی است که سرویس‌های مختلف سیستم را به طور نمونه در زمان بالا آمدن سیستم دایر می‌کنند.
/bin, /usr/bin	این دو دایرکتوری شامل اکثر برنامه‌های سیستم هستند. دایرکتوری **/bin** دارای برنامه‌های خاصی است که سیستم برای عمل کردن نیاز دارد، در حالیکه **/usr/bin** محتوی برنامه‌هایی برای کاربران سیستم است.
/sbin, /usr/sbin	دایرکتوری‌های **sbin** شامل برنامه‌هایی برای مدیریت سیستم هستند، بیشتر برای استفاده توسط کاربر ارشد.
/usr	دایرکتوری **/usr** شامل اقلام متنوعی است که برنامه‌های کاربردی کاربر را پشتیبانی می‌کنند. برخی از مهمترین‌ها: **/usr/share/X11** فایلهای پشتیبانی سیستم پنجره X **/usr/share/dict** واژه‌نامه‌ها برای بررسی درست نوشتن. شرط می‌بندم که نمی‌دانستید لینوکس یک بازرسی کننده درست نوشتن داشته باشد. مستندات **look** و **aspell** را ببینید. **/usr/share/doc** فایلهای مستندات گوناگون در قالب‌های متنوع. **/usr/share/man** صفحات man در اینجا نگهداری می‌شوند. **/usr/src** فایلهای کد منبع. اگر شما بسته کد منبع هسته را نصب کرده باشید، کل کد منبع هسته لینوکس را در اینجا پیدا خواهید نمود.
/usr/local	**/usr/local** و دایرکتوری‌های فرعی آن جهت نصب نرم‌افزار و فایلهای دیگر برای استفاده در ماشین محلی استفاده می‌شود. این در واقع به معنای آن است که نرم‌افزاری که بخشی از توزیع رسمی (که معمولاً به **/usr/bin** می‌روند) نیست در اینجا قرار می‌گیرد. موقعی که شما برنامه‌های جالبی برای نصب در سیستم خود به دست می‌آورید، آنها باید در یکی از دایرکتوری‌های **/usr/local** نصب بشوند. بیشتر مواقع، دایرکتوری انتخابی **/usr/local/bin** است.
/var	دایرکتوری **/var** محتوی فایلهایی است که هنگام در حال اجرا بودن سیستم تغییر می‌کنند. شامل این موارد: **/var/log** دایرکتوری که شامل فایلهای ثبت رخداد است. این فایلها هنگامیکه سیستم کار می‌کند به روز رسانی می‌گردند. شما باید برای کنترل کردن سلامت سیستم، گاه گاهی فایلهای این دایرکتوری را نگاه کنید. **/var/spool**

	این دایرکتوری برای نگهداری فایلهایی به کار می‌رود که برای برخی پردازش‌ها در نوبت گذاشته می‌شوند، از قبیل پیغام‌های پستی و کارهای چاپ. موقعی که اول پیغام پستی کاربر در سیستم محلی می‌رسد (فرض که نامه رسانی محلی دارید)، ابتدا پیغام‌ها در **/var/spool/mail** ذخیره می‌شوند.
/lib	کتابخانه‌های به اشتراک گذاشته شده (مشابه DLLها در آن سیستم عامل دیگر) در اینجا نگهداری می‌شوند.
/home	**/home** جایی است که کاربران کارهای شخصی‌شان را نگهداری می‌کنند. به طور کلی، اینجا تنها مکانی است که کاربران مجاز می‌شوند در فایلها بنویسند.این کار چیزها را خوب و پاکیزه نگاه می‌دارد
/root	این دایرکتوری خانگی کاربر ارشد است.
/tmp	**/tmp** یک دایرکتوری است که برنامه‌ها می‌توانند فایلهای موقت‌شان را در آن بنویسند.
/dev	دایرکتوری **/dev** یک دایرکتوری خاص است، چون به راستی شامل فایلهای به مصداق معمول نیست. بلکه، شامل دستگاه‌هایی است که برای سیستم در دسترس هستند. در لینوکس (مانند یونیکس)، با دستگاه‌ها مانند فایلها رفتار می‌شود. شما می‌توانید دستگاه‌ها را مثل اینکه آنها فایل هستند بخوانید و در آنها بنویسید. برای مثال **/dev/fd0** گرداننده دیسک نرم اول است، **/dev/sda** (و **/dev/hda** روی سیستم‌های قدیمی‌تر) گرداننده دیسک سخت اول است. تمام دستگاه‌هایی که هسته می‌شناسد در اینجا نمایانده می‌شوند.
/proc	دایرکتوری **/proc** نیز ویژه است. این دایرکتوری شامل فایلها نیست. در حقیقت، این دایرکتوری به هیچ وجه واقعاً وجود ندارد. کاملاً مجازی است. دایرکتوری **/proc** شامل روزنه‌های کوچکی در خود کرنل است. یک گروه اقلام شماره‌دار در این دایرکتوری وجود دارد که در ارتباط با تمام پردازش‌های در حال اجرا در سیستم هستند. همچنین، تعدادی اقلام نامبرده وجود دارند که دسترسی به پیکربندی جاری سیستم را مجاز می‌کنند. بسیاری از این اقلام می‌توانند دیده شوند. دیدن **/proc/cpuinfo** را امتحان کنید. این مدخل به شما می‌گوید که هسته در باره CPU شما چه می‌اندیشد.
/media,/mnt	سرانجام، به **/media** می‌رسیم، یک دایرکتوری معمولی که به روش خاصی استفاده می‌شود. دایرکتوری **/media** برای *محل‌های اتصال* به کار می‌رود. چنانکه در درس دوم آموختیم، دستگاه‌های فیزیکی ذخیره مختلف (مانند دیسک‌گردان‌ها) در محل‌های مختلفی به درخت سیستم فایل متصل می‌گردند. این فرایند اتصال یک دستگاه به درخت *mounting* نامیده می‌شود. برای اینکه یک دستگاه قابل استفاده باشد، اول باید mount(متصل) بشود. موقعی که سیستم شما بالا می‌آید، فهرست دستورالعمل‌های متصل کردن در فایل **/etc/fstab** را می‌خواند، که شرح می‌دهند کدام دستگاه در کدام محل اتصال در درخت دایرکتوری متصل می‌شود. این فایل اتصالِ دیسک‌های سخت را تامین می‌کند، اما ممکن است شما دارای دستگاه‌هایی نیز باشید که به طور موقتی بازبینی می‌شوند، از قبیل CD-ROMها، و گردانندههای دیسک نرم. چون اینها جدا شدنی هستند، آنها تمام وقت متصل باقی نمی‌مانند. دایرکتوری **/media** به وسیله مکانیسمِ متصل کردنِ خودکارِ موجود در توزیع‌های لینوکس مدرنِ متمایل به میز کار استفاده می‌شود. در سیستم‌هایی که نیازمند اتصال دستی دستگاه‌های جدا شدنی هستند، دایرکتوری **/mnt** مکان مناسبی برای اتصال این دستگاه‌های موقتی فراهم می‌کند. شما بارها دایرکتوری‌های **/mnt/floppy** و **/mnt/cdrom** را خواهید دید. برای دیدن آنکه کدام دستگاه‌ها و محل‌های اتصال استفاده می‌شوند، تایپ کنید **mount**.

یک نوع فایل مرموز...

در خلال این سیاحت، احتمالاً شما به یک نوع مدخل عجیب دایرکتوری توجه نمودید، مخصوصاً در دایرکتوری‌های **/boot** و **/lib** موقعی که با **ls -l** لیست شدند، شما باید موردی مانند این را دیده باشید:

```
lrwxrwxrwx     25 Jul  3 16:42 System.map -> /boot/System.map-2.0.36-3
-rw-r--r-- 105911 Oct 13  1998 System.map-2.0.36-0.7
-rw-r--r-- 105935 Dec 29  1998 System.map-2.0.36-3
-rw-r--r-- 181986 Dec 11  1999 initrd-2.0.36-0.7.img
-rw-r--r-- 182001 Dec 11  1999 initrd-2.0.36.img
lrwxrwxrwx     26 Jul  3 16:42 module-info -> /boot/module-info-2.0.36-3
-rw-r--r--  11773 Oct 13  1998 module-info-2.0.36-0.7
-rw-r--r--  11773 Dec 29  1998 module-info-2.0.36-3
lrwxrwxrwx     16 Dec 11  1999 vmlinuz -> vmlinuz-2.0.36-3
-rw-r--r-- 454325 Oct 13  1998 vmlinuz-2.0.36-0.7
-rw-r--r-- 454434 Dec 29  1998 vmlinuz-2.0.36-3
```

به فایلهای `System.map`، `module-info` و `vmlinuz` توجه کنید. یک نشانه ناشناس را پس از نام فایلها می‌بینید؟

این سه فایل *پیوندهای نمادین* نامیده می‌شوند. پیوندهای نمادین یک نوع خاص فایل هستند که به یک فایل دیگر اشاره می‌کنند. با پیوندهای نمادین، داشتن نامهای چندگانه برای یک فایل واحد امکان پذیر می‌گردد. چگونگی کارکرد آن این است: هر گاه نام فایلی به سیستم داده شود که لینک نمادین باشد، به طور ناپیدا آن را به فایلی که در حال اشاره به آن است، هدایت می‌کند.

دقیقاً این برای چه موردی مناسب است؟ این یک خصیصه بسیار سودمند است. بیایید لیست دایرکتوری فوق را در نظر بگیریم (که دایرکتوری `/boot` از یک سیستم قدیمی Red Hat 5.2 است). این سیستم دارای نگارش‌های نصب شده چندین هسته لینوکس بوده است. ما این مطلب را از فایلهای `vmlinuz-2.0.36-0.7` و `vmlinuz-2.0.36-3` می‌توانیم ببینیم. این نام فایلها اظهار می‌کنند که هر دو نگارش 2.0.36-0.7 و 2.0.36-3 نصب شده‌اند. به دلیل نام فایلهای شامل شماره نگارش، دیدن تفاوت‌ها در لیست دایرکتوری آسان است. اگر چه، این مطلب باعث سردرگمی برنامه‌هایی می‌شد که به نام ثابت فایل کرنل استناد می‌کنند این برنامه‌ها ممکن بود انتظار داشته باشند که هسته لینوکس به سادگی "`vmlinuz`" نامیده شود. اینجا آنجایی است که حُسنِ پیوند نمادین به کار می‌آید. به وسیله ایجاد یک پیوند نمادین به نام `vmlinuz` که به `vmlinuz-2.0.36-3` اشاره می‌کند، ما مشکل را حل کرده‌ایم.

برای ایجاد پیوند نمادین، از فرمان **ln** استفاده کنید.

مترجم: در ضمن به اولین کاراکتر سطر مربوط به این فایلها توجه نمایید، حرف **l** بیانگر لینک بودنشان است.

دستکاری فایلها

این درس فرمانها زیر را به شما معرفی خواهد نمود:

- `cp` ـ کپی فایلها و دایرکتوری‌ها
- `mv` ـ جابجایی یا تغییر نام فایلها و دایرکتوری‌ها
- `rm` ـ حذف فایلها و دایرکتوری‌ها
- `mkdir` ـ ایجاد دایرکتوری‌ها

این چهار فرمان در میان فرمانهای لینوکس دارای رایج‌ترین مورد استفاده هستند. آنها فرمانهای اساسی برای دستکاری فایلها و دایرکتوری‌ها هستند.

اکنون، صادقانه، برخی از وظایف انجام شده با این فرمانها به طور آسانتری با یک مدیر فایل گرافیکی انجام می‌شوند. با یک مدیر فایل، شما می‌توانید یک فایل را از یک دایرکتوری به دیگری بکشید و بیاندازید، قیچی کنید و بچسبانید، و غیره. پس این برنامه‌های خط فرمان قدیمی برای چه؟

پاسخ در قدرت و انعطاف پذیری است. در حالیکه انجام دستکاری ساده فایل با یک مدیر فایل گرافیکی آسان است، وظایف پیچیده با برنامه‌های خط فرمان آسانتر می‌توانند انجام بشوند. برای مثال، چطور تمام فایلهای HTML را از یک دایرکتوری به دیگری کپی می‌کردید، اما فقط کپی فایلهایی که در دایرکتوری مقصد وجود نداشتند یا جدیدتر از نگارش موجود در دایرکتوری مقصد بودند؟ به وسیله یک مدیر فایل تا حدی سخت. به وسیله خط فرمان به طور دلپذیر آسان:

```
[me@linuxbox me]$ cp -u *.html destination
```

کاراکترهای عام

قبل از شروع با فرمانهایمان، لازم است در باره یک ویژگی پوسته که این فرمانها را چنین قدرتمند می‌سازد صحبت کنم. چون پوسته نام فایلها را زیاد استفاده می‌کند، برای کمک به شما که گروه‌های فایلها را به سرعت مشخص نمایید، کاراکترهای مخصوصی فراهم می‌کند. این کاراکترهای ویژه **کاراکترهای عام** (*wildcards*) نامیده می‌شوند. کاراکترهای عام به شما امکان می‌دهند نام فایلها را براساس الگوهایی از کاراکترها انتخاب کنید. جدول زیر کاراکترهای عام و آنچه آنها انتخاب می‌کنند را لیست می‌کند:

خلاصه کاراکترهای عام و معنی آنها

کاراکتر عام	معنی
*	با همه کاراکترها منطبق می‌گردد
?	با هر کاراکتر منفرد منطبق می‌شود

<table>
<tr><td>[characters]</td><td>بر هر کاراکتری که عضوی از مجموعه characters باشد منطبق می‌شود. مجموعه کاراکترها می‌تواند به عنوان یک کلاس کاراکتر POSIX نیز بیان بشود مانند یکی از موارد زیر:
کلاسهای کاراکتر POSIX<table><tr><td>[:alnum:]</td><td>کاراکترهای الفبا عددی</td></tr><tr><td>[:alpha:]</td><td>کاراکترهای الفبایی</td></tr><tr><td>[:digit:]</td><td>عددی</td></tr><tr><td>[:upper:]</td><td>کاراکترهای الفبایی حروف بزرگ</td></tr><tr><td>[:lower:]</td><td>کاراکترهای الفبایی حروف کوچک</td></tr></table></td></tr>
<tr><td>[!characters]</td><td>با هر کاراکتری که عضوی از مجموعه characters نباشد منطبق می‌گردد</td></tr>
</table>

با استفاده از کاراکترهای عام ساختن ضوابط بسیار پیچیده برای انتخاب نام فایلها امکان‌پذیر است. این هم چند مثال از الگوها و آنچه بر آنها منطبق می‌شوند:

مثالهای انطباق کاراکترهای عام

الگو	موارد انطباق
`*`	نام تمام فایلها
`g*`	تمام فایلهایی که نامشان با کاراکتر g شروع می‌شود
`b*.txt`	تمام فایلهایی که نام آنها با کاراکتر b شروع می‌شود و با کاراکترهای .txt تمام می‌شوند
`Data???`	هر فایلی که نام آن باکاراکترهای Data شروع بشود و دقیقاً سه کاراکتر بیشتر از آن در ادامه‌اش باشد
`[abc]*`	هر فایلی که نام آن با a یا b یا c شروع بشود و با هر کاراکتر یا کاراکترهای دیگر ادامه یابد
`[[:upper:]]*`	هر فایلی که نام آن بایک حرف بزرگ شروع بشود. این یک مثال از کلاس کاراکتر است.

`BACKUP.[[:digit:]][[:digit:]]`	یک مثال دیگرِ کلاس‌های کاراکتر. این الگو بر هر نام فایلی که با کاراکترهای BACKUP. شروع شده و دقیقاً با دو کاراکتر عددی ادامه یابد، منطبق می‌گردد.
`*[![:lower:]]`	هر نام فایلی که به یک حرف کوچک ختم نمی‌گردد.

شما می‌توانید از کاراکترهای عام با هر فرمانی که نام فایل‌ها را به عنوان شناسه می‌پذیرد استفاده کنید.

cp

برنامه **cp** فایلها و دایرکتوری‌ها را کپی می‌کند. در این ساده‌ترین شکل، یک فایل واحد را کپی می‌کند:

```
[me@linuxbox me]$ cp file1 file2
```

همچنین می‌تواند برای کپی چندین فایل (و-یا دایرکتوری) به یک دایرکتوری متفاوت به کار برود:

```
[me@linuxbox me]$ cp file... directory
```

یک نکته در باره نشانه‌گذاری: ... حاکی از آن است که یک فقره می‌تواند یکبار یا بیشتر تکرار گردد.

مثالهای مفید دیگری از **cp** و گزینه‌هایش:

مثالهای فرمان cp

فرمان	نتایج
`cp file1 file2`	محتویات ***file1*** را به ***file2*** کپی می‌کند. اگر ***file2*** موجود نباشد، ایجاد می‌شود، **وگرنه , *file2* به طور خاموش و بدون اعلام به وسیله محتویات *file1* رونویسی می‌گردد.**
`cp -i file1 file2`	مانند مورد فوق اگر چه، چون گزینه -i (interactive) تعیین گردیده، در صوریتکه file2 موجود باشد، قبل از رونویسی با محتویات ***file1*** کاربر به پاسخ‌گویی وادار می‌شود.
`cp file1 dir1`	محتویات ***file1*** را (داخل فایلی به نام ***file1***) در دایرکتوری ***dir1*** کپی می‌کند.
`cp -R dir1 dir2`	کپی محتویات دایرکتوری ***dir1*** به دایرکتوری ***dir2***. اگر دایرکتوری ***dir2*** وجود نداشته باشد، ایجاد می‌گردد. وگرنه یک دایرکتوری به نام ***dir1*** در داخل دایرکتوری ***dir2*** ایجاد می‌گردد.

mv

فرمان **mv** فایلها و دایرکتوری‌ها را بر اساس چگونگی کاربرد آن تغییر محل یا تغییر نام می‌دهد. یک فایل یا بیشتر را به یک دایرکتوری متفاوت منتقل خواهد نمود، یا یک فایل یا دایرکتوری را تغییر نام خواهد داد. برای تعییر نام فایل، به این صورت به کار می‌رود:

```
[me@linuxbox me]$ mv filename1 filename2
```

برای انتقال فایلها (و-یا دایرکتوری‌ها) به دایرکتوری متفاوت:

```
[me@linuxbox me]$ mv file... directory
```

مثالهایی از **mv** و گزینه‌هایش:

مثالهای فرمان mv

فرمان	نتایج
`mv file1 file2`	اگر ***file2*** موجود نباشد، آنوقت ***file1*** به ***file2*** تعییر نام داده می‌شود. **اگر *file2* موجود باشد، محتویاتش به طور خاموش و بدون اعلام با محتویات *file1* رو نویسی می‌گردد.**
`mv -i file1 file2`	مانند مورد فوق اما، چون `-i` (interactive) تعیین گردیده است، اگر ***file2*** موجود باشد، قبل از رونویسی با محتویات ***file1*** کاربر به پاسخ‌گویی وادار می‌شود.
`mv file1 file2 file3 dir1`	فایلهای ***file1, file2, file3*** به دایرکتوری ***dir1*** منتقل می‌شوند. اگر ***dir1*** موجود نباشد، **mv** با یک پیغام خطا خارج خواهد گردید.
`mv dir1 dir2`	اگر ***dir2*** وجود نداشته باشد، آنوقت ***dir1*** به ***dir2*** تغییر نام داده می‌شود. در صورتیکه ***dir2*** موجود باشد، دایرکتوری ***dir1*** به داخل دایرکتوری ***dir2*** منتقل می‌گردد.

rm

فرمان **rm** فایلها و دایرکتوری‌ها را پاک (حذف) می‌کند.

```
[me@linuxbox me]$ rm file...
```

همچنین می‌تواند برای حذف دایرکتوری‌ها به کار برود:

```
[me@linuxbox me]$ rm -r directory...
```

مثالهایی از **rm** و گزینه‌هایش:

مثالهای فرمان rm

فرمان	نتایج
`rm file1 file2`	حذف ***file1*** و ***file2***.
`rm -i file1 file2`	مانند مورد فوق اما، `-i` (interactive) تعیین گردیده است، قبل از اینکه هر فایل حذف گردد کاربر وادار به پاسخ‌گویی می‌شود.
`rm -r dir1 dir2`	دایرکتوری‌های ***dir1*** و ***dir2*** همراه با تمام محتویات آنها حذف می‌گردند.

با rm محتاط باشید!

لینوکس دارای فرمان undelete نیست. وقتی شما موردی را با **rm** حذف کنید، از دست رفته است. اگر با **rm** محتاط نباشید، شما می‌توانید آسیب هولناکی به سیستم خود وارد آورید، مخصوصاً همراه با کاراکترهای عام.

قبل از اینکه rm را با کاراکترهای عام به کار ببرید، این شگرد سودمند را امتحان کنید: به جای آن فرمان‌تان را با استفاده از **ls** بسازید. با انجام این کار، می‌توانید تأثیر کاراکترهای عام به کار گرفته را قبل از حذف فایلها مشاهده کنید. پس از اینکه فرمان‌تان را با **ls** آزمایش کرده‌اید، فرمان را با کلید جهت‌ـبالا فراخوانی کرده و سپس **rm** را در فرمان به جای **ls** جایگزین کنید.

mkdir

فرمان **mkdir** برای ایجاد دایرکتوری‌ها به کار می‌رود. برای استفاده از آن، به سادگی تایپ کنید:

```
[me@linuxbox me]$ mkdir directory...
```

استفاده از فرمانها با کاراکترهای عام

چون فرمانهایی که ما در اینجا پوشش دادیم نام چند فایل و دایرکتوری را به عنوان شناسه قبول می‌کنند، شما می‌توانید از کاراکترهای عام برای مشخص کردن آنها استفاده کنید. این هم چند مثال:

مثالهای فرمان با کاربرد کاراکترهای عام

فرمان	نتایج

`cp *.txt text_files`	کپی تمام فایلهای دایرکتوری کاری فعلی که دارای نامهای ختم شده به کاراکترهای .txt هستند به یک دایرکتوری موجود به نام ***text_files***
`mv my_dir ../*.bak my_new_dir`	انتقال دایرکتوری فرعی ***my_dir*** و تمام فایلهای دایرکتوری پدر دایرکتوری کاری جاری که نامشان به .bak ختم می‌شود به یک دایرکتوری موجود به نام ***my_new_dir***
`rm *~`	حذف تمام فایلهای دایرکتوری جاری که به کاراکتر ~ ختم می‌گردند. برخی برنامه‌های کاربردی فایلهای پشتیبان را با استفاده از این طرحِ نام گذاری ایجاد می‌کنند. کاربرد این فرمان آنها را از دایرکتوری پاک می‌کند.

« تغییر مسیر ورودی–خروجی
فهرست مطالب
کار با فایلها »

کار با فرمانها

تا اینجا شما تعدادی از فرمانها با گزینه‌ها و شناسه‌های مبهم آنها را دیده‌اید. در این درس، کوشش خواهیم کرد مقداری از آن ابهام را برطرف نماییم. این درس فرمانهای زیر را معرفی خواهد نمود.

- `type` ـ اطلاعاتی در باره نوع فرمان نمایش می‌دهد
- `which` ـ محل یک فرمان را مشخص می‌کند
- `help` ـ صفحه مرجع داخلی پوسته را نمایش می‌دهد
- `man` ـ برای نمایش یک مرجعِ فرمانِ حین کار است

«فرمانها» کدامند؟

فرمانها می‌توانند در یکی از این چهار دسته متفاوت باشند:

1. **یک برنامه اجرایی** مانند تمام فایلهایی که ما در /usr/bin دیدیم. در داخل این دسته، برنامه‌ها می‌توانند ***باینری‌های کامپایل شده*** از قبیل برنامه‌های نوشته شده در C و C++ باشند، یا برنامه‌های نوشته شده در ***زبانهای اسکریپت‌نویسی***، از قبیل پوسته، پرل، پایتون، روبی و غیره باشند.
2. **یک فرمان ساخته شده در داخل خود پوسته.** bash یک تعداد فرمان از درون فراهم می‌کند که ***builtinهای پوسته*** نامیده می‌شوند. برای مثال، فرمان `cd` یک builtin (داخلی) پوسته است.
3. **یک تابع پوسته.** اینها اسکریپت‌های پوسته کوچکی هستند که در داخل ***محیط*** ترکیب شده‌اند. ما در درسهای بعدی، پیکربندی محیط و نوشتن توابع پوسته را پوشش خواهیم داد، اما برای حالا، فقط مطلع باشید که آنها وجود دارند.
4. **یک مستعار.** فرمانهایی که شما خودتان می‌توانید با ساختن از سایر فرمانها تعریف کنید، این مورد بعداً در یک درس پوشش داده خواهد شد.

شناسایی فرمانها

اغلب دانستن آن که دقیقاً کدامیک از چهار نوع فرمان استفاده می‌شود مفید است و لینوکس دو روش برای پی بردن به آن ارایه می‌کند.

type

فرمان **type** یک دستور داخلی پوسته است و نوع فرمانی را نمایش می‌دهد که پوسته با یک نام فرمان داده شدهِ بخصوص اجرا خواهد نمود. به این طریق کار می‌کند:

```
type command
```

که در آن «command» نام فرمانی است که شما می‌خواهید بررسی کنید.این هم چند مثال:

```
[me@linuxbox me]$ type type
type is a shell builtin
[me@linuxbox me]$ type ls
ls is aliased to `ls --color=tty'
[me@linuxbox me]$ type cp
cp is /bin/cp
```

در اینجا ما نتایج برای سه فرمان متفاوت را می‌بینیم. توجه کنید که در آن مورد مربوط به ls (اخذ شده از یک سیستم فدورا) چطور فرمان ls در حقیقت یک مستعار برای فرمان ls همراه با گزینه افزوده شده color=tty -- است. اکنون می‌دانیم که چرا خروجی ls رنگی نمایش داده می‌شود!

which

گاهی اوقات بیش از یک نگارشِ برنامه قابل اجرای نصب شده روی یک سیستم وجود دارد. در حالیکه این مطلب در سیستم های میزکار رایج نیست، در سرویس‌دهنده‌های بزرگ غیر معمول نیست. برای تعیین محل دقیق یک برنامه اجراشدنیِ مفروض، فرمان **which** استفاده می‌شود:

```
[me@linuxbox me]$ which ls
/bin/ls
```

which تنها برای برنامه‌های قابل اجرا کار می‌کند، نه برای داخلی‌ها و نه برای مستعارهایی که جایگزین برنامه‌های اجرایی واقعی می‌گردند.

دریافت مستندات فرمان

با این آگاهی که یک فرمان از کدام دسته است، اکنون ما می‌توانیم مستندات در دسترس برای هر نوع فرمان را جستجو کنیم.

help

bash دارای یک وسیله help درون ساختِ قابل دسترس برای هر فرمان داخلی پوسته است. برای استفاده آن، help و به دنبال آن نام فرمان داخلی انتخابی پوسته را تایپ کنید، می‌توانید گزینه -m را برای تغییر قالب خروجی اضافه نمایید. برای مثال:

```
[me@linuxbox me]$ help -m cd

NAME
    cd - Change the shell working directory.

SYNOPSIS
    cd [-L|-P] [dir]

DESCRIPTION
    Change the shell working directory.

    Change the current directory to DIR.  The default DIR is the value of the
    HOME shell variable.

    The variable CDPATH defines the search path for the directory containing
    DIR.  Alternative directory names in CDPATH are separated by a colon (:).
    A null directory name is the same as the current directory.  If DIR begins
    with a slash (/), then CDPATH is not used.

    If the directory is not found, and the shell option `cdable_vars' is set,
    the word is assumed to be  a variable name.  If that variable has a value,
    its value is used for DIR.

    Options:
        -L      force symbolic links to be followed
        -P      use the physical directory structure without following symbolic
        links

    The default is to follow symbolic links, as if `-L' were specified.

    Exit Status:
    Returns 0 if the directory is changed; non-zero otherwise.

SEE ALSO
    bash(1)

IMPLEMENTATION
    GNU bash, version 4.1.5(1)-release (i486-pc-linux-gnu)
    Copyright (C) 2009 Free Software Foundation, Inc.
```

یادداشتی در باره نشانه‌گذاری: وقتی براکت‌ها در تعریف ترکیب دستوری پوسته ظاهر می‌شوند، نشان‌دهنده انتخابی بودن اقلام هستند. یک کاراکتر میله عمودی بیانگر اقلام انحصاری دوطرفه (دوگانه غیرقابل جمع) هستند. در مورد فرمان **cd** فوق:

```
cd [-L|-P] [dir]
```

این نشانه‌گذاری می‌گوید که فرمان **cd** ممکن است با یکی از گزینه‌های انتخابی -L یا -P و بعلاوه شناسه انتخابی «dir» دنبال شود.

--help

بسیاری از برنامه های اجرایی از گزینه --help پشتبانی می‌کنند که شرحی از ترکیب دستوری حمایت شدهِ فرمان و گزینه‌هایش را نمایش می‌دهد. برای مثال:

```
[me@linuxbox me]$ mkdir --help

Usage: mkdir [OPTION] DIRECTORY...
Create the DIRECTORY(ies), if they do not already exist.

   -Z, --context=CONTEXT (SELinux) set security context to CONTEXT
Mandatory arguments to long options are mandatory for short options
too.
   -m, --mode=MODE   set file mode (as in chmod), not a=rwx - umask
   -p, --parents     no error if existing, make parent directories as
                     needed
   -v, --verbose     print a message for each created directory
   --help            display this help and exit
   --version         output version information and exit
```

بعضی برنامه‌ها گزینه --help را پشتیبانی نمی‌کنند، اما به هرحال آن را امتحان کنید. اغلب به یک پیغام خطایی منجر می‌گردد که مشابه اطلاعات کاربرد فرمان است.

man

اکثر برنامه‌های اجراییِ نامزد به کار رفتن در خط فرمان، یک بخشِ مستندات رسمی فراهم می‌کنند که یک ***manual*** یا ***صفحه man*** نامیده می‌شود. یک برنامه صفحه‌بندی خاص به نام **man** برای دیدن آنها به کار می‌رود. این برنامه به این شکل استفاده می‌شود:

```
man program
```

که در آن «program» نام فرمان مورد نظر برای دیدن مستندات آن است. قالب صفحه‌های Man تا اندازه‌ای تغییر می‌کند اما به طور کلی یک عنوان، یک خلاصه ترکیب دستوری فرمان، یک شرح در باره هدف فرمان، و فهرستی از گزینه‌های فرمان با شرحی در باره هر یک از آنها را شامل می‌گردد. به هر حال صفحه‌های Man، معمولاً شامل مثال نیستند، و به عنوان یک مرجع در نظر گرفته می‌شوند، نه راهنمای آموزشی. به عنوان یک

مثال، بیاید مشاهده صفحه man برای فرمان **ls** را امتحان کنیم:

```
[me@linuxbox me]$ man ls
```

در اکثر سیستم‌های لینوکس، برنامه **man** از **less** برای نمایش صفحه مستندات استفاده می‌کند، بنابراین تمام فرمانهای آشنای **less** در خلال نمایش صفحه کار می‌کنند.

README و سایر فایلهای مستندات

بسیاری از برنامه‌های نرم‌افزاری نصب شده در سیستم شما دارای فایلهای مستنداتِ مقیم در دایرکتوری /usr/share/doc هستند. بیشتر این مستندات در قالب متن ساده ذخیره می‌شوند و می‌توانند با برنامه **less** مشاهده بشوند. برخی از فایلها در قالب HTML هستند و می‌توانند با مرورگر وب شما دیده شوند. شما ممکن است با فایلهایی روبرو شوید که به پسوند .gz ختم می‌شوند. این پسوند نشان دهنده آن است که آنها با برنامه فشرده سازی **gzip** فشرده گردیده‌اند. بسته gzip یک نگارش خاص از برنامه **less** به نام **zless** را شامل می‌شود که محتویات فایلهای متن فشرده شده با gzip را نمایش می‌دهد.

« بسط | فهرست مطالب | کار با فرمانها »

تغییر مسیر ورودی- خروجی

در این درس، ما یک ویژگی قدرتمند استفاده شده با بسیاری از برنامه‌های خط فرمان به نام ***تغییر مسیر ورودی- خروجی*** رامرور می‌کنیم. به طوری که دیده‌ایم، بسیاری فرمانها از قبیل **ls** خروجی‌شان را در نمایشگر چاپ می‌کنند. هر چند این نباید مسئله‌ای باشد. با استفاده از برخی نشانه‌گذاری‌ها می‌توانیم خروجی بسیاری از فرمانها را به فایلها، دستگاه‌ها، و حتی ورودی سایر فرمانها ***تغییر مسیر*** بدهیم.

خروجی استاندارد

اکثر برنامه‌های خط فرمان که نتایج‌شان را نمایش می‌دهند، با فرستادن نتایج به وسیله‌ای به نام ***خروجی استاندارد*** این کار را انجام می‌دهند. به طور پیش‌فرض خروجی استاندارد محتویاتش را به نمایشگر هدایت می‌کند. برای تغییر مسیر خروجی استاندارد به یک فایل، کاراکتر «**>**» به این صورت به کار می‌رود:

```
[me@linuxbox me]$ ls > file_list.txt
```

در این مثال، فرمان **ls** اجرا می‌شود و نتایج در یک فایل به نام file_list.txt نوشته می‌شوند. چون خروجی **ls** به فایل تغییر مسیر یافته بود، نتایجی در نمایشگر ظاهر نمی‌گردد.

هر بار که فرمان فوق تکرار بشود، فایل file_list.txt از ابتدا با خروجی فرمان **ls** رونویسی می‌گردد (مترجم: در حقیقت ابتدا محتویات فایل پاک می‌شود و پس از آن مطالب جدید از ابتدای آن نوشته می‌شود یعنی به هر حال محتوای فایل از بین می‌رود حتی اگر خروجی جدیدی وجود نداشته باشد.). در صورتیکه می‌خواهید به جای آن نتایج جدید به فایل **پیوست** بشوند، «**>>**» را بکار ببرید، مانند این:

```
[me@linuxbox me]$ ls >> file_list.txt
```

وقتی نتایج پیوست می‌گردند، نتایج جدید به انتهای فایل افزوده می‌شوند، بدین ترتیب هر نوبت که فرمان تکرار می‌شود فایل را طویل‌تر می‌سازد. اگر موقعی که سعی می‌کنید خروجی تغییر مسیر یافته را به فایل پیوست کنید، فایل موجود نباشد، ایجاد خواهد شد.

ورودی استاندارد

بسیاری از فرمانها می‌توانند ورودی را از وسیله‌ای به نام ***ورودی استاندارد*** دریافت کنند. به طور پیش‌فرض، ورودی استاندارد محتویاتش را از صفحه کلید می‌گیرد، اما مانند خروجی استاندارد، می‌تواند تغییر مسیر داده شود. برای تغییر مسیر ورودی استاندارد از یک فایل به جای صفحه کلید، کاراکتر «**<**» به این صورت به کار می‌رود:

```
[me@linuxbox me]$ sort < file_list.txt
```

در مثال فوق، ما از فرمان **sort** برای پردازش محتویات فایل file_list.txt استفاده کردیم. نتایج در نمایشگر خارج می‌شوند چون خروجی استاندارد تغییر مسیر نیافته بود. می‌توانستیم خروجی استاندارد را به فایل دیگری تغییر مسیر بدهیم، مانند این:

```
[me@linuxbox me]$ sort < file_list.txt > sorted_file_list.txt
```

به طوری که می‌توانید ببینید، هر دو خروجی استاندارد و ورودی استاندارد یک فرمان می‌توانند تغییر مسیر داده شوند. آگاه باشید که ترتیب تغییر مسیر اهمیت ندارد. تنها التزام آن است که عملگرهای تغییر مسیر («**<**» و «**>**») باید بعد از سایر گزینه‌ها و شناسه‌ها در فرمان ظاهر شوند.

(مترجم: یعنی می‌توانستیم فرمان فوق را به این شکل نیز بنویسیم
(**sort > sorted_file_list.txt < file_list.txt**)

خط لوله‌ها

مفیدترین و قدرتمندترین موردی که می‌توانید با تغییر مسیر ورودی-خروجی برای اتصال چندین فرمان با یکدیگر انجام بدهید به وسیله چیزی است که ***خط لوله*** نامیده می‌شود. با خط لوله‌ها، خروجی استاندارد یک فرمان به ورودی یک فرمان دیگر تغذیه می‌شود. این هم موردی که کاملاً دلپسند من است:

```
[me@linuxbox me]$ ls -l | less
```

در این مثال، خروجی فرمان **ls** به برنامه **less** خورانده می‌شود. با استفاده از این شگردِ **"| less"**، شما می‌توانید خروجی هر فرمانی را قابل مرور کردن نمایید. من این روش را همیشه به کار می‌برم.

با متصل کردن فرمانها به یکدیگر، می‌توانیدکارهای برجسته شگفت‌انگیزی انجام بدهید. در اینجا چند مثال که شما برای امتحان کردن لازم دارید، آمده است:

مثالهای فرمانهای استفاده شده با یکدیگر در خط لوله‌ها

فرمان	آنچه انجام می‌دهد
`ls -lt | head`	نمایش 10 فایل جدیدتر در دایرکتوری جاری.
`du | sort -nr`	نمایش فهرستی از دایرکتوری‌های فرعی و اندازه فضای مصرفی آنها،به صورت مرتب شده از بزرگترین به کوچکترین فایل.
`find . -type f -print | wc -l`	نمایش تعداد کل فایلها در دایرکتوری کاری جاری و تمام دایرکتوری‌های فرعی آن.

فیلترها

یک نوع برنامه که بارها در خط لوله‌ها به کار می‌رود ***فیلتر*** نامیده می‌شود. فیلترها ورودی استاندارد را گرفته و عملیاتی روی آن انجام می‌دهند و نتایج را به خروجی استاندارد می‌فرستند. به این ترتیب، آنها می‌توانند در روشهای نیرومند برای پردازش اطلاعات ترکیب بشوند. این هم برخی برنامه‌های رایج که می‌توانند به عنوان فیلتر عمل کنند:

تعدادی از فرمانهای فیلتر که بیشتر متداول هستند

برنامه	کاری که انجام می‌دهد
`sort`	ورودی استاندارد را مرتب می‌کنند و سپس نتیجه مرتب شده را به خروجی استاندارد می‌فرستد.
`uniq`	از یک جریان مرتب شده داده‌های دریافت شده از ورودی استاندارد سطرهای تکراری داده‌ها را حذف می‌کند (یعنی اطمینان ایجاد می‌کند که هر یک از سطرها منحصر بفرد است).
`grep`	هر سطر داده را که از ورودی استاندارد دریافت می‌کند، معاینه کرده و هر سطری را که شامل الگوی تعیین شده‌ای از کاراکترها باشد خارج می‌کند.
`fmt`	متن را از ورودی استاندارد می‌خواند، سپس متن قالب‌بندی شده را درخروجی استاندارد بیرون می‌دهد.
`pr`	ورودی متن را از ورودی استاندارد دریافت می‌کند و داده‌ها را با نشانه‌های صفحه، سرآیند و پانویس به صفحه‌های آماده برای چاپ شدن، تفکیک می‌کند.
`head`	چند سطر ابتدای ورودی‌اش را بیرون می‌دهد. مناسب برای به دست آوردن سرآیند یک فایل.

tail	چند سطر انتهای ورودی‌اش را بیرون می‌دهد. مناسب برای مواردی مانند به دست آوردن آخرین ورودی‌های یک فایل ثبت رخداد.
tr	کاراکترها را ترجمه می‌کند. می‌تواند برای انجام وظایفی از قبیل تغییر و تبدیل‌های حالت کوچک و بزرگ حروف یا برگرداندن کاراکترهای خاتمه دهنده سطر از یک نوع به نوع دیگر (برای مثال، تبدیل فالهای متن DOS به فایلهای متن سَبکِ Unix) به کار برود.
sed	ویرایشگر جریانی. می‌تواند ترجمه‌های متن پیچیده‌تر از **tr** انجام بدهد.
awk	یک زبان برنامه‌نویسی کامل طراحی شده برای ساختن فیلترها. فوق‌العاده قدرتمند.

انجام وظایف با خط لوله‌ها

1. **چاپ کردن از خط فرمان.** لینوکس برنامه‌ای به نام **lpr** فراهم می‌کند که ورودی استاندارد را می‌پذیرد و آن را به چاپگر می‌فرستد. این برنامه اغلب با لوله‌ها و فیلترها به کار می‌رود. این هم یک جفت مثال:

```
cat poorly_formatted_report.txt | fmt | pr | lpr
cat unsorted_list_with_dupes.txt | sort | uniq | pr | lpr
```

در مثال نخست، ما از برنامه **cat** برای خواندن فایل و بیرون دادن خروجی استانداردش، که به ورودی استاندارد **fmt** لوله‌کشی شده است، استفاده می‌کنیم. **fmt** متن را به پاراگراف‌های مرتب قالب‌دهی می‌کند و در خروجی استانداردش بیرون می‌دهد، که به ورودی استاندارد برنامه **pr** لوله‌کشی گردیده. **pr** متن را به به صفحه‌های مرتب تفکیک نموده و در خروجی استاندارد که به برنامه **lpr** لوله‌کشی شده است بیرون می‌دهد. **lpr** ورودی استانداردش را می‌گیرد و آن را به چاپگر می‌فرستد.

مثال دوم با یک لیست نامرتب داده‌ها که دارای مدخل‌های تکراری است شروع می‌گردد. ابتدا، **cat** لیست را به **sort** می‌فرستد که آن را مرتب می‌کند و به **uniq** تغذیه می‌کند که موارد تکراری را حذف می‌نماید. بعد **pr** و **lpr** برای صفحه‌بندی و چاپ لیست استفاده می‌شوند.

2. **دیدن محتویات فایلهای tar**: غالباً نرم‌افزارهای توزیع شده را به صورت یک ***فایل gzipped tar*** مشاهده می‌کنید. این یک فایل بایگانی نوار سنتی شیوه یونیکس است (ایجاد شده با **tar**) که به وسیله **gzip** فشرده گردیده است. شما این فایلها را می‌توانید با پسوندهای قراردادی آنها شناسایی کنید، «.tar.gz» یا «.tgz». می‌توانید از فرمان زیر برای دیدن فهرست چنین فایلی در یک سیستم لینوکس استفاده کنید:

```
tar tzvf name_of_file.tar.gz | less
```

بسط

هر گاه شما یک سطرِ فرمان تایپ می‌کنید و کلید اینتر را فشار می‌دهید، bash قبل از انجام دادن فرمان شما چند پردازش روی آن اجرا می‌کند. ما در چند موقعیت دیده‌ایم که چگونه یک رشته کاراکتر ساده، به عنوان مثال «*»، می‌تواند معنای زیادی برای پوسته داشته باشد. پردازشی که باعث این رویداد می‌گردد، بسط نامیده می‌شود. به وسیله بسط، شما موردی را تایپ می‌کنید و آن مورد قبل از عمل کردن پوسته بر روی آن، به چیز دیگری توسعه داده می‌شود. برای نمایش دادن آنکه چه هدفی از این مطلب داریم، بیایید نگاهی به فرمان **echo** داشته باشیم. **echo** یک فرمان داخلی پوسته است که یک وظیفه بسیار ساده را اجرا می‌کند. شناسه‌های متنی‌اش را در خروجی استاندارد چاپ می‌کند:

```
[me@linuxbox me]$ echo this is a test
this is a test
```

تا حدی سر راست است. هر شناسه عبور داده شده به **echo** نمایش داده می‌شود. بیایید مثال دیگری را امتحان کنیم:

```
[me@linuxbox me]$ echo *
Desktop Documents ls-output.txt Music Pictures Public Templates Videos
```

پس دقیقاً چه اتفاقی رخ می‌دهد؟ چرا **echo** این «*» را چاپ نمی‌کند؟ به طوری که از کارمان با کاراکترهای عام به یاد دارید، کاراکتر «*» به معنی انطباق با همه کاراکترها در یک نام فایل است، اما آنچه ما در بحث نخستین‌مان ندیدیم آن بود که پوسته چگونه آن را انجام می‌دهد. پاسخ ساده آن است که پوسته «*» را قبل از اینکه فرمان **echo** اجرا بشود به چیز دیگری بسط می‌دهد (در این نمونه، به نام فایلهای دایرکتوری کاری فعلی). وقتی کلید اینتر فشرده می‌شود، پوسته به طور خودکار همه کاراکترهای واجد شرایط در خط فرمان را قبل از اینکه فرمان اجرا شود بسط می‌دهد، بنابراین فرمان **echo** اصلاً «*» را ندید، بلکه نتیجه بسط یافته را دید. با آگاهی از این مطلب، ما می‌توانیم مشاهده کنیم که **echo** همانطور که انتظار می‌رفت عمل نموده است.

بسط نام مسیر

ساز و کاری که کاراکترهای عام توسط آن عمل می‌کنند ***بسط نام مسیر*** نامیده می‌شود. اگر برخی روش‌هایی را که در درسهای قبل‌تر به کار گرفتیم، امتحان کنیم، خواهیم دید که آنها واقعاً بسط هستند. در یک دایرکتوری خانگی مفروض که به این شباهت دارد:

```
[me@linuxbox me]$ ls

Desktop
ls-output.txt
Documents Music
Pictures
Public
Templates
```

```
Videos
```

می‌توانستیم بسط‌های زیر را اجرا کنیم:

```
[me@linuxbox me]$ echo D*
Desktop Documents
```

و:

```
[me@linuxbox me]$ echo *s
Documents Pictures Templates Videos
```

یا حتی:

```
[me@linuxbox me]$ echo [[:upper:]]*
Desktop Documents Music Pictures Public Templates Videos
```

و نگریستن به ماورای دایرکتوری خانگی‌مان:

```
[me@linuxbox me]$ echo /usr/*/share
/usr/kerberos/share /usr/local/share
```

بسط مد

به طوری که ممکن است شما از معرفی فرمان **cd** به خاطر بیاورید، کاراکتر مد («~») معنای ویژه‌ای دارد. هنگامی که در ابتدای یک کلمه به کار برود، به نام دایرکتوری خانگی کاربر نامبرده، یا در صورتیکه کاربری ذکر نشده باشد، به دایرکتوری خانگی کاربر جاری، بسط داده می‌شود:

```
[me@linuxbox me]$ echo ~
/home/me
```

اگر کاربر foo یک حساب کاربری دارد، آنوقت:

```
[me@linuxbox me]$ echo ~foo
/home/foo
```

بسط حسابی

پوسته اجازه می‌دهد به وسیله بسط، محاسبه انجام بشود. این مورد به ما امکان می‌دهد اعلان پوسته را به عنوان یک ماشین حساب به کار ببریم:

```
[me@linuxbox me]$ echo $((2 + 2))
4
```

بسط حسابی این قالب را استفاده می‌کند:

```
$((expression))
```

که در آن «expression» یک عبارت حسابی متشکل از متغیرها و عملگرهای محاسباتی است.

بسط حسابی فقط اعداد صحیح (اعداد کامل، نه اعشاری) را پشتیبانی می‌کند، اما کاملاً می‌تواند عملیات متعدد مختلفی را انجام بدهد.

در بسط حسابی فاصله‌ها با اهمیت نیستند و عبارت می‌تواند تو در تو باشد. به عنوان مثال، برای پنج به توان دو ضربدر سه:

```
[me@linuxbox me]$ echo $(($((5**2)) * 3))
75
```

پرانتزهای منفرد می‌توانند برای گروه‌بندی عبارتهای فرعی به کار بروند. با این روش، ما می‌توانیم مثال فوق را بازنویسی کنیم و همان نتیجه را به جای دو بسط با کاربرد یک بسط به دست بیاوریم:

```
[me@linuxbox me]$ echo $(((5**2) * 3))
75
```

این هم یک مثال با استفاده از عملگرهای تقسیم و باقیمانده. به اثر تقسیم صحیح توجه کنید:

```
[me@linuxbox me]$ echo Five divided by two equals $((5/2))
Five divided by two equals 2
[me@linuxbox me]$ echo with $((5%2)) left over.
with 1 left over.
```

بسط ابرو

گویا عجیب‌ترین بسط، ***بسط ابرو*** نامیده می‌شود. با این بسط، شما می‌توانید از یک الگوی شامل ابروها چندین رشته متن خلق کنید. اکنون یک مثال:

```
[me@linuxbox me]$ echo Front-{A,B,C}-Back
Front-A-Back Front-B-Back Front-C-Back
```

الگوهایی که بسط ابرو داده می‌شوند ممکن است شامل یک بخش پیشتاز به نام ***preamble*** (***مقدمه***) و یک بخش دنباله به نام ***postscript***

(**پی‌نوشت**) باشند. بسط ابرو خودش می‌تواند لیستی از رشته‌های جدا شده با کاما، یا محدوده‌ای از اعداد صحیح یا کاراکترهای منفرد باشد. الگو ممکن است شامل فضاهای سفید تعبیه شده باشد. در اینجا مثالی از کاربرد محدوده اعداد صحیح آمده است:

```
[me@linuxbox me]$ echo Number_{1..5}
Number_1 Number_2 Number_3 Number_4 Number_5
```

یک محدوده از حروف با ترتیب وارونه:

```
[me@linuxbox me]$ echo {Z..A}
Z Y X W V U T S R Q P O N M L K J I H G F E D C B A
```

بسط ابرو می‌تواند تو در تو باشد:

```
[me@linuxbox me]$ echo a{A{1,2},B{3,4}}b
aA1b aA2b aB3b aB4b
```

برای چه کاری مناسب است؟ رایج‌ترین کاربرد آن برای ساختن فهرست‌هایی از فایلها یا دایرکتوری‌هایی است که باید ایجاد گردند. برای مثال، اگر شما یک عکاس بودید و مجموعه بزرگی از تصاویر داشتید که می‌خواستید آنها را نسبت به سال و ماه مرتب کنید، اولین کاری که شاید شما انجام بدهید ایجاد یک گروه دایرکتوری‌های نامگذاری شده در قالب «سال_ماه» است. در این روش، نام دایرکتوری‌ها به ترتیب تاریخی رخداد مرتب خواهند شد. شما می‌توانستید لیست کامل دایرکتوری‌ها را به تفصیل تایپ کنید، اما این کارِ خیلی زیادی است و متمایل به خطا نیز هست. به جای آن، شما می‌توانستید این مورد را انجام بدهید:

```
[me@linuxbox me]$ mkdir Photos
[me@linuxbox me]$ cd Photos
[me@linuxbox Photos]$ mkdir {2007..2009}-0{1..9} {2007..2009}-{10..12}
[me@linuxbox Photos]$ ls

2007-01 2007-07 2008-01 2008-07 2009-01 2009-07
2007-02 2007-08 2008-02 2008-08 2009-02 2009-08
2007-03 2007-09 2008-03 2008-09 2009-03 2009-09
2007-04 2007-10 2008-04 2008-10 2009-04 2009-10
2007-05 2007-11 2008-05 2008-11 2009-05 2009-11
2007-06 2007-12 2008-06 2008-12 2009-06 2009-12
```

نسبتاً ماهرانه!

بسط پارامتر

ما در این درس فقط می‌خواهیم به طور خلاصه ***بسط پارامتر*** را مطرح کنیم، اما در آینده بیشتر آن را پوشش خواهیم داد. این خصیصه‌ای است که بیشتر از آنکه به طور مستقیم در خط فرمان سودمند باشد، در اسکریپت‌های پوسته مفید است. بسیاری از امکانات آن در ارتباط با توانایی سیستم

برای ذخیره قطعه‌های کوچکی از داده‌ها و اختصاص یک نام به هر قطعه است. بسیاری از چنین قطعه‌هایی، که نام شایسته‌تر آنها **متغیر**است، برای بررسی شما در دسترس هستند. برای مثال، متغیری به نام «USER» شامل نام کاربری شما است. برای فراخوانی بسط پارامتر و آشکارسازی محتوای آن شما باید به این طریق عمل کنید:

```
[me@linuxbox me]$ echo $USER
me
```

برای دیدن فهرستی از متغیرهای در دسترس، این را امتحان کنید:

```
[me@linuxbox me]$ printenv | less
```

ممکن است شما متوجه شده‌اید که با انواع دیگر بسط، اگر یک الگو را اشتباه تایپ کنید، بسط صورت نخواهد گرفت و فرمان **echo** به سادگی الگوی تایپ شدهِ اشتباه را نمایش خواهد داد. با بسط پارامتر، در صورتیکه شما املای نام متغیر را اشتباه کنید، باز هم بسط انجام خواهد شد، اما به یک رشته تهی:

```
[me@linuxbox me]$ echo $SUER
[me@linuxbox ~]$
```

جایگزینی فرمان

جایگزینی فرمان به ما امکان می‌دهد خروجی یک فرمان را مانند یک بسط استفاده کنیم:

```
[me@linuxbox me]$ echo $(ls)
Desktop Documents ls-output.txt Music Pictures Public Templates Videos
```

یکی از موارد مورد پسند من چیزی مانند این است:

```
[me@linuxbox me]$ ls -l $(which cp)
-rwxr-xr-x 1 root root 71516 2007-12-05 08:58 /bin/cp
```

در اینجا ما نتایج **which cp** را به عنوان یک شناسه به فرمان **ls** عبور داده‌ایم، به موجب آن، فهرستی از برنامه **cp** را بدون دانستن نام مسیر کامل آن به دست می‌آوریم. ما فقط به فرمانهای ساده محدود نیستیم. خط لوله‌های کامل هم می‌توانند استفاده شوند (تنها بخشی از خروجی نشان داده شده):

```
[me@linuxbox me]$ file $(ls /usr/bin/* | grep bin/zip)

/usr/bin/bunzip2:
/usr/bin/zip:      ELF 32-bit LSB executable, Intel 80386, version 1
(SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.15, stripped
```

```
/usr/bin/zipcloak: ELF 32-bit LSB executable, Intel 80386, version 1
(SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.15, stripped
/usr/bin/zipgrep:  POSIX shell script text executable
/usr/bin/zipinfo:  ELF 32-bit LSB executable, Intel 80386, version 1
(SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.15, stripped
/usr/bin/zipnote:  ELF 32-bit LSB executable, Intel 80386, version 1
(SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.15, stripped
/usr/bin/zipsplit: ELF 32-bit LSB executable, Intel 80386, version 1
(SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.15, stripped
```

در این مثال، نتایج خط لوله لیست شناسه‌های فرمان **file** شده‌اند. یک ترکیب دستوری پیشنهادی در برنامه‌های پوسته قدیمی‌تر برای جایگزینی فرمان هست که در **bash** نیز پشتیبانی شده است. آن ترکیب به جای علامت دلار و پرانتزها از نقل‌قول‌های وارونه استفاده می‌کند:

```
[me@linuxbox me]$ls -l `which cp`
-rwxr-xr-x 1 root root 71516 2007-12-05 08:58 /bin/cp
```

نقل‌قول

اکنون که دیده‌ایم پوسته به روش‌های بسیاری می‌تواند بسط‌ها را انجام بدهد، وقت آن است که یاد بگیریم، ما چطور می‌توانیم آن را کنترل نماییم. این مثال را در نظر بگیرید:

```
[me@linuxbox me]$echo this is a     test
this is a test
```

یا:

```
[me@linuxbox me]$[me@linuxbox ~]$ echo The total is $100.00
The total is 00.00
```

در مثال نخست، تفکیک کلمه توسط پوسته فضاهای سفید اضافی در فهرست شناسه‌های فرمان **echo** را پاک کرده است. در مثال دوم، بسط پارامتر یک رشته تهی برای مقدار $1 جایگزین نمود زیرا یک متغیر تعریف نشده بود. پوسته برای موقوف کردن انتخابی بسط‌های ناخواسته ساز و کاری به نام نقل‌قول فراهم می‌کند.

نقل‌قول‌های دوگانه

اولین نوع نقل‌قولی که به آن خواهیم پرداخت نقل‌قول دوگانه است، تمام کاراکترهای خاص مورد استفاده پوسته داخل نقل‌قول دوگانه معنای ویژه خود را از دست می‌دهند و به عنوان کاراکترهای معمولی با آنها رفتار می‌گردد. موارد استثنا عبارتند از $، \ و ` (نقل‌قول برعکس). این بدان معنی است که تفکیک کلمه، بسط نام مسیر، بسط مد، و بسط ابرو خاموش می‌شوند، اما بسط پارامتر، بسط حسابی، و جایگزینی فرمان بازهم اجرا می‌گردند. با استفاده از نقل‌قول‌های دوگانه، ما می‌توانیم نام‌فایلهای شامل فاصله‌های تعبیه شده را حریف بشویم. فرض کنیم شما قربانی بداقبالِ فایلی

به نام two words.txt بودید. اگر شما سعی در استفاده از این فایل در خط فرمان می‌نمودید، تفکیک کلمه باعث می‌گردید با این نام به جای یک شناسه منفردِ مطلوب، به عنوان دو شناسه جداگانه رفتار بشود:

```
[me@linuxbox me]$ ls -l two words.txt

ls: cannot access two: No such file or directory
ls: cannot access words.txt: No such file or directory
```

با کاربرد نقل‌قول‌های دوگانه، شما می‌توانید تفکیک کلمه را متوقف کنید و نتیجه دلخواه را به دست آورید، بعلاوه شما حتی می‌توانید عیب را برطرف نمایید:

```
[me@linuxbox me]$ ls -l "two words.txt"
-rw-rw-r-- 1 me me 18 2008-02-20 13:03 two words.txt
[me@linuxbox me]$ mv "two words.txt" two_words.txt
```

در اینجا، اکنون ما نباید به تایپ کردن آن نقل‌قول‌های ناخوشایند ادامه بدهیم. به خاطر داشته باشید داخل نقل‌قول‌های دوگانه، بسط پارامتر، بسط حسابی، و جایگزینی فرمان بازهم رخ می‌دهند:

```
[me@linuxbox me]$ echo "$USER $((2+2)) $(cal)"

me 4
February 2008
Su Mo Tu We Th Fr Sa
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29
```

ما باید زمان کوتاهی را به تاثیر نقل‌قول‌های دوگانه بر جایگزینی فرمان بپردازیم. ابتدا بیایید کمی عمیق‌تر نگاه کنیم که تفکیک کلمه چگونه کار می‌کند. در مثال قدیمی‌تر، ما دیدیم در زدودن فاصله‌های اضافه از متن، تفکیک کلمه چگونه ظاهر می‌شود:

```
[me@linuxbox me]$ echo this is a     test
this is a test
```

به طور پیش‌فرض، تفکیک کلمه حضور فاصله‌ها، tabها، و سطرهای جدید (کاراکترهای تعویض سطر) را جستجو می‌کند و با آنها به عنوان جداکننده‌های میان کلمات رفتار می‌نماید. این به معنای آن است که فاصله‌ها، tabها، و سطرهای جدیدِ نقل‌قولی نشده به عنوان بخشی از متن به حساب نمی‌آیند. آنها فقط به عنوان جداکننده‌ها خدمت می‌کنند. چون آنها کلمات را به شناسه‌های متمایز جدا می‌کنند، سطر فرمان مثال ما شامل یک فرمانِ دنبال شده با چهار شناسهِ قابل تشخیص است. در صورتیکه ما نقل‌قول‌های دوگانه را اضافه کنیم:

```
[me@linuxbox me]$ echo "this is a     test"
this is a     test
```

تفکیک کلمه منکوب می‌گردد و فاصله‌های گنجانده شده به عنوان جدا کننده‌ها به حساب نمی‌آیند، بلکه آنها بخشی از شناسه می‌شوند. وقتی نقل‌قول‌های دوگانه افزوده می‌شوند، سطر فرمان ما شامل یک فرمانِ دنبال شده با یک شناسه منفرد است. اینکه سطرهای جدید توسط ساز و کار تفکیک کلمه به عنوان جداکننده به شمار می‌روند باعث یک تاثیر جالب توجه، اگرچه فریبنده، بر جایگزینی فرمان می‌گردد. مورد زیر را ملاحظه کنید:

```
[me@linuxbox me]$ echo $(cal)
February 2008 Su Mo Tu We Th Fr Sa 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
19 20 21 22 23 24 25 26 27 28 29
[me@linuxbox me]$ echo "$(cal)"
February 2008
Su Mo Tu We Th Fr Sa
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29
```

در نمونه نخست، جایگزینی فرمان نقل‌قولی نشده به یک سطر فرمانِ سی و هشت شناسه‌ای منجر می‌شود. در دومی، یک سطر فرمان با یک شناسه که شامل فاصله‌ها و سطرهای جدید جاسازی شده است.

نقل‌قول‌های منفرد

اگر شما احتیاج داشته باشید تمام بسط‌ها را خاموش کنید، نقل‌قول‌های منفرد را استفاده می‌کنید. این هم مقایسه‌ای بین نقل‌قول نشده، نقل‌قول‌های دوگانه، و نقل‌قول‌های منفرد:

```
[me@linuxbox me]$ echo text ~/*.txt {a,b} $(echo foo) $((2+2)) $USER
text /home/me/ls-output.txt a b foo 4 me
[me@linuxbox me]$ echo "text ~/*.txt {a,b} $(echo foo) $((2+2)) $USER"
text ~/*.txt {a,b} foo 4 me
[me@linuxbox me]$ echo 'text ~/*.txt {a,b} $(echo foo) $((2+2)) $USER'
text ~/*.txt {a,b} $(echo foo) $((2+2)) $USER
```

به طوریکه می‌توانید مشاهده کنید، با هر مرحله بعدیِ نقل‌قول کردن، بسط‌های بیشتر و بیشتری خاموش می‌گردند.

کاراکترهای معاف کردن (Escaping)

گاهی اوقات شما می‌خواهید تنها یک کاراکتر منفرد را نقل‌قول کنید. برای انجام این کار، می‌توانید یک \ جلوی کاراکتر قرار بدهید، که در این بافت ***کاراکتر گریز*** نامیده می‌شود. این مورد اغلب در داخل نقل‌قول‌های دوگانه برای پیش‌گیری گزینشی از یک بسط به کار می‌رود:

```
[me@linuxbox me]$ echo "The balance for user $USER is: \$5.00"
The balance for user me is: $5.00
```

استفاده از معاف کردن برای زدودن معنای ویژه یک کاراکتر در نام یک فایل نیز متداول است. برای مثال، استفاده از کاراکترهایی در نام فایل که به طور معمول برای پوسته دارای معنای خاص هستند، امکان‌پذیر است. اینها می‌توانند شامل $، !، &، و دیگران باشند. برای قرار دادن یک کاراکتر خاص در نام یک فایل می‌توانید اینطور عمل کنید:

```
[me@linuxbox me]$ mv bad\&filename good_filename
```

برای اجازه دادن به حضور یک کاراکتر \ با تایپ کردن به این شکل \\ آن را معاف کنید(از معنای خاص). توجه نمایید که داخل نقل‌قول منفرد \ معنای ویژه را از دست می‌دهد و به عنوان یک کاراکتر معمولی عمل می‌کند.

شگردهای بیشتر با Backslash

اگر به صفحه‌های **man** برای هر برنامه نوشته شده توسط پروژه گنو نگاه کنید، متوجه خواهید گردید که علاوه بر گزینه‌های خط فرمانِ شاملِ یک خط تیره و یک کاراکتر، نام‌های گزینه بلند نیز وجود دارند که با دو خط تیره شروع می‌شوند. برای مثال،موارد زیر مترادف هستند:

```
ls -r
ls --reverse
```

چرا آنها هر دو پشتیبانی می‌شوند؟ شکل کوتاه برای تایپ کننده‌های کُند در خط فرمان است و شکل بلند بیشتر برای اسکریپت‌ها، اگر چه برخی گزینه‌ها ممکن است فقط به شکل بلند باشند. من گاهی اوقات گزینه‌های مبهم به کار می‌برم، و در صورتیکه باید بعد از ماه‌ها اسکریپتی را که نوشته‌ام بازبینی کنم شکل بلند را مفید می‌یابم. دیدن گزینه بلند کمک می‌کند ملتفت شوم گزینه چه کار می‌کند، رهایی بخشیدن من از یک مراجعه به صفحه **man**. اندکی تایپ بیشتر در حال، مقدار زیادی کار کمتر در آینده. تنبلی پشتیبانی می‌شود.

همچنانکه شاید شما حدس بزنید، استفاده از شکل بلند گزینه‌ها می‌تواند یک سطر فرمان منفرد را خیلی طولانی کند. برای مبارزه با این مشکل، می‌توانید از یک \ جهت صرفنظر کردن پوسته از کاراکتر تعویض سطر، استفاده کنید، مانند این:

```
ls -l \
   --reverse \
   --human-readable \
   --full-time
```

کاربرد backslash به این روش به ما امکان می‌دهد سطرهای جدید را در فرمان خود درج کنیم. توجه نمایید که برای آنکه این ترفند کار کند، سطرجدید باید به طور بدون فاصله پس از backslash تایپ شود. اگر شما یک فاصله پس از backslash قرار بدهید، کاراکتر فاصله صرفنظر خواهد شد، نه سطرجدید. Backslashها برای درج کاراکترهای خاص داخل متن ما نیز به کار می‌روند. اینها ***کاراکترهای گریزِ backslash*** نامیده می‌شوند. این هم برخی از موارد رایج:

کاراکتر گریز	نام	استفاده‌های امکان‌پذیر
\n	سطرجدید	افزودن سطرهای خالی به متن
\t	tab	درج tabهای افقی به متن
\a	هشدار	ایجاد صدای بوق کوتاه ترمینال
\\	backslash	یک \ درج می‌کند
\f	تغذیه کاغذ	فرستادن این کاراکتر به چاپگرِ شما صفحه رابیرون می‌دهد

استفاده از این کاراکترهای گریزِ backslash بسیار رایج است. این اندیشه ابتدا در زبان برنامه‌نویسی C ظاهر گردید. امروزه، شل، C++، پرل، پایتون، awk، tcl، و دیگر زبانهای برنامه‌نویسی این تدبیر را به کار می‌برند. کاربرد فرمان **echo** با گزینه -e نمایش دادن آنها را برای ما امکان‌پذیر می‌کند:

```
[me@linuxbox me]$ echo -e "Inserting several blank lines\n\n\n"
Inserting several blank lines



[me@linuxbox me]$ echo -e "Words\tseparated\tby\thorizontal\ttabs."
Words separated   by  horizontal  tabs
[me@linuxbox me]$ echo -e "\aMy computer went \"beep\"."

My computer went "beep".

[me@linuxbox me]$ echo -e "DEL C:\\WIN2K\\LEGACY_OS.EXE"

DEL C:\WIN2K\LEGACY_OS.EXE
```

مجوزها

سیستم‌های یونیکس_مانند، از قبیل لینوکس با سایر سیستم‌های کامپیوتری تفاوت دارند برای آن که نه فقط ***چند وظیفه‌ای*** بلکه ***چند کاربری*** نیز هستند.

این دقیقاً به چه معنی است؟ به معنای آن است که در یک زمان بیش از یک کاربر می‌تواند مشغول کار با کامپیوتر باشد. در حالیکه کامپیوتر شما فقط یک صفحه کلید و یک نمایشگر دارد، باز هم می‌تواند توسط بیش از یک کاربر استفاده بشود. برای مثال، اگر کامپیوتر شما به شبکه یا اینترنت متصل باشد، کاربران راه دور می‌توانند از طریق **ssh** (پوسته امن) به سیستم داخل شده با کامپیوتر کار کنند. به راستی، کاربران راه دور می‌توانند برنامه‌های کاربردی گرافیکی را اجرا کنند و خروجی نمایش یافته روی یک کامپیوتر راه دور را داشته باشند. سیستم پنجره X این را پشتیبانی می‌کند.

توانایی چند کاربری سیستم‌های یونیکس_مانند، خصیصه‌ای است که عمیقاً ریشه در طراحی سیستم عامل دارد. اگر شما محیطی را که یونیکس در آن ایجاد گردیده بود به خاطر بیاورید، این کاملاً قابل درک می‌گردد. سالها قبل، پیش از اینکه کامپیوترها شخصی باشند، آنها بسیار بزرگ، گران، و متمرکز بودند. به طور نمونه یک سیستم کامپیوتر دانشگاه متشکل شده بود ازیک کامپیوتر پردازنده مرکزی مستقر در برخی ساختمان‌های دانشگاه و ***ترمینالهایی*** که در سراسر دانشگاه مستقر شده بودند، و هر یک به کامپیوتر بزرگ مرکزی متصل بودند. کامپیوتر باید در یک زمان کاربران بسیاری را پشتیبانی می‌کرد.

برای عملی ساختن این مورد، باید روشی برای محافظت کاربران از یکدیگر ابداع می‌گردید. گذشته از این، شما نه می‌توانستید فعالیت یک کاربر برای خرابی کامپیوتر را اجازه بدهید، نه می‌توانستید دستکاری کردن یک کاربر در فایلهای متعلق به یک کاربر دیگر را اجازه بدهید.

این درس فرمانهای زیر را پوشش خواهد داد:

- **chmod** ـ ویرایش حقوق دسترسی فایل
- **su** ـ به طور موقت کاربر ارشد شدن
- **sudo** ـ به طور موقت از حقوق کاربر ارشد برخوردار شدن
- **chown** ـ تعویض مالکیت فایل
- **chgrp** ـ تعویض مالکیت گروه فایل

مجوزهای فایل

در یک سیستم لینوکس، برای هر فایل و دایرکتوری حقوق دسترسی مالک فایل، اعضای گروه کاربران وابسته، و هر شخص دیگر تعیین می‌گردد. حقوقی که برای خواندن یک فایل، نوشتن فایل، و اجرای فایل (یعنی اجرا کردن فایل به عنوان یک برنامه) می‌توانند تعیین بشوند.

برای دیدن مجوزهای تنظیم شده برای یک فایل، می‌توانیم فرمان **ls** را به کار ببریم. به عنوان یک مثال، به **bash** نگاه خواهیم نمود، برنامه‌ای که در دایرکتوری **/bin** قرار دارد:

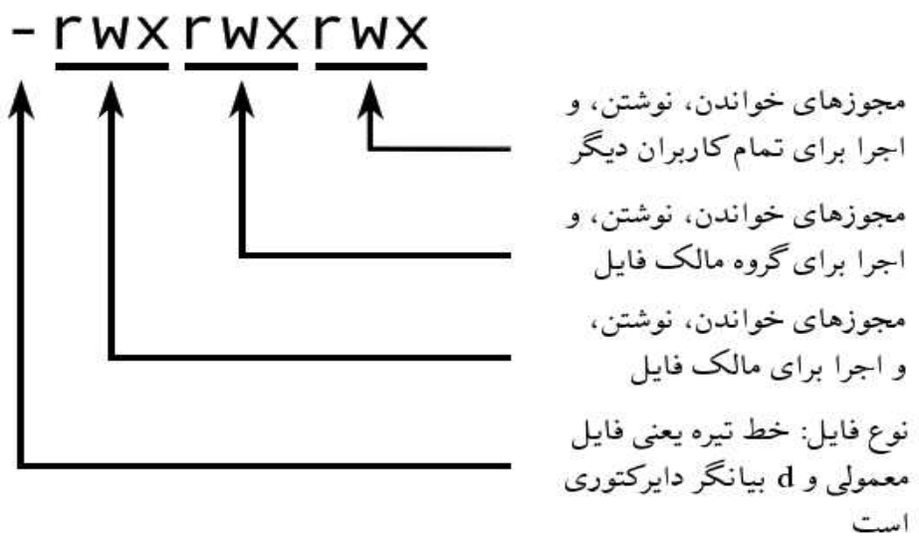
```
[me@linuxbox me]$ ls -l /bin/bash

-rwxr-xr-x 1 root root  316848 Feb 27  2000 /bin/bash
```

در اینجا می‌توانیم مشاهده کنیم:

- فایل /bin/bash در مالکیت کاربر root است
- کاربر ارشد دارای حق خواند، نوشتن، و اجرای این فایل است
- فایل متعلق به گروه root است
- اعضای گروه root نیز می‌توانند این فایل را بخوانند و اجرا کنند
- هر شخص دیگری می‌تواند این فایل را بخواند و اجرا نماید

در نمایش ترسیمی پایین، مشاهده می‌کنیم که جزء نخست چطور تفسیر می‌گردد. این جزء متشکل از یک کاراکترِ نشان دهنده نوع فایل است، که با مجموعه کاراکترهای سه‌تایی دنبال می‌شود که مجوز خواندن، نوشتن، و اجرای فایل برای مالک، گروه، و سایرین را بیان می‌کنند.



chmod

فرمان **chmod** برای تعویض مجوزهای یک فایل یا دایرکتوری به کار می‌رود. برای انجام این کار، شما تنظیم مجوزهای مورد نظر و فایل یا فایلهایی را که مایل به ویرایش هستید مشخص می‌کنید. دو روش برای مشخص نمودن مجوزها وجود دارد. در این درس ما بر یکی از آنها که روش ***نشانه‌گذاری اُکتال*** نامیده می‌شود، تمرکز خواهیم کرد.

تصور نمودن تنظیم مجوزها به عنوان گروه‌هایی از بیت‌ها (روشی است که کامپیوتر در باره آنها فکر می‌کند) آسان است. این هم چگونگی کار کرد آن است:

```
rwx rwx rwx = 111 111 111
rw- rw- rw- = 110 110 110
rwx --- --- = 111 000 000
```

```
و به همین ترتیب ...

rwx = 111 in binary = 7
rw- = 110 in binary = 6
r-x = 101 in binary = 5
r-- = 100 in binary = 4
```

اکنون، اگر هر مجموعه سه‌تایی مجوزها (مالک، گروه، و سایرین) را به صورت یک رقم منفرد نمایش بدهید، شما دارای یک روش نسبتاً راحت بیان کردن تنظیمات مجوزها هستید. برای مثال، اگر می‌خواستیم `some_file` را برای داشتن مجوز خواندن و نوشتنِ مالک تنظیم کنیم، اما می‌خواستیم فایل را از سایرین پوشیده نگاه داریم، به این صورت می‌نوشتیم:

```
[me@linuxbox me]$ chmod 600 some_file
```

در اینجا جدولی از اعدادی که تمام تنظیم‌های رایج را پوشش می‌دهد آمده است. آنها که با «7» شروع می‌شوند برای برنامه‌ها به کار می‌روند (چون اجرا را فعال می‌کنند) و بقیه برای سایر انواع فایل هستند.

مقدار	معنی
777	**(rwxrwxrwx)** بدون هرگونه محدودیت مجوز. هر شخصی می‌تواند هر کاری انجام بدهد. به طور کلی تنظیم مطلوبی نیست.
755	**(rwxr-xr-x)** مالک فایل می‌تواند فایل را بخواند، بنویسد، و اجرا کند. تمام افراد دیگر می‌توانند فایل را اجرا کنند. این تنظیم برای برنامه‌هایی که توسط تمام کاربران استفاده می‌شوند رایج است.
700	**(rwx------)** مالک فایل می‌تواند فایل را بخواند، بنویسد، و اجرا نماید. هیچ فرد دیگری هیچ حقی ندارد. این تنظیم برای برنامه‌هایی سودمند است که تنها مالک می‌تواند به کار ببرد و باید پوشیده از سایرین نگهداری بشوند.
666	**(rw-rw-rw-)** تمام کاربران می‌توانند فایل را بخوانند و بنویسند.
644	**(rw-r--r--)** مالک می‌تواند فایل را بخواند و بنویسد، درحالیکه تمام کاربران دیگر فقط می‌توانند فایل را بخوانند. تنظیم رایج برای فایلهای داده‌ها که هر شخصی بنواند بخواند، اما فقط مالک بتواند تغییر بدهد.
600	**(rw-------)** مالک می‌تواند فایل را بخواند و بنویسد. تمام اشخاص دیگر فاقد حقوق هستند. یک تنظیم رایج برای فایلهای داده‌ها که مالک می‌خواهد محرمانه نگاه دارد.

مجوزهای دایرکتوری

فرمان **chmod** برای کنترل مجوزهای دسترسی به دایرکتوری‌ها نیز می‌تواند به کار برود. دوباره، ما می‌توانیم از نشانه‌گذاری اُکتال برای تنظیم مجوزها استفاده کنیم، اما معنی صفات r، w، و x متفاوت است:

- **r** ـ درصورتیکه صفت x نیز تنظیم باشد، اجازه می‌دهد محتویات دایرکتوری لیست بشود.
- **w** ـ در صورتیکه صفت x نیز تنظیم باشد اجازه می‌دهد فایلها در داخل دایرکتوری ایجاد بشوند، حذف گردند، یا تغییر نام داده شوند.
- **x** ـ وارد شدن به دایرکتوری (یعنی **cd dir**) را مجاز می‌سازد.

این هم برخی تنظیمات متداول برای دایرکتوری‌ها:

مقدار	معنی
777	**(rwxrwxrwx)** بدون محدودیت مجوز. هر شخص می‌تواند فایلها را لیست کند، فایل‌های جدید در دایرکتوری ایجاد نماید و فایلها را از دایرکتوری حذف کند. به طور کلی تنظیم مناسبی نیست.
755	**(rwxr-xr-x)** مالک دایرکتوری دسترسی کامل دارد. تمام افراد دیگر می‌توانند فهرست فایلها را ببینند، اما نه می‌توانند فایل ایجاد کنند نه حذف کنند. این تنظیم برای دایرکتوری‌هایی که شما مایل هستید با کاربران دیگر به اشتراک بگذارید متداول است.
700	**(rwx------)** مالک دایرکتوری دسترسی کامل دارد. هیچ شخص دیگری حقوقی ندارد. این تنظیم برای دایرکتوری‌هایی که فقط مالک می‌تواند استفاده کند و باید از سایرین پوشیده نگاه داشته شود مناسب است.

کاربر ارشد شدن برای یک مدت کوتاه

کاربر ارشد شدن، اغلب برای انجام وظایف مدیریتی مهم سیستم لازم است، اما به طوریکه به شما اخطار شده است، شما نباید به عنوان کاربر ارشد متصل به سیستم بمانید. در اکثر توزیع‌ها، برنامه‌ای وجود دارد که می‌تواند دسترسی به مزایای کاربر ارشد را به طور موقت به شما بدهد. این برنامه **su** نامیده می‌شود (short for **s**ubstitute **u**ser) و در چنان موقعیت‌هایی که نیاز است شما برای انجام وظایف کوچکی کاربر ارشد باشید می‌تواند به کار برود. برای کاربر ارشد شدن، به سادگی فرمان **su** را تایپ کنید. آمادگی پذیرفتن کلمه عبور کاربر ارشد به شما اعلام می‌شود:

```
[me@linuxbox me]$ su
Password:
[root@linuxbox me]#
```

پس از اجرای فرمان **su**، شما یک نشست پوسته جدید به عنوان کاربر ارشد خواهید داشت. برای خروج از نشست کاربر ارشد، **exit** را تایپ کنید و شما به نشست قبلی‌تان باز خواهید گشت.

در برخی توزیع‌ها، به طور برجسته در Ubuntu، یک روش جایگزین به کار می‌رود. به جای استفاده از **su**، این سیستم‌ها فرمان **sudo** را به کار می‌گیرند. با فرمان **sudo**، بر اساس نیاز، به یک یا چند کاربر مزایای کاربر ارشد واگذار می‌شود. برای اجرای یک فرمان به عنوان کاربر ارشد، به سادگی فرمان **sudo** بر فرمان مورد نظر مقدم می‌گردد. بعد از اینکه فرمان وارد شد، اعلان ورود کلمه عبور کاربر به جای کاربر ارشد نمایش داده می‌شود:

```
[me@linuxbox me]$ sudo some_command
Password:
[me@linuxbox me]$
```

تغییر مالکیت فایل

شما می‌توانید مالک یک فایل را به وسیله فرمان **chown** تغییر بدهید. اکنون یک مثال: فرض کنید من می‌خواستم مالک some_file را از me به you تغییر بدهم. به این شکل می‌توانستم:

```
[me@linuxbox me]$ su
Password:
[root@linuxbox me]# chown you some_file
[root@linuxbox me]# exit
[me@linuxbox me]$
```

توجه کنید که به منظور تغییر مالک یک فایل، شما باید کاربر ارشد باشید. برای انجام این کار، مثال ما فرمان **su** را به کار گرفت، سپس **chown** را اجرا کردیم، و سرانجام برای برگشت به نشست قبلی **exit** را تایپ نمودیم.

chown برای دایرکتوری‌ها به همان طریق کار می‌کند که در مورد فایلها عمل می‌کند.

تغییر مالکیت گروه

مالکیت گروه یک فایل یا دایرکتوری می‌تواند با **chgrp** تغییر داده شود. این فرمان به این شکل به کار می‌رود:

```
[me@linuxbox me]$ chgrp new_group some_file
```

در مثال بالا، ما مالکیت گروه فایل some_file را از گروه قبلی‌اش به new_group تغییر دادیم. برای انجام یک **chgrp** شما باید مالک فایل یا دایرکتوری باشید.

کنترل Job

در درس قبل، ما به برخی ضرورت‌های جانبیِ چند کاربری بودن لینوکس نگاه کردیم. در این درس، ماهیت چند وظیفه‌ای لینوکس و چگونگی اداره کردن آن با رابط خط فرمان را رسیدگی می‌کنیم.

مانند هر سیستم عامل چند وظیفه‌ای، لینوکس چندین پردازش همزمان را اجرا می‌کند. خُب، به هر حال آنها به ظاهر همزمان هستند. در عمل، یک پردازشگر منفرد کامپیوتر تنها می‌تواند یک پردازش را در یک زمان اجرا کند اما هسته لینوکس نوبت‌دهی به هر پردازش در پردازشگر را مدیریت می‌کند و به نظر می‌آید که در یک زمان در حال اجرا شدن هستند.

چند فرمان وجود دارد که می‌توانند برای کنترل پردازشها به کار بروند. آنها عبارتند از:

- **ps** ـ فهرست نمودن پردازشهای در حال اجرا در سیستم
- **kill** ـ فرستادن یک سیگنال به یک یا چند پردازش (معمولاً برای کشتن–kill– پردازش)
- **jobs** ـ یک روش جایگزین برای لیست کردن پردازش‌های خودتان
- **bg** ـ قرار دادن یک پردازش در پس‌زمینه
- **fg** ـ قرار دادن یک پردازش در پیش‌زمینه

یک مثال کاربردی

در حالیکه ممکن است به نظر بیاید این مطلب نسبتاً مبهم است، برای کاربران متوسطی که بیشتر با رابط گرافیکی کاربر کار می‌کنند، می‌تواند خیلی کاربردی باشد. شما شاید این را ندانید، اما اکثر (اگر نه همه)برنامه‌های گرافیکی می‌توانند از طریق خط فرمان راه‌اندازی بشوند. این هم یک مثال: یک برنامه کوچک فراهم شده با سیستم پنجره X به نام **xload** وجود دارد که نموداری به نمایندگی از بارگیری سیستم نمایش می‌دهد. شما می‌توانید این برنامه را با تایپ به شرح زیر اجرا کنید:

```
[me@linuxbox me]$ xload
```

توجه نمایید که پنجره کوچک **xload** ظاهر می‌گردد و نمایش نمودار بارگیری سیستم را شروع می‌کند. همچنین توجه داشته باشید که اعلان فرمان شما بعد از راه‌اندازی برنامه دوباره ظاهر نگردید. پوسته قبل از باز گرداندن کنترل به شما، در انتظار خاتمه یافتن برنامه می‌ماند. اگر شما پنجره **xload** را ببندید، برنامه **xload** خاتمه یافته و اعلان فرمان باز می‌گردد.

قرار دادن یک برنامه در پس‌زمینه

اکنون به منظور اندکی آسان‌تر نمودن مطلب، می‌خواهیم دوباره برنامه **xload** را راه‌اندازی کنیم، اما این دفعه آن را در پس‌زمینه قرار خواهیم داد به طوری که اعلان فرمان برگشت داده شود. برای انجام این کار، ما xload را به این شکل اجرا می‌کنیم:

```
[me@linuxbox me]$ xload &
[1] 1223

[me@linuxbox me]$
```

در این حالت، اعلان فرمان برگشت می‌شود زیرا پردازش در پس‌زمینه قرار داده شد.

اکنون تصور کنید شما استفاده از علامت & برای قرار دادن برنامه در پس‌زمینه را فراموش کردید. باز هم امید هست. شما می‌توانید Ctrl-z را تایپ کنید و برنامه به حالت تعلیق در خواهد آمد. پردازش هنوز وجود دارد، امابدون استفاده است. برای ادامه یافتن پردازش در پس‌زمینه، فرمان **bg** را تایپ کنید (کوتاه شده background). این هم یک مثال:

```
[me@linuxbox me]$ xload
[2]+ Stopped xload

[me@linuxbox me]$ bg
[2]+ xload &
```

فهرست پردازش‌های شما

حال که ما یک پردازش در پس‌زمینه داریم، نمایش دادن فهرستی از پردازش‌هایی که ما راه‌اندازی کرده‌ایم مفید است. برای انجام این مورد، می‌توانیم یا از فرمان **jobs** یا از فرمان قدرتمندتر **ps** استفاده کنیم.

```
[me@linuxbox me]$ jobs
[1]+ Running xload &

[me@linuxbox me]$ ps
PID TTY TIME CMD
1211 pts/4 00:00:00 bash
1246 pts/4 00:00:00 xload
1247 pts/4 00:00:00 ps

[me@linuxbox me]$
```

کشتن یک پردازش

فرض کنید که شما یک برنامه دارید که بی توجه می‌شود، چگونه از دست آن خلاص می‌شوید؟ البته شما فرمان **kill** را به کار می‌برید. اجازه بدهید اثر آن روی xload را امتحان کنیم. ابتدا، نیاز دارید برنامه‌ای را که می‌خواهید بکُشید، شناسایی کنید. شما می‌توانید یا **jobs** یا **ps** را برای انجام این کار استفاده کنید. اگر **jobs** را به کار ببرید یک شماره job به دست می‌آورید. با فرمان **ps**، یک شماره شناسایی پردازش (PID) به شما داده می‌شود. ما با هر دو روش آن را انجام خواهیم داد:

```
[me@linuxbox me]$ xload &
[1] 1292

[me@linuxbox me]$ jobs
[1]+ Running xload &

[me@linuxbox me]$ kill %1
```

```
[me@linuxbox me]$ xload &
[2] 1293
[1] Terminated xload

[me@linuxbox me]$ ps
PID TTY TIME CMD
1280 pts/5 00:00:00 bash
1293 pts/5 00:00:00 xload
1294 pts/5 00:00:00 ps

[me@linuxbox me]$ kill 1293
[2]+ Terminated xload

[me@linuxbox me]$
```

کمی بیشتر در باره kill

در حالیکه فرمان **kill** برای کشتن پردازش‌ها به کار می‌رود، هدف واقعی آن فرستادن ***سیگنال‌ها*** به پردازش‌ها است. اکثر اوقات سیگنال برای اینکه به پردازش بگوید کنار برود درنظر گرفته می‌شود، اما مواردی بیش از این برایش وجود دارد. برنامه‌ها (در صورتیکه به طور صحیح نوشته شده باشند) بیشتر مواقع به منظور میسر کردن روش کم و بیش برازنده‌ای برای خاتمه یافتن، به سیگنال‌های سیستم عامل گوش می‌کنند و به آنها پاسخ می‌دهند. برای مثال، یک ویرایشگر متن می‌تواند برای هر سیگنالی که بیانگر قطع ارتباط کاربر، یا شروع فرآیند خاموش شدن سیستم است، گوش بسپارد. موقعی که این سیگنال را دریافت می‌کند، قبل از اینکه خارج شود، کار در حال پردازش را ذخیره می‌کند. فرمان **kill** می‌تواند سیگنال‌های متنوعی به پردازش‌ها بفرستد. با تایپ:

```
kill -l
```

فهرستی از سیگنال‌هایی که پشتبانی می‌شوند دریافت خواهید نمود. بیشتر آنها نسبتاً مبهم هستند، اما شناختن برخی از آنها مفید است:

شماره سیگنال	نام	توضیحات
1	SIGHUP	سیگنال توقف. برنامه‌ها می‌توانند به این سیگنال گوش بسپارند و به محض شنیدن آن عمل کنند. این سیگنال موقعی که شما ترمینال را می‌بندید برای پردازشهای در حال اجرای ترمینال فرستاده می‌شود.
2	SIGINT	سیگنال وقفه. این سیگنال برای قطع کردن پردازشها به آنها داده می‌شود. برنامه‌ها می‌توانند این سیگنال را تحویل گرفته و براساس آن عمل کنند. همچنین شما می‌توانید این سیگنال را به طور مستقیم با تایپ Ctrl-c در پنجره ترمینال، جایی که برنامه در حال اجراست صادر کنید.

15	SIGTERM	سیگنال خاتمه. این سیگنال برای خاتمه دادن پردازشها به آنها داده می‌شود. دوباره، برنامه‌ها می‌توانند این سیگنال را تحویل گرفته بر اساس آن عمل کنند. این سیگنال پیش‌فرضی است که اگر سیگنالی تعیین نگردیده باشد توسط فرمان **kill** فرستاده می‌شود.
9	SIGKILL	سیگنال کشتن. این سیگنال باعث خاتمه یافتن فوری پردازش توسط کرنل لینوکس می‌شود. برنامه‌ها نمی‌توانند به این سیگنال گوش بسپارند.

اکنون بیایید فرض کنیم شما برنامه‌ای دارید که به طور ناامیدانه‌ای معلق می‌شود و شما می خواهید آن را مرخص کنید. اکنون کاری که شما انجام می‌دهید:

1. فرمان **ps** را برای بدست آوردن شماره شناسایی پردازشی (PID) که می‌خواهید خاتمه بدهید، به کار ببرید.
2. یک فرمان **kill** برای آن PID صادر کنید.
3. اگر پردازش از خاتمه یافتن امتناع کند (یعنی از سیگنال صرفنظر کند)، سیگنالهای به طور فزاینده خشن‌تر می‌فرستید تا خاتمه یابد.

```
[me@linuxbox me]$ ps x | grep bad_program
PID TTY STAT TIME COMMAND
2931 pts/5 SN 0:00 bad_program

[me@linuxbox me]$ kill -SIGTERM 2931

[me@linuxbox me]$ kill -SIGKILL 2931
```

در مثال فوق من از فرمان **ps** با گزینه x برای لیست تمام پردازشهایم (حتی آنهایی که از ترمینال جاری راه‌اندازی نگردیده‌اند) استفاده کردم. بعلاوه، من خروجی فرمان **ps** را برای اینکه فقط برنامه‌هایی که من علاقمند بودم لیست بشوند به داخل **grep** لوله کشی نمودم. بعد، **kill** را برای صدور سیگنال SIGTERM به برنامه دردسر آفرین به کار بردم. در تجربه واقعی انجام آن به روش زیر معمول‌تر است چون سیگنال پیش‌فرض فرستاده شده توسط **kill** همان SIGTERM است و همچنین **kill** به جای نام سیگنال، می‌تواند از شماره سیگنال استفاده کند:

```
[me@linuxbox me]$ kill 2931
```

آنوقت، در صورتیکه پردازش خاتمه نیابد، با سیگنال SIGKILL آن را مجبور می‌کنیم:

```
[me@linuxbox me]$ kill -9 2931
```

کفایت می‌کند!

به این ترتیب سری درسهای « آموزش پوسته» به پایان می‌رسد. در سری بعد، ***«نوشتن اسکریپت‌های پوسته»***، به چگونگی خودکارسازی وظایف توسط پوسته می‌پردازیم.

Writing Shell Scripts

نوشتن اسکریپت‌های پوسته

اینجاست که خوشی آغاز می‌گردد

با هزاران فرمان معتبر برای کاربر خط فرمان، چگونه می‌توانید تمام آنها را به خاطر بسپارید؟ پاسخ این است، نمی‌توانید. قدرت واقعی کامپیوتر توانایی‌اش در انجام کار برای شما است. برای نیل به این منظور، ما از قدرت پوسته برای خودکار کردن کارها استفاده می‌کنیم. ما ***اسکریپت‌های پوسته*** را می‌نویسیم.

اسکریپت‌های پوسته چه هستند؟

در ساده‌ترین عبارت، یک اسکریپت پوسته یک فایلِ محتوی یک سری از فرمانها است. پوسته این فایل را می‌خواند و فرمانها را مثل اینکه آنها به طور مستقیم در خط فرمان وارد شده‌اند اجرا می‌کند.

پوسته تا اندازه‌ای منحصر به فرد است، از این حیث که هم رابط خط فرمان قدرتمندی برای سیستم است و هم یک مفسر زبان اسکریپت‌نویسی است. به طوری که خواهیم دید، اکثر مواردی که می‌توانند در خط فرمان انجام بشوند، در اسکریپت‌ها نیز می‌توانند انجام بشوند، و اکثر مواردی که در اسکریپت‌ها می‌توانند انجام بشوند در خط فرمان هم می‌توانند انجام بشوند.

ما خیلی از ویژگی‌های پوسته را پوشش داده‌ایم، اما بر ویژگی‌هایی تمرکز نموده‌ایم که بیشتر به طور مستقیم در خط فرمان استفاده می‌شوند. پوسته همچنین ویژگی‌هایی فراهم می‌نماید که معمولاً (اما نه همیشه) موقع نوشتن اسکریپت‌ها به کار می‌روند.

اسکریپت‌ها قدرت ماشین لینوکس شما را آشکار می‌سازند. بنابراین بیایید قدری لذت ببریم!

فهرست مطالب

نوشتن اولین اسکریپت شما و آماده کار نمودن آن

برای نوشتن یک اسکریپت به طور موفقیت‌آمیز، شما باید سه مورد را انجام بدهید:

1. یک اسکریپت بنویسید
2. مجوز اجرای پوسته به آن بدهید
3. آن را جایی قرار بدهید که پوسته بتواند پیدایش کند

نوشتن یک اسکریپت

یک اسکریپت پوسته فایلی است که شامل متن ASCII است. برای ایجاد اسکریپت پوسته، شما یک ***ویرایشگر متن*** را به کار می‌برید. یک ویرایشگر متن برنامه‌ایست، مانند پردازشگر کلمه، که فایلهای متن ASCII را می‌خواند و می‌نویسد. ویرایشگرهای متن خیلی زیادی برای سیستم لینوکس شما در دسترس هستند، هم برای محیط خط فرمان و هم برای محیط رابط گرافیکی کاربر. در اینجا فهرستی از برخی موارد رایج آمده است:

نام	توضیحات	رابط کاربر
`vi, vim`	پدر بزرگ ویرایشگرهای متن یونیکس، **vi**، برای دشواری و ساختار فرمان غیر مستقیم خود بدنام است. در جهت مثبت، **vi** قدرتمند، سبک، و سریع است. آموختن **vi** یک مرحله سرنوشت‌ساز در یونیکس است، چون به طور فراگیر در سیستم‌های بر مبنای یونیکس در دسترس است. در بیشتر توزیع‌های لینوکس، یک نگارش توسعه‌یافته ویرایشگر **vi** به نام **vim** استفاده می‌شود.	خط فرمان
Emacs	غول واقعی دنیای ویرایشگرهای متن، ویرایشگر Emacs نوشته Richard Stallman است. Emacs شامل هر ویژگی تا به حال طرح شده برای یک ویرایشگر متن است (یا می‌تواند برای این شمول آماده بشود) . باید اشاره شود که دوستداران **vi** و Emacs در مورد برتری هر کدام، با کشمکش‌های تند و تیز و طعنه‌آمیزی با یکدیگر جدال می‌کنند.	خط فرمان
`nano`	**nano** یک ویرایشگر متن آزادِ همزاد برنامه پست الکترونیکی **pine** است. استفاده از **nano** بسیار آسان اما ویژگی‌های آن بسیار اندک است. من **nano** را به کاربران تازه کاری که به یک ویرایشگر خط	خط فرمان

	فرمان نیاز دارند پیشنهاد می‌کنم.	
gedit	**gedit** ویرایشگر عرضه شده با محیط میزکار گنوم است.	گرافیکی
kwrite	**kwrite** یک ویرایشگر توسعه یافته ارایه شده با KDE است. از highlighting (برجسته سازیِ) ترکیب دستوری، یعنی یک ویژگی سودمند برای برنامه‌نویسان و نویسندگان اسکریپت برخوردار است.	گرافیکی

اکنون، ویرایشگر متن را روشن کنید و اولین اسکریپت خود به شرح زیر را در آن تایپ کنید:

```
#!/bin/bash
# My first script

echo "Hello World!"
```

افراد زیرک از میان شما کشف خواهند نمود که چطور متن را کپی و در ویرایشگر متن خود بچسبانند 😉

اگر شما اصلاً یک کتاب برنامه‌نویسی را باز کرده باشید فوری تشخیص خواهید داد که این یک برنامه سنتی «Hello World» است. فایل‌تان را با یک نام توصیفی ذخیره کنید. hello_world چطور است؟

اولین سطر اسکریپت مهم است. این یک راهنمای ارایه شده به پوسته به نام ***shebang*** است، که نشان می‌دهد کدام برنامه باید برای تفسیر اسکریپت استفاده شود. در این حالت، برنامه /bin/bash است. سایر زبانهای اسکریپتی از قبیل Perl، awk، tcl، Tk، و python نیز از این ساز و کار استفاده می‌کنند.

سطر دوم یک ***توضیح*** است. هر چیزی که بعد از یک نشانه **#** ظاهر بشود به وسیله **bash** نادیده گرفته می‌شود. هنگامیکه اسکریپت شما بزرگتر و پیچیده‌تر می‌گردد، توضیحات ضروری می‌شوند. آنها توسط برنامه‌نویسان برای روشن کردن آنچه پیش می‌رود و برای اینکه دیگران بتوانند آن را بفهمند، به کار می‌روند. آخرین سطر فرمان **echo** است. این فرمان به سادگی شناسه‌هایش را در نمایشگر چاپ می‌کند.

تنظیم مجوزها

مورد بعدی که ما باید انجام بدهیم دادن مجوز اجرا به اسکریپت شما است. این کار به شکل زیر با فرمان **chmod** انجام می‌شود:

```
[me@linuxbox me]$ chmod 755 hello_world
```

«755» مجوز خواندن، نوشتن، و اجرا به شما خواهد داد. هر فرد دیگر نیز فقط مجوز خواندن و اجرا به دست خواهد آورد. اگر شما می‌خواهید اسکریپت شما خصوصی باشد (یعنی، فقط شما بتوانید بخوانید و اجرا کنید)، به جای آن از «700» استفاده کنید.

قرار دادن آن در Path شما

در این وضعیت، اسکریپت اجرا خواهد شد. این را امتحان کنید:

```
[me@linuxbox me]$ ./hello_world
```

شما باید Hello World! نمایش یافته را مشاهده کنید. در صورتیکه چنین نیست، ببینید واقعاً اسکریپت خود را در کدام دایرکتوری ذخیره نموده‌اید، آنجا بروید و دوباره امتحان کنید.

قبل از اینکه جلوتر برویم، من باید توقف کرده و مقداری در باره pathها صحبت کنم. هنگامی که شما نام یک فرمان را تایپ می‌کنید، سیستم تمام کامپیوتر را برای پیدا کردن جایی که برنامه قرار داده می‌شود جستجو نمی‌کند. این کار خیلی زمان می‌برد. شما توجه نموده‌اید که معمولاً نباید نام مسیر کامل برنامه‌ای را که می‌خواهید اجرا کنید، وارد نمایید، به نظر می‌رسد پوسته آگاه است.

خُب، حق با شماست. پوسته می‌داند. چگونگی آن اینجاست: پوسته فهرستی از دایرکتوری‌هایی که فایلهای قابل اجرا (برنامه‌ها) در آنها قرار داده شده‌اند، نگهداری می‌کند، و فقط دایرکتوری‌های داخل آن فهرست را جستجو می‌کند. اگر بعد از جستجوی هر دایرکتوری موجود در آن فهرست برنامه را پیدا نکند، پیغام خطای مشهور `command not found` را صادر می‌کند.

این فهرست دایرکتوری‌ها *path* شما نامیده می‌شود. شما می‌توانید لیست دایرکتوری‌های آن را با فرمان زیر مشاهده کنید:

```
[me@linuxbox me]$ echo $PATH
```

این فرمان فهرست جداشده با کاراکتر کولن(`:`) دایرکتوری‌هایی را برگشت می‌دهد که اگر موقع صدور فرمان، نام مسیرِ معینی ارایه نشده باشد، جستجو خواهند شد. ما در اولین کوشش خود برای اجرای اسکریپت شما، یک نام مسیر (`./`) برای فایل مشخص نمودیم.

شما می‌توانید دایرکتوری‌ها را با فرمان زیر به path خود اضافه کنید، که در آن ***directory*** نام آن دایرکتوری است که شما می‌خواهید اضافه شود:

```
[me@linuxbox me]$ export PATH=$PATH:directory
```

یک روش بهتر ویرایش کردن فایل `.bash_profile` یا `.profile` شما (نسبت به توزیع شما) برای پیوست کردن فرمان فوق است. به آن طریق، هر بار که شما به سیستم وارد می‌شوید این دایرکتوری به طور خودکار به path اضافه می‌شود.

اکثر توزیع‌های لینوکس رویه‌ای را تشویق می‌کنند که در آن هر کاربر یک دایرکتوری برای برنامه‌های مورد استفاده شخصی‌اش دارد. این دایرکتوری `bin` نام دارد و یک دایرکتوری فرعی در دایرکتوری خانگی شماست. در صورتیکه شما از قبل آن را ندارید، با فرمان زیر آن را ایجاد کنید:

```
[me@linuxbox me]$ mkdir bin
```

اسکریپت‌تان را به دایرکتوری جدید `bin` خود انتقال بدهید و شما آماده هستید. اکنون فقط باید تایپ کنید:

```
[me@linuxbox me]$ hello_world
```

و اسکریپت شما اجرا خواهد شد. در برخی توزیع‌ها، به طور قابل ذکر در Ubuntu، لازم است قبل از اینکه دایرکتوری `bin` ایجاد شده جدید شما شناسایی بشود، یک نشست جدیدِ ترمینال باز کنید.

ویرایش کردن اسکریپت‌هایی که از قبل دارید

قبل از اینکه شما به نوشتن اسکریپت‌های جدید بپردازید، من می‌خواهم نشان بدهم که شما از قبل برخی اسکریپت‌های خودتان را دارید. این اسکریپت‌ها هنگامی که حساب کاربری شما ایجاد شده در دایرکتوری خانگی شما قرار داده شده‌اند، و برای پیکربندی نشست‌های شما در کامپیوتر به‌کار می‌روند. شما می‌توانید این اسکریپت‌ها را برای تغییر امور ویرایش نمایید.

در این درس، ما به دو تا از این اسکریپت‌ها نگاه خواهیم نمود و چند مفهوم جدیدِ مهم را در باره پوسته یاد می‌گیریم.

در جریان یک نشستِ شما، سیستم تعدادی از داده‌ها در باره نشست را در حافظه‌اش نگهداری می‌کند. این اطلاعات ***محیط*** نامیده می‌شوند. محیط شامل مواردی از قبیل path شما، نام کاربری شما، نام فایلی که پست شما به آنجا تحویل می‌شود، و بسیاری دیگر است. شما می‌توانید فهرست کاملی از آنچه را که در محیط شما هست با فرمان `set` مشاهده کنید.

دو نوع از فرمانها اغلب در محیط پیوست می‌شوند. آنها ***مستعارها*** و ***توابع پوسته*** هستند.

محیط چگونه برقرار می‌شود؟

وقتی شما به سیستم متصل می‌شوید، برنامه bash شروع می‌شود، و یک سری از اسکریپت‌های پیکربندی را که *startup files*(فایلهای راه‌اندازی) نامیده می‌شوند، می‌خواند. این فایلها محیط پیش فرض مشترک میان تمام کاربران را تعیین می‌کنند. این مورد با فایلهای راه‌اندازی دیگری در دایرکتوری خانگی شما که محیط شخصی شما را تعیین می‌کنند تعقیب می‌گردد. ترتیب دقیق آن به نوع نشست پوسته‌ای که شروع می‌شود بستگی دارد. دو نوع وجود دارد: یک ***نشست پوسته login*** و یک ***نشست پوسته بدون login***. نشست پوسته login آن نشستی است که در آن اعلان ورود نام کاربری و کلمه عبور به ما داده می‌شود، به عنوان مثال، موقعی که ما یک نشست از کنسول مجازی را شروع می‌کنیم. یک نشست پوسته غیر login به طور نمونه موقعی رخ می‌دهد که ما در رابط گرافیکی کاربر یک نشست ترمینال راه‌اندازی می‌کنیم.

پوسته‌های لاگین یک یا چند فایل راه‌اندازی را به طوری که در ادامه نشان داده شده است می‌خوانند:

فایل	محتویات
`/etc/profile`	یک اسکریپت پیکربندی سراسری که بر تمام کاربران اِعمال می‌گردد.
`~/.bash_profile`	یک فایل راه‌اندازی شخصی کاربر. می‌تواند برای گسترش یا باطل کردن تنظیمات اسکریپت پیکربندی سراسری استفاده شود.
`~/.bash_login`	اگر `~/.bash_profile` پیدا نشود، bash تلاش می‌کند این اسکریپت را بخواند.
`~/.profile`	در صورتیکه نه فایل `~/.bash_profile` پیدا شود و نه فایل `~/.bash_login` آنوقت bash سعی در خواندن این فایل می‌کند. این پیش فرضِ توزیع‌های مبتنی بر دبیان از قبیل اوبونتو است.

نشست‌های پوسته بدون لاگین فایلهای راه‌اندازی زیر را می‌خوانند:

فایل	محتویات
`/etc/bash.bashrc`	یک اسکریپت پیکربندی سراسری که تمام کاربران را شامل می‌شود.

~/.bashrc	یک فایل راه‌اندازی شخصی کاربر. می‌تواند برای برتری یافتن یا توسعه تنظیمات اسکریپت پیکربندی سراسری به کار برود.

علاوه بر خواندن فایلهای راه‌اندازی فوق، پوسته‌های بدون لاگین از پردازش پدر خود که معمولاً یک پوسته لاگین است محیط را به ارث می‌برند.

نگاهی به سیستم خود بیاندازید و ببینید کدام یک از این فایلهای راه‌اندازی را دارید. به خاطر بیاورید- چون نام اکثر فایلهای فهرست شده در بالا با یک نقطه شروع می‌شوند(به معنای اینکه آنها مخفی هستند)، لازم خواهد بود شما موقع استفاده از **ls** گزینه **-a** را به کار ببرید.

فایل **~/.bashrc** از نقطه نظر کاربر عادی احتمالاً مهمترین فایل راه‌اندازی است، چون تقریباً همیشه خوانده می‌شود. پوسته‌های بدون لاگین به طور پیش فرض آن را می‌خوانند و فایلهای راه‌اندازی اکثر پوسته‌های لاگین طوری نوشته می‌شوند که فایل **~/.bashrc** را هم بخوانند.

اگر به داخل یک فایل .bash_profile نمونه نگاه کنیم (این نمونه از یک سیستم CentOS 4 اخذ گردیده است)، موردی مشابه این به نظر می‌رسد:

```
# .bash_profile
# Get the aliases and functions
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi

# User specific environment and startup programs
PATH=$PATH:$HOME/bin
export PATH
```

سطرهایی که با یک "#" شروع می‌شوند توضیحات هستند و به وسیله پوسته خوانده نمی‌شوند. اینها برای خوانایی انسانی در آنجا هستند. اولین مورد جالب در سطر چهارم با کد پایین رخ می‌دهد:

```
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi
```

این یک ***فرمان مرکب if*** نامیده می‌شود، که ما در یک درس دیگر به طور کامل پوشش خواهیم داد، اما در حال حاضر به این جمله ترجمه می‌کنیم:

در صورتیکه فایل "~/.bashrc" موجود است، آنوقت فایل "~/.bashrc" را بخوان.

می‌توانیم مشاهده کنیم که چگونگی دریافت کردن محتویات فایل .bashrc توسط پوسته لاگین، در این خُرده کُد است. مورد بعدی که فایل راه‌انداز ما انجام می‌دهد، تنظیم نمودن متغیر PATH برای اضافه کردن دایرکتوری ~/bin به این متغیر است.

آخر از همه، داریم: export PATH

فرمان **export** به پوسته می‌گوید محتویات متغیر PATH را برای پردازشهای فرزند این پوسته قابل دسترس نماید.

مستعارها

یک مستعار روش آسانی برای تولید فرمان جدیدی است که به عنوان یک کوته‌نوشت برای فرمان بلندتر عمل می‌کند. دارای ترکیب دستوری به شکل زیر است:

```
alias name=value
```

که در آن *name* نام فرمان جدید است و *value* متنی است برای آنکه هرگاه *name* در خط فرمان وارد گردد، اجرا بشود.

بیایید یک مستعار به نام "**l**" ایجاد کنیم و آن را یک مخفف برای فرمان "**ls-l**" نماییم. اطمینان یابید که در دایرکتوری خانگی خود هستید. ویرایشگر متن مورد علاقه خود را به کار برده، و فایل `.bashrc` را باز کنید و این سطر را به انتهای فایل اضافه کنید:

```
alias l='ls -l'
```

با افزودن فرمان **alias** به فایل، ما یک فرمان جدید به نام "**l**" ایجاد نموده‌ایم که "**ls -l**" را انجام خواهد داد. برای امتحان فرمان جدیدتان، نشست ترمینال خود را بسته و نشست جدیدی آغاز کنید. با این کار فایل `.bashrc` دوباره بارگیری می‌شود. با استفاده از این شگرد می‌توانید هر تعداد فرمان سفارشی برای خودتان ایجاد کنید. این هم یک مورد دیگر برای آزمایش کردن توسط شما:

```
alias today='date +"%A, %B %-d, %Y"'
```

این مستعار یک فرمان جدید به نام "**today**" ایجاد می‌کند که تاریخ امروز را با قالب‌بندی دلپذیر نمایش می‌دهد.

در ضمن، فرمان **alias** یک فرمان داخلی(builtin) دیگر پوسته است. شما می‌توانید مستعارهای خود را به طور مستقیم در اعلان فرمان ایجاد کنید، اما آنها فقط در خلال نشست فعلی پوسته اثر گذار خواهند بود. برای مثال:

```
[me@linuxbox me]$ alias l='ls -l'
```

توابع پوسته

مستعارها برای فرمانهای ساده مناسب هستند، اما در صورتیکه بخواهید مورد پیچیده‌تری ایجاد نمایید، شما باید *توابع پوسته* را آزمایش کنید. توابع پوسته را می‌توان به عنوان «اسکریپت داخل اسکریپت» یا اسکریپت‌های فرعی کوچک در نظر گرفت. اجازه بدهید یک مورد را امتحان کنیم. دوباره فایل `.bashrc` را با ویرایشگر متن خود باز کنید و مستعار "**today**" را با کد زیر تعویض کنید:

```
today() {
    echo -n "Today's date is: "
    date +"%A, %B %-d, %Y"
}
```

باور کنید یا نکنید، **()** نیز یک builtin پوسته است، و همچون **alias**، شما می‌توانید توابع پوسته را به طور مستقیم در اعلان فرمان وارد کنید.

```
[me@linuxbox me]$ today() {
> echo -n "Today's date is: "
> date +"%A, %B %-d, %Y"
```

```
> }
[me@linuxbox me]$
```

اگرچه، باز هم مانند **alias**، توابع پوسته‌ای که به طور مستقیم در خط فرمان تعریف شده‌اند تنها در زمان پوسته جاری دوام دارند.

« متغیرها | فهرست مطالب | ویرایش اسکریپت نوشته شده »

here scriptها

با شروع این درس، ما یک برنامه کاربردی سودمند را طرح‌ریزی خواهیم نمود. این برنامه کاربردی یک سند HTML را تولید خواهد نمود که محتوی اطلاعاتی در باره سیستم شما است. من زمان زیادی برای اندیشیدن در باره چگونگی آموزش برنامه‌نویسی شل صرف نمودم، و راهکاری که من انتخاب کرده‌ام بسیار متفاوت با اکثر آنهای دیگری است که من دیده‌ام. اکثرا برخورد سیستماتیک با ویژگی‌های پوسته را ترجیح می‌دهند و اغلب وجود تجربیاتی از سایر زبانهای برنامه‌نویسی را مسلم فرض می‌کنند. با وجود آنکه من فرض نمی‌کنم شما از قبل می‌دانید که چطور برنامه بنویسید، تصدیق می‌کنم که امروزه بسیاری اشخاص می‌دانند چگونه HTML بنویسند، بنابراین برنامه ما یک صفحه web تولید خواهد نمود. همچنانکه ما اسکریپت‌مان را طرح‌ریزی می‌کنیم، قدم به قدم ابزارهای مورد نیاز برای حل مشکل پیش روی خود را مکشوف خواهیم ساخت.

نوشتن یک فایل HTML با یک اسکریپت

همچنانکه شاید بدانید، یک فایل HTML خوش ساخت شامل محتویات زیر است:

```
<html>
<head>
    <title>
    عنوان صفحه شما
    </title>
</head>

<body>
    محتویات صفحه شما در اینجا قرار می‌گیرد.
</body>
</html>
```

اکنون، با آنچه از قبل می‌دانیم، می‌توانیم اسکریپتی برای تولید محتوای فوق بنویسیم:

```
#!/bin/bash

# sysinfo_page - اسکریپتی برای تولید یک فایل html
```

```
echo "<html>"
echo "<head>"
echo "  <title>"
echo "  عنوان صفحه شما"
echo "  </title>"
echo "</head>"
echo ""
echo "<body>"
echo "  محتویات صفحه شما در اینجا قرار می‌گیرد."
echo "</body>"
echo "</html>"
```

این اسکریپت به صورت زیر می‌تواند به کار برود:

```
[me@linuxbox me]$ sysinfo_page > sysinfo_page.html
```

گفته شده است که بزرگترین برنامه‌نویسان تنبل‌ترین نیز هستند. آنها برنامه‌هایی برای صرفه‌جویی در کارشان می‌نویسند. همچنین، موقعی که برنامه‌نویسان باهوش برنامه‌ها را می‌نویسند، سعی می‌کنند در تایپ کردن‌شان صرفه‌جویی کنند.

اولین بهبود برای این اسکریپت، تعویض کاربرد تکراری فرمان **echo** با یک نمونه منفرد به وسیله استفاده با کفایت‌ترِ نقل‌قول خواهد بود:

```
#!/bin/bash

# sysinfo_page - اسکریپتی برای تولید یک فایل html

echo "<html>
 <head>
   <title>
   عنوان صفحه شما
   </title>
 </head>

 <body>
   محتویات صفحه شما در اینجا قرار می‌گیرد.
 </body>
 </html>"
```

با کاربرد نقل‌قول، تعبیه تعویض سطر در متن ما و داشتن شناسه فرمان echo به صورت گسترده در چند سطر میسر می‌گردد.

در حالیکه این مورد به طور یقین یک بهبود است، دارای یک محدودیت نیز هست. چون بسیاری از انواع نشانه‌گذاری به کار رفته در html خودشان در علامت‌های نقل‌قول به هم پیوسته می‌شوند، کاربرد یک رشته نقل‌قول شده را کمی بد قواره می‌سازد. یک رشته نقل‌قولی می‌تواند به کار

برود اما لازم خواهد بود هر علامت نقل‌قول به کار رفته، با کاراکتر \ پوشش داده شود.

به منظور پرهیز از تایپ اضافی، نیازمند جستجوی یک روش مناسب‌تر برای تولید متن خود هستیم. خوشبختانه، پوسته یک مورد را فراهم می‌کند. این مورد ***here script*** نامیده می‌شود.

```
#!/bin/bash

# sysinfo_page - HTML اسکریپتی برای تولید یک فایل

cat << _EOF_
<html>
<head>
    <title>
   عنوان صفحه شما
    </title>
</head>

<body>
   محتویات صفحه شما در اینجا قرار می‌گیرد.
</body>
</html>
_EOF_
```

یک *here script* (همچنین گاهی اوقات ***here document*** نامیده شده) یک شکل اضافی از تغییر مسیر I/O است. این ویژگی روشی برای پیوست کردن محتویاتی که به ورودی استاندارد فرمان داده خواهد شد، فراهم می‌کند. در مورد اسکریپت فوق، یک جریان متن از اسکریپت ما به ورودی استاندارد فرمان **cat** داده شد.

یک*here script* به این شکل ایجاد می‌گردد:

```
command << token
محتویاتی که به عنوان ورودی استاندارد فرمان استفاده می‌شود
token
```

token هر رشته‌ای از کاراکترها می‌تواند باشد. من از "_EOF_" استفاده می‌کنم (EOF یک مخفف برای"End Of File" است) چون عُرف است، اما شما می‌توانید هر رشته‌ای را به کار ببرید، مشروط به اینکه با یک کلمه رزرو شده bash تداخل نکند. آن token که *here script* را خاتمه می‌دهد، باید به طور دقیق نظیر همان باشد که آنرا شروع کرده است، وگرنه بقیه اسکریپت شما به عنوان ورودی استانداردِ بیشتر برای فرمان تفسیر خواهد شد.

یک شگرد اضافی وجود دارد که می‌تواند با *here script* به کار برود. بیشتر اوقات شما برای بهبود بخشی به خوانایی اسکریپت خود، به دندانه‌دار کردن قسمت محتوی *here script* نیاز خواهید داشت. در صورتیکه اسکریپت را به صورت زیر تغییر بدهید می‌توانید آن را انجام بدهید:

```
#!/bin/bash
```

```
# sysinfo_page - HTML اسکریپتی برای تولید یک فایل

cat <<- _EOF_
    <html>
    <head>
        <title>
   عنوان صفحه شما
        </title>
    </head>

    <body>
   محتویات صفحه شما در اینجا قرار می‌گیرد.
    </body>
    </html>
_EOF_
```

تعویض "**<<**" به "**-<<**" باعث می‌شود bash از کاراکترهای tab مقدم (اما فاصله‌ها خیر) در *here script* چشم پوشی کند. خروجی فرمان **cat** شامل هیچ یک از کاراکترهای tab مقدم نخواهد بود.

بسیار خوب، بیایید صفحه خود را بسازیم. ما صفحه‌مان را برای اینکه مطلبی را بیان کند ویرایش می‌کنیم:

```
#!/bin/bash

# sysinfo_page - HTML اسکریپتی برای تولید یک فایل

cat <<- _EOF_
    <html>
    <head>
        <title>
        My System Information
        </title>
    </head>

    <body>
    <h1>My System Information</h1>
    </body>
    </html>
_EOF_
```

در درس بعدی، اسکریپت خود را برای تولید اطلاعات واقعی در باره سیستم آماده خواهیم نمود.

متغیرها

```
#!/bin/bash

# sysinfo_page - HTML اسکریپتی برای تولید یک فایل

cat <<- _EOF_
    <html>
    <head>
        <title>
        My System Information
        </title>
    </head>

    <body>
    <h1>My System Information</h1>
    </body>
    </html>
_EOF_
```

حال که اسکریپت قابل کارکرد خود را داریم، بیایید آن را بهبود بدهیم. اول از همه، برخی تغییرات ایجاد خواهیم نمود زیرا می‌خواهیم تنبل باشیم. در اسکریپت فوق، می‌بینیم که عبارت "My System Information" تکرار می‌شود. این تایپ کردن بیهوده است (و کار اضافی!) بنابراین آن را به این صورت بهبود می‌دهیم:

```
#!/bin/bash

# sysinfo_page - HTML اسکریپتی برای تولید یک فایل

title="My System Information"

cat <<- _EOF_
    <html>
    <head>
        <title>
        $title
```

```
        </title>
    </head>

    <body>
    <h1>$title</h1>
    </body>
    </html>
_EOF_
```

به طوری که می‌توانید مشاهده کنید، یک سطر به ابتدای اسکریپت اضافه کردیم و دو موردِ وجود عبارت "My System Information" را با `$title` تعویض نمودیم.

متغیرها

آنچه ما انجام داده‌ایم معرفی ایده بسیار اساسی ***متغیرها*** است که تقریباً در هر زبان برنامه‌نویسی ظاهر می‌شود. متغیرها محدوده‌هایی از حافظه هستند که می‌توانند برای ذخیره اطلاعات و مراجعه به آنها بواسطه یک نام، به کار بروند. در مورد اسکریپت خودمان، ما متغیری به نام "title" ایجاد کردیم و عبارت "My System Information" را در حافظه قرار دادیم. داخل *here script* که شامل HTML ما است، از "$title" استفاده می‌کنیم که به پوسته بگوییم ***بسط پارامتر*** را انجام بدهد و نام متغیر را با محتوای متغیر تعویض نماید.

هر گاه پوسته کلمه‌ای را که با یک "$" شروع شود مشاهده کند، تلاش می‌کند آنچه به آن اختصاص داده شده را بیابد و جایگزین آن سازد.

چگونگی ایجاد یک متغیر

برای ایجاد یک متغیر، یک سطر در اسکریپت خود قرار بدهید که شامل نام متغیر و به دنبال آن بلافاصله علامت تساوی("=") باشد. هیچ فاصله‌ای مجاز نیست. بعد از علامت تساوی، اطلاعاتی را که مایل به ذخیره هستید تخصیص بدهید.

نام متغیرها از کجا ناشی می‌شوند؟

شما آنها را می‌سازید. درست است، شما نامها را برای متغیرهایتان انتخاب می‌کنید. چند قاعده وجود دارد.

1. نامها باید با یک حرف شروع بشوند.
2. یک نام نباید محتوی فاصله‌ها باشد. به جای آن از underscoreها(خط زیر) استفاده کنید.
3. نمی‌توانید علامت‌های نشانه‌گذاری را به کار ببرید.

این مطلب چگونه تنبلی ما را توسعه می‌دهد؟

افزودن متغیر `title` به دو طریق کار ما را آسانتر می‌کند. نخست، مقدار تایپی که باید انجام بدهیم را کاهش می‌دهد. دوم و به طور مهم‌تر، پشتیبانی اسکریپت ما را آسانتر می‌سازد.

همچنانکه شما اسکریپت‌های بیشتر و بیشتری می‌نویسید (یا هر نوع برنامه‌نویسی دیگری انجام می‌دهید)، خواهید آموخت که برنامه‌ها اصلاً به ندرت تکمیل می‌شوند. آنها به وسیله تولید کنندگان‌شان و سایرین ویرایش می‌شوند و بهبود می‌یابند. به هر حال، این همان موردی است که توسعه منبع باز متوجه آن است. بیایید فرض کنیم شما می‌خواستید در نگارش قبلی اسکریپت، عبارت "My System Information" را به

"Linuxbox System Information." تغییر بدهید، شما می‌بایست در دو محل این را تعویض می‌کردید. در نگارش جدید با متغیر `title` فقط در یک محل باید آن را تعویض کنید. چون اسکریپت ما اینقدر کوچک است، شاید این موضوع کم اهمیتی به نظر آید، اما چنانچه اسکریپت بزرگتر و پیچیده‌تر بشود، بسیار با اهمیت می‌گردد.

متغیرهای محیط

موقعی که شما نشست پوسته خود را آغاز می‌کنید، برخی متغیرها از قبل توسط فایل راه‌اندازی که قبلاً دیدیم، برقرار گردیده‌اند. برای دیدن تمام متغیرهایی که در محیط شما هستند، فرمان **printenv** را به کار ببرید. یک متغیر در محیط شما محتوی نام میزبان برای سیستم شما است. ما به این صورت این متغیر را به اسکریپت‌مان اضافه می‌کنیم:

```
#!/bin/bash

# sysinfo_page - HTML اسکریپتی برای تولید یک فایل

title="System Information for"

cat <<- _EOF_
    <html>
    <head>
        <title>
        $title $HOSTNAME
        </title>
    </head>

    <body>
    <h1>$title $HOSTNAME</h1>
    </body>
    </html>
_EOF_
```

اکنون اسکریپت ما همواره شامل نام ماشینی است که روی آن در حال اجرا هستیم. توجه کنید که، مطابق قرارداد، نام متغیرهای محیط با حروف بزرگ هستند.

جایگزینی فرمان و ثابت‌ها

در درس قبل، چگونگی ایجاد متغیرها و انجام بسط با آنها را آموختیم. در این درس، این ایده را برای نشان دادن آن که چطور می‌توانیم نتایج یک فرمان را جایگزین نماییم، گسترش خواهیم داد.

موقعی که آخرین بار اسکریپت‌مان را ترک کردیم، می‌توانست یک صفحه HTML تولید کند که شامل چند سطر متنِ ساده، به انضمام نام ماشینی باشد که ما از متغیر محیط HOSTNAME به دست آوردیم. اکنون، برای نشان دادن آخرین زمانی که به روزرسانی شده همراه با نام کاربری که آن را انجام داده، یک نشانه زمان به صفحه اضافه خواهیم نمود.

```
#!/bin/bash

# sysinfo_page - HTML اسکریپتی برای تولید یک فایل

title="System Information for"

cat <<- _EOF_
    <html>
    <head>
        <title>
        $title $HOSTNAME
        </title>
    </head>

    <body>
    <h1>$title $HOSTNAME</h1>
    <p>Updated on $(date +"%x %r %Z") by $USER</p>
    </body>
    </html>
_EOF_
```

همچنانکه می‌توانید مشاهده نمایید، یک متغیر محیط دیگر، USER، را برای بدست آوردن نام کاربر استفاده کردیم. بعلاوه، مورد زیر با قیافه ناشناس را به کار بردیم:

```
$(date +"%x %r %Z")
```

کاراکترهای "$( )" به پوسته می‌گویند، «نتایج فرمان محصور شده در پرانتزها را جایگزین کن». ما در اسکریپت‌مان می‌خواهیم پوسته نتایج فرمان

date +"%x %r %Z" را که بیان‌کننده تاریخ و زمان جاری است، درج کند. فرمان **date** دارای ویژگی‌ها و گزینه‌های قالب‌بندی بسیار است. برای دیدن تمام آنها، این کد را امتحان کنید:

```
[me@linuxbox me]$ date --help | less
```

آگاه باشید که یک ترکیب جایگزین قدیمی‌تر برای "$(command)" وجود دارد که کاراکتر backtick (`) را استفاده می‌کند. این شکل قدیمی‌تر با پوسته اصیل (Bourne (sh سازگار است. من مایل به استفاده از این شکل قدیمی‌تر نیستم چون من اینجا در حال آموزش **bash** مدرن هستم، نه **sh**، و گذشته از این، فکر می‌کنم backtickها زشت هستند. پوسته bash اسکریپت‌های نوشته شده برای sh را کاملاً پشتیبانی می‌کند بنابراین ترکیب‌های زیر نتایج یکسانی دارند:

```
$(command)
`command`
```

تخصیص نتایج یک فرمان به یک متغیر

شما همچنین می‌توانید نتایج یک فرمان را به یک متغیر تخصیص بدهید:

```
right_now=$(date +"%x %r %Z")
```

حتی می‌توانید متغیرها را تو در تو نمایید (قرار دادن یکی در داخل دیگری)، مانند این:

```
right_now=$(date +"%x %r %Z")
time_stamp="Updated on $right_now by $USER"
```

ثابت‌ها

همچنانکه نام متغیر اشاره می‌کند، محتوای یک متغیر تابع تغییر است. این به معنای آن است که انتظار می‌رود در خلال اجرای اسکریپت شما، شاید بواسطه موردی که شما انجام می‌دهید محتوایش ویرایش گردیده باشد.

از طرف دیگر، ممکن است مقادیری وجود داشته باشند که یکبار تنظیم شوند، و هرگز نباید تغییر کنند. اینها *ثابت‌ها* نامیده می‌شوند. من این را در اینجا می‌آورم زیرا یک مفهوم رایج در برنامه‌نویسی است. اکثر زبانهای برنامه‌نویسی دارای امکانات خاصی برای پشتیبانی از کمیت‌هایی هستند که تغییر آنها اجازه داده نمی‌شود. Bash نیز دارای این امکانات هست، اما صادقانه، من هرگز استفاده آن را ندیدم. در عوض، اگر یک کمیت نامزد ثابت بودن باشد، یک نام با حروف بزرگ به آن داده می‌شود برای اینکه به برنامه نویس یادآوری گردد که آن کمیت باید یک ثابت در نظر گرفته شود، حتی اگر مجبور به آن نشود.

متغیرهای محیط به طور معمول ثابت در نظر گرفته می‌شوند چون آنها به ندرت تغییر داده می‌شوند. همانند ثابت‌ها، به متغیرهای محیط مطابق قرارداد نام‌های با حروف بزرگ داده می‌شود. در اسکریپت‌های پس از این، من این قرارداد را به کار خواهم برد ـ نام‌های با حروف بزرگ برای ثابت‌ها و حروف کوچک برای متغیرها.

بنابراین با آنچه ما می‌دانیم، برنامه ما اینطور ظاهر می‌گردد:

```
#!/bin/bash

# sysinfo_page - HTML اسکریپتی برای تولید یک فایل

title="System Information for $HOSTNAME"
RIGHT_NOW=$(date +"%x %r %Z")
TIME_STAMP="Updated on $RIGHT_NOW by $USER"

cat <<- _EOF_
    <html>
    <head>
        <title>
        $title
        </title>
    </head>

    <body>
    <h1>$title</h1>
    <p>$TIME_STAMP</p>
    </body>
    </html>
_EOF_
```

توابع پوسته

همچنانکه برنامه‌ها بزرگتر و پیچیده‌تر می‌گردند، برای طراحی، کدنویسی، و پشتیبانی دشوارتر می‌شوند. همانند با هر جد و جهد بزرگ، بیشتر اوقات تفکیک یک وظیفه بزرگ به یک گروه از وظایف کوچکتر، سودمند است.

در این درس، ما تفکیک اسکریپت یکپارچه خود به یک تعداد توابع جداگانه را آغاز خواهیم نمود.

برای آشنا شدن با این اندیشه، بیایید شرح یک وظیفه روزانه را در نظر بگیریم ــ رفتن به فروشگاه برای خرید غذا. فرض کنید ما می‌خواستیم وظیفه را برای مردی از مریخ تشریح کنیم.

اولین تشریح سطح بالای ما شاید چیزی مانند این به نظر بیاید:

1. ترک کردن خانه
2. راندن به طرف فروشگاه
3. پارک کردن ماشین
4. وارد شدن به فروشگاه
5. خریداری غذا
6. راندن به طرف خانه
7. پارک کردن ماشین
8. ورود به خانه

این توضیحات پردازش کلی رفتن به فروشگاه را پوشش می‌دهد، اما مردی از مریخ احتمالاً به جزییات اضافی نیاز خواهد داشت. برای مثال، وظیفه فرعی «پارک ماشین» می‌توانست به صورت زیر تشریح گردد:

1. یافتن مکان پارک
2. راندن ماشین به این مکان
3. خاموش کردن موتور
4. کشیدن ترمز دستی
5. خروج از ماشین
6. قفل کردن ماشین

البته وظیفه «خاموش کردن موتور» دارای چند مرحله از قبیل «بستن سوییچ» و «بیرون آوردن سوییچ» و مانند آن است.

این پردازش شناسایی مراحل سطح بالا و گسترش افزایشی جنبه‌های جزیی آن مراحل، *طراحی بالا به پایین* نامیده می‌شود. این اسلوب، تجزیه وظایف پیچیده بزرگ به وظایف کوچک و ساده بسیار را برای شما میسر می‌سازد.

همچنانکه اسکریپت ما به رشد ادامه می‌دهد، ما از طراحی بالا به پایین برای کمک به خود جهت طراحی و کدنویسی اسکریپت‌مان استفاده می‌کنیم.

اگر به وظایف سطح بالای اسکریپت خود نگاه کنیم، لیست پایین را تشخیص می‌دهیم:

1. باز کردن صفحه
2. باز کردن بخش head

3. نوشتن عنوان
4. بستن بخش head
5. باز کردن بخش body
6. نوشتن عنوان
7. نوشتن نشانه زمان
8. بستن بخش body
9. بستن صفحه

تمام این وظایف اجرا می‌شوند، اما می‌خواهیم وظایف دیگری اضافه کنیم. اجاز ه بدهید چند وظیفه اضافی بعد از وظیفه شماره ۷ درج کنیم:

7. نوشتن نشانه زمان
8. نوشتن اطلاعات نگارش سیستم
9. نوشتن مدت زمان روشن بودن سیستم
10. نوشتن فضای دیسک
11. نوشتن فضای دایرکتوری خانگی
12. بستن بخش body
13. بستن صفحه

خیلی خوب بود اگر فرمانهایی وجود داشتند که این وظایف را انجام می‌دادند، ما می‌توانستیم جایگزینی فرمان را برای قرار دادن آنها در اسکریپت‌مان به این شکل به کار ببریم:

```
#!/bin/bash

# sysinfo_page - اسکریپتی برای تولید یک فایل HTML اطلاعات سیستم

##### ثابت‌ها

TITLE="System Information for $HOSTNAME"
RIGHT_NOW=$(date +"%x %r %Z")
TIME_STAMP="Updated on $RIGHT_NOW by $USER"

##### Main

cat <<- _EOF_
  <html>
  <head>
      <title>$TITLE</title>
  </head>

  <body>
      <h1>$TITLE</h1>
      <p>$TIME_STAMP</p>
```

```
      $(system_info)
      $(show_uptime)
      $(drive_space)
      $(home_space)
  </body>
  </html>
_EOF_
```

اگر چه فرمانهایی وجود ندارند که به طور دقیق آنچه را ما نیاز داریم انجام بدهند، ما می‌توانیم آنها را با کاربرد ***توابع پوسته*** ایجاد کنیم.

به طوریکه در درس دوم آموختیم، توابع پوسته همچون «برنامه‌های کوچک در درون برنامه‌ها» عمل می‌کنند و پیروی از اصول طراحی بالا به پایین را برای ما میسر می‌سازند. برای افزودن توابع به اسکریپت‌مان، آن را به این صورت تغییر می‌دهیم:

```
#!/bin/bash

# sysinfo_page - اسکریپتی برای تولید یک فایل HTML اطلاعات سیستم

##### ثابت‌ها

TITLE="System Information for $HOSTNAME"
RIGHT_NOW=$(date +"%x %r %Z")
TIME_STAMP="Updated on $RIGHT_NOW by $USER"

##### توابع

system_info()
{

}


show_uptime()
{

}


drive_space()
{

}
```

```
home_space()
{

}

##### Main

cat <<- _EOF_
  <html>
  <head>
      <title>$TITLE</title>
  </head>

  <body>
      <h1>$TITLE</h1>
      <p>$TIME_STAMP</p>
      $(system_info)
      $(show_uptime)
      $(drive_space)
      $(home_space)
  </body>
  </html>
_EOF_
```

یک زوج نکته مهم در باره توابع: نخست، آنها باید قبل از آنکه شما سعی در استفاده از آنها بنمایید، پدیدار شوند. دوم، بدنه تابع (قسمتی از تابع مابین کاراکترهای { و }) حداقل باید شامل یک فرمان معتبر باشد. اسکریپت به طوریکه نوشته شده، بدون خطا اجرا نخواهد گردید، زیرا بدنه تابع‌ها خالی هستند. راه آسان برای تعمیر این مورد، قرار دادن یک جمله **return** در بدنه هر تابع است. بعد از آنکه این را انجام بدهید، اسکریپت دوباره به طور موفقیت‌آمیز اجرا می‌گردد.

اسکریپت‌هایتان را آماده کار نگهدارید

موقعی که یک برنامه را توسعه می‌دهید، بیشتر اوقات اضافه کردن یک مقدار اندک کد، اجرای اسکریپت، اضافه کردن مقدار دیگری کد، اجرای اسکریپت، و به همین منوال، شیوه مناسبی است. این روش، در صورتیکه شما در کُدتان اشتباه وارد کنید، یافتن و تصحیح آن را آسانتر خواهد نمود.

همچنانکه شما توابع را به اسکریپت خود اضافه می‌کنید، می‌توانید اسلوبی را که ***stubbing*** نامیده می‌شود برای کمک به مراقبت از منطق توسعه اسکریپت خود به کار ببرید. Stubbing به این صورت عمل می‌کند: تصور کنید که ما می‌خواهیم تابعی به نام "system_info" ایجاد کنیم اما هنوز تمام جزییات کُد آن را معین نکرده‌ایم. به جای معطل کردن پیشرفت اسکریپت تا زمانی که ما system_info را تمام کرده باشیم، فقط یک فرمان **echo** به این صورت اضافه می‌کنیم:

```
system_info()
{
    # Temporary function stub
    echo "function system_info"
}
```

به این طریق، اسکریپت ما بازهم به طور موفق اجرا خواهد گردید، ولواینکه ما هنوز دارای یک تابع تکمیل شده system_info نیستیم. ما در آینده کد موقت ***stubbing*** را با نگارش کامل کارگر تعویض خواهیم نمود.

علت استفاده ما از یک فرمان **echo** آن است که ما واکنشی از اسکریپت دریافت می‌کنیم که بیانگر اجرا شدن توابع است.

بیایید پیش برویم و برای اسکریپت‌مان ***stub***هایی بنویسیم و اسکریپت را قابل اجرا نگاه داریم.

```
#!/bin/bash

# sysinfo_page - اسکریپتی برای تولید یک فایل HTML اطلاعات سیستم

##### ثابت‌ها

TITLE="System Information for $HOSTNAME"
RIGHT_NOW=$(date +"%x %r %Z")
TIME_STAMP="Updated on $RIGHT_NOW by $USER"

##### توابع

system_info()
{
    # تابع stub موقت
    echo "function system_info"
}


show_uptime()
{
    # تابع stub موقت
    echo "function show_uptime"
}


drive_space()
```

```
{
    # Temporary function stub
    echo "function drive_space"
}



home_space()
{
    # موقت stub تابع
    echo "function home_space"
}



##### بخش اصلی

cat <<- _EOF_
  <html>
  <head>
      <title>$TITLE</title>
  </head>

  <body>
      <h1>$TITLE</h1>
      <p>$TIME_STAMP</p>
      $(system_info)
      $(show_uptime)
      $(drive_space)
      $(home_space)
  </body>
  </html>
_EOF_
```

قدری کار واقعی

در این درس، برخی از توابع پوسته خود را برای تولید مقداری اطلاعات سودمند گسترش می‌دهیم.

show_uptime

تابع show_uptime خروجی فرمان **uptime** را نمایش خواهد داد. فرمان **uptime** چند نکته مهم در مورد سیستم، از جمله طول مدت زمانی که سیستم از آخرین راه‌اندازی (up) در حال اجرا بوده است، تعداد کاربران و بارگیری اخیر سیستم را بیرون می‌دهد.

```
[me@linuxbox me]$ uptime
9:15pm up 2 days, 2:32, 2 users, load average: 0.00, 0.00, 0.00
```

برای به دست آوردن خروجی فرمان **uptime** در صفحه HTML خود، ما تابع پوسته‌مان را با تعویض کد stub موقت با نگارش تکمیل شده به این صورت می‌نویسیم:

```
show_uptime()
{
    echo "<h2>System uptime</h2>"
    echo "<pre>"
    uptime
    echo "</pre>"
}
```

همانطور که می‌توانید مشاهده نمایید، این تابع جریانی از متن شامل یک مخلوطی از برچسب‌های HTML و خروجی فرمان را بیرون می‌دهد. هنگامیکه در بدنه اصلی برنامه ما جایگزینی فرمان صورت بگیرد، خروجی تابع ما قسمتی از ***here script*** می‌گردد.

drive_space

تابع drive_space فرمان **df** را برای فراهم نمودن خلاصه‌ای از فضای تمام سیستم فایل‌های متصل شده استفاده خواهد نمود.

```
[me@linuxbox me]$ df
Filesystem   1k-blocks      Used Available Use% Mounted on

/dev/hda2       509992    225772    279080  45% /
/dev/hda1        23324      1796     21288   8% /boot
```

```
/dev/hda3     15739176   1748176  13832360  12% /home
/dev/hda5      3123888   3039584     52820  99% /usr
```

از لحاظ ساختار، تابع drive_space بسیار مشابه تابع show_uptime است:

```
drive_space()
{
    echo "<h2>Filesystem space</h2>"
    echo "<pre>"
    df
    echo "</pre>"
}
```

home_space

تابع home_space مقدار فضایی را که هر کاربر در دایرکتوری خانگی‌اش استفاده می‌کند نمایش خواهد داد. این اطلاعات را به صورت یک لیست نمایش می‌دهد، که به طور نزولی بر حسب مقدار فضای مورد استفاده مرتب شده است.

```
home_space()
{
    echo "<h2>Home directory space by user</h2>"
    echo "<pre>"
    echo "Bytes Directory"
    du -s /home/* | sort -nr
    echo "</pre>"
}
```

توجه نمایید که برای اینکه این تابع به طور موفق اجرا گردد، اسکریپت باید توسط کاربر ارشد اجرا گردد، چون فرمان **du** برای بررسی محتویات دایرکتوری /home نیازمند مزایای کاربر ارشد است.

system_info

ما هنوز برای تمام کردن تابع system_info آماده نیستیم. تا آن هنگام، کد stubbing را به طوریکه HTML معتبری تولید کند بهینه‌سازی می‌نماییم:

```
system_info()
{
    echo "<h2>System release info</h2>"
```

```
    echo "<p>Function not yet implemented</p>"
}
```

« پرهیز از دردسر
فهرست مطالب
برخی کارهای واقعی »

کنترل جریان ـ بخش 1

در این درس، به چگونگی افزودن هوش به اسکریپت‌مان نگاه خواهیم نمود. تا اینجا، اسکریپت پروژه ما تنها از یک توالی فرمانها تشکیل گردیده است که در سطر اول شروع می‌شود و تا رسیدن به انتها سطر به سطر ادامه می‌یابد. اکثر برنامه‌ها کاری بسیار بیش از این انجام می‌دهند. آنها تصمیم‌گیری می‌کنند و بر اساس شرایط، متفاوت عمل می‌کنند.

پوسته چندین فرمان ارایه می‌کند که ما می‌توانیم برای کنترل جریانِ اجرا در برنامه خود به‌کار ببریم. در این درس، ما به موارد زیر نگاه خواهیم نمود:

- `if`
- `test`
- `exit`

if

اولین فرمانی که بررسی خواهیم کرد `if` است. فرمان `if` در ظاهر تا اندازه‌ای ساده است، بر اساس ***وضعیت خروج***ِ یک فرمان تصمیمی می‌گیرد. فرمان `if` دارای ترکیب دستوری زیر است:

```
if commands; then
commands
[elif commands; then
commands...]
[else
commands]
fi
```

که در آن ***commands*** لیستی از فرمانها است. در نگاه اول این کمی مغشوش کننده است. اما پیش از آن‌که ما بتوانیم آن را واضح کنیم، باید نگاه کنیم که پوسته چطور موفقیت یا شکست یک فرمان را ارزیابی می‌کند.

وضعیت خروج

فرمان‌ها (شامل اسکریپت‌ها و توابعی که ما می‌نویسیم) هنگامی که خاتمه می‌یابند، کمیتی به سیستم صادر می‌کنند که یک وضعیت خروج نامیده می‌شود. این کمیت، که یک عدد صحیح در محدوده 0 تا 255 است نشان‌دهنده موفقیت یا شکست اجرای فرمان است. مطابق قرارداد، مقدار صفر بیانگر موفقیت و هر مقدار دیگری نشان دهنده شکست است. پوسته پارامتری فراهم می‌کند که ما می‌توانیم برای بازپرسی وضعیت خروج به‌کار ببریم. در اینجا آن را در عمل مشاهده می‌کنیم:

```
[me@linuxbox ~]$ ls -d /usr/bin
/usr/bin
[me@linuxbox ~]$ echo $?
0
[me@linuxbox ~]$ ls -d /bin/usr
ls: cannot access /bin/usr: No such file or directory
[me@linuxbox ~]$ echo $?
2
```

در این مثال، ما فرمان **ls** را دوبار اجرا می‌کنیم. دفعه اول، فرمان موفقیت‌آمیز اجرا می‌گردد. اگر ما مقدار پارامتر **$?** را نمایش بدهیم، مشاهده می‌کنیم که صفر است. فرمان **ls** را نوبت دوم اجرا می‌کنیم، یک خطا تولید می‌کند و دوباره پارامتر **$?** را بازپرسی می‌کنیم. این دفعه محتوی 2 است، نشان‌دهنده آن که فرمان با یک خطا مواجه گردیده است. بعضی فرمانها مقادیرِ وضعیتِ خروجِ متفاوتی به منظورِ میسر کردن عیب‌شناسی خطاها به کار می‌برند، در حالیکه بسیاری از فرمانها وقتی ناموفق می‌شوند به سادگی با یک مقدار 1 خارج می‌گردند. صفحه‌های man اغلب بخشی با عنوان «وضعیت خروج» در توضیح این که کدام کُدها استفاده می‌شوند ضمیمه می‌کنند. در هر صورت صفر همیشه بیانگر موفقیت است.

پوسته دو فرمان به شدت سادهِ داخلی فراهم می‌کند که کاری انجام نمی‌دهند غیر از اینکه به یک وضعیت خروج صفر یا یک منتهی می‌شوند. فرمان **true** همیشه به طور موفقیت‌آمیز اجرا می‌گردد و فرمان **false** همواره به طور ناموفق اجرا می‌شود:

```
[me@linuxbox~]$ true
[me@linuxbox~]$ echo $?
0
[me@linuxbox~]$ false
[me@linuxbox~]$ echo $?
1
```

ما می‌توانیم از این فرمانها برای مشاهده آنکه جمله **if** چطور کار می‌کند استفاده نماییم. آنچه جمله **if** واقعاً انجام می‌دهد ارزیابی موفقیت یا شکست فرمانها می‌باشد:

```
[me@linuxbox ~]$ if true; then echo "It's true."; fi
It's true.
[me@linuxbox ~]$ if false; then echo "It's true."; fi
[me@linuxbox ~]$
```

فرمان `echo "It's true."` موقعی اجرا می‌گردد که فرمان بعد از **if** به طور موفق اجرا می‌شود، و هنگامی که اجرای فرمان بعد از **if** موفقیت‌آمیز نیست اجرا نمی‌گردد.

test

اکثر اوقات فرمان **test** با فرمان **if** برای انجام دادن داوری‌های صحیح-غلط به کار می‌رود. فرمانی غیر عادی است از این حیث که دارای دو شکل دستوری متفاوت است:

```
# شکل اول

test expression

# شکل دوم

[ expression ]
```

فرمان **test** به طور ساده کار می‌کند. اگر عبارت داده شده صحیح باشد، **test** با وضعیت خروج صفر خارج می‌شود، در غیر اینصورت با یک وضعیت 1 خارج می‌گردد.

ویژگی پاکیزه **test** در تنوع عبارت‌هایی است که شما می‌توانید تولید کنید. این هم یک مثال:

```
if [ -f .bash_profile ]; then
    echo "You have a .bash_profile. Things are fine."
else
    echo "Yikes! You have no .bash_profile!"
fi
```

مترجم: اگر شما فرمان echo را با شناسه محتوی کاراکتر ! در پوسته محاوره‌ای به کار ببرید با خطایی مانند bash: !": event not found مواجه می‌شوید. توضیح کامل

در این مثال، ما عبارت " **-f .bash_profile** " را به کار می‌بریم. این عبارت می‌پرسد، «آیا .bash_profile یک فایل است؟» اگر عبارت صحیح باشد، آنوقت **test** با یک صفر (بیانگر صحیح) خارج می‌شود و فرمان **if** دستور(ات) پس از کلمه **then** را اجرا می‌کند. در صورتیکه عبارت غلط باشد، آنوقت **test** با وضعیت یک خارج می‌شود و فرمان **if** دستور(ات) بعد از کلمه **else** را اجرا می‌کند.

این هم یک فهرست نیمه کامل از شرایطی که **test** می‌تواند ارزیابی نماید. چون **test** یک builtin پوسته است، از"**help test**" برای دیدن فهرست کامل استفاده نمایید.

عبارت	توضیح
-d ***file***	در صورتیکه ***file*** یک دایرکتوری باشد صحیح است.
-e ***file***	اگر ***file*** موجود باشد صحیح است.
-f ***file***	اگر ***file*** موجود و یک فایل معمولی باشد صحیح است.
-L ***file***	اگر ***file*** یک پیوند نمادین باشد صحیح است.

قبل از اینکه پیش برویم، من می‌خواهم باقیمانده مثال بالا را شرح بدهم، چون ایده‌های با اهمیت‌تری را هم آشکار می‌سازد.

در سطر نخست از اسکریپت، فرمان **if** را می‌بینیم که به وسیله فرمان **test** ادامه یافته، و با یک semicolon دنبال می‌گردد، و سرانجام کلمه **then**. من استفاده از **[*expression*]** را برای فرمان **test** انتخاب کردم چون اکثر اشخاص خواندن آن را آسانتر می‌دانند. توجه نمایید که فاصله مابین "**[**" و شروعِ عبارت ضروری است. به همچنین، فاصله بین انتهای عبارت و "**]**" پس از آن لازم است.

کاراکتر **;** یک جداکننده فرمان است. کاربرد آن به شما اجازه می‌دهد بیش از یک فرمان را در یک سطر قرار بدهید. برای مثال:

```
[me@linuxbox me]$ clear; ls
```

صفحه را پاک کرده و فرمان **ls** را اجرا خواهد نمود.

من از semicolon استفاده می‌کنم چون به من اجازه می‌دهد کلمه **then** را در همان سطر فرمان **if** قرار بدهم، زیرا من گمان می‌کنم به این ترتیب برای خواندن آسانتر است.

در سطر دوم، دوست قدیمی ما **echo** حضور دارد. تنها مورد قابل توجه در این سطر تو گذاری است. دوباره برای سودمندی خوانایی، توگذاری تمام قطعه کُد شرطی، یعنی هر کدی که تنها در صورتی اجرا می‌گردد که شرایط معینی تحقق یابد، قراردادی است. پوسته به این مورد نیاز ندارد، این کار خواندن کد را آسان می‌کند.

به بیان دیگر، ما می‌توانستیم به صورت زیر بنویسیم و همان نتایج را به دست می‌آوریم:

```
# شکل جا یگزین

if [ -f .bash_profile ]
then
    echo "You have a .bash_profile. Things are fine."
else
    echo "Yikes! You have no .bash_profile!"
fi

# یک شکل جا یگزین د یگر

if [ -f .bash_profile ]
then echo "You have a .bash_profile. Things are fine."
else echo "Yikes! You have no .bash_profile!"
fi
```

exit

برای اینکه نویسندگان اسکریپت خوبی باشیم، ما باید وضعیت خروج را برای موقعی که اسکریپت تمام می‌شود تنظیم کنیم. برای انجام این کار، فرمان **exit** را به کار می‌بریم. فرمان **exit** باعث می‌شود بدون واسطه خاتمه یافته و وضعیت خروج را به هر مقدار ارایه شده به عنوان شناسه‌اش

تنظیم نماید. برای مثال:

```
exit 0
```

اسکریپت شما خارج می‌شود و وضعیت خروج را به صفر (موفقیت) تنظیم می‌کند، در حالیکه با

```
exit 1
```

اسکریپت شما خارج می‌شود و وضعیت خروج را به 1 (شکست) تنظیم می‌کند.

بررسی برای کاربر ارشد

وقتی آخرین بار اسکریپت خود را ترک کردیم، نیاز داشتیم که با مزایای کاربر ارشد اجرا گردد. چنین است زیرا تابع **home_space** به بررسی کردن اندازه دایرکتوری خانگی هر کاربر احتیاج دارد، و تنها کاربر ارشد اجازه انجام آن را دارد.

اما اگر یک کاربر عادی اسکریپت ما را اجرا کند چه رخ می‌دهد؟ مقدار زیادی پیغام خطاهای بد ریخت تولید می‌کند. اگر می‌توانستیم چیزی در اسکریپت قرار بدهیم که وقتی کاربر معمولی تلاش در اجرای آن نماید متوقف شود چطور؟

فرمان **id** می‌تواند به ما بگوید چه کسی کاربر جاری است. موقعی که با گزینه **-u** به کار برود، شماره شناسایی کاربر جاری را چاپ می‌کند.

```
[me@linuxbox me]$ id -u
501
[me@linuxbox me]$ su
Password:
[root@linuxbox me]# id -u
0
```

اگر کاربر ارشد **id -u** را اجرا کند، فرمان "0" را بیرون می‌دهد. این واقعیت می‌تواند اساس بررسی ما قرار گیرد:

```
if [ $(id -u) = "0" ]; then
    echo "superuser"
fi
```

در این مثال، در صورتیکه خروجی فرمان **id -u** معادل رشته "0" باشد، آنوقت رشته "superuser" چاپ می‌شود.

در حالیکه اگر کاربر superuser باشد این کد تشخیص خواهد داد، این هنوز به طور واقعی مشکل را رفع نمی‌کند. ما می‌خواهیم در صورتیکه کاربر جاری کاربر ارشد نباشدتوقف شود، بنابراین کدی مانند این می‌نویسیم:

```
if [ $(id -u) != "0" ]; then
    echo "You must be the superuser to run this script" >&2
    exit 1
fi
```

با این کد، اگر خروجی فرمان **id -u** معادل با "0" نباشد، آنوقت اسکریپت یک پیغام خطای وصفی چاپ کرده، خارج می‌شود، و برای نشان دادن آنکه اسکریپت به طور ناموفق اجرا گردیده وضعیت خروج را به 1 تنظیم می‌کند.

به ">&2" در انتهای فرمان **echo** توجه نمایید. این شکل دیگری از تغییر مسیر ورودی-خروجی است. بیشتر اوقات شما این مورد را در روال‌هایی که پیغام‌های خطا نمایش می‌دهند ملاحظه خواهید نمود. اگر این تغییر مسیر انجام نمی‌شد، پیغام خطا به خروجی استاندارد می‌رفت. با این تغییر مسیر پیغام به خطای استاندارد فرستاده شد. چون ما در حال اجرای اسکریپت و فرستادن خروجی استانداردش به یک فایل هستیم، خواستار پیغام خطای جداشده از خروجی معمولی هستیم.

می‌توانستیم این روال را نزدیک ابتدای اسکریپت‌مان قرار بدهیم به این ترتیب شانس تشخیص دادن خطای احتمالی قبل از پیش رفتن موارد را دارد، اما به منظور اجرای این اسکریپت به عنوان یک کاربر معمولی، همان ایده را به کار می‌بریم و در عوض تابع home_space را برای تست امتیازهای مناسب به این صورت ویرایش می‌کنیم:

```
function home_space
{
    # فقط کاربر ارشد می‌تواند این اطلاعات را به دست بیاورد

    if [ "$(id -u)" = "0" ]; then
        echo "<h2>Home directory space by user</h2>"
        echo "<pre>"
        echo "Bytes Directory"
            du -s /home/* | sort -nr
        echo "</pre>"
    fi

}   # end of home_space
```

به این طریق، اگر یک کاربر معمولی اسکریپت را اجرا کند، کد دردسر آفرین به جای آنکه اجرا شود، چشم پوشی می‌گردد و مشکل رفع خواهد شد.

از دردسر بر حذر باشید

اکنون که اسکریپت‌های ما کمی پیچیده‌تر می‌شوند، من می‌خواهم به برخی اشتباهات رایج که ممکن است برخورد نمایید اشاره کنم. برای انجام این کار، اسکریپ زیر به نام **trouble.bash** را ایجاد کنید. اطمینان حاصل کنید که آن را دقیقاً همانطور که نوشته شده است وارد کنید.

```
#!/bin/bash

number=1

if [ $number = "1" ]; then
    echo "Number equals 1"
else
    echo "Number does not equal 1"
fi
```

موقعی که شما این اسکریپت را اجرا می‌کنید، باید سطر "Number equals 1" را بیرون بدهد، خوب به علت اینکه **number** مساوی 1 است. درصورتیکه خروجی مورد انتظار را دریافت نکردید، تایپ خود را بررسی کنید، اشتباه کرده‌اید.

متغیرهای تهی

اسکریپت را ویرایش کنید و سطر 3 را از:

```
number=1
```

تبدیل کنید به:

```
number=
```

و دوباره اسکریپت را اجرا کنید. این دفعه بایدنتیجه زیر را به دست بیاورید:

```
[me@linuxbox me]$ ./trouble.bash
/trouble.bash: [: =: unary operator expected.
Number does not equal 1
```

چنانکه می‌توانید مشاهده کنید، وقتی اسکریپت اجرا شود **bash** یک پیغام خطا نمایش می‌دهد. احتمالاً شما گمان می‌کنید با حذف کردن "1" از سطر سوم در این سطر یک خطای دستوری ایجاد گردیده است، اما اینطور نیست. بیایید دوباره به پیغام خطا نگاه کنیم:

```
./trouble.bash: [: =: unary operator expected
```

می‌توانیم ببینیم که **./trouble.bash** خطا گزارش می‌کند و خطا باید مربوط به "**[**" باشد. به خاطر بیاورید که "**[**" یک اختصار برای فرمان داخلی **test** پوسته است. از این مطلب می‌توانیم متوجه شویم خطا در سطر 5 رخ می‌دهد و نه در سطر 3.

نخست، اجازه بدهید من بگویم که در سطر 3 اشتباهی وجود ندارد. **number=** ترکیب دستوری کاملاً مناسبی است. گاهی اوقات شما می‌خواهید مقدار متغیر را به هیچ تنظیم کنید. می‌توانید با آزمایش کردن این مورد در خط فرمان آن راتصدیق کنید:

```
[me@linuxbox me]$ number=
[me@linuxbox me]$
```

مشاهده کنید، هیچ پیغام خطایی نیست. بنابراین در سطر 5 چه موردی اشتباه است؟ قبلاً کار می‌کرد.

برای فهمیدن این خطا، ما باید آنچه را پوسته می‌بیند مشاهده کنیم. به خاطر بیاورید که پوسته مقدار زیادی از وقتش را صرف بسط دادن متن می‌کند. در سطر 5، پوسته در جایی که **$number** را می‌بیند، مفدار **number** را بسط می‌دهد. در آزمون نخست ما (موقع **number=1**)، پوسته 1 را برای **$number** جایگزین نمود، مانند این:

```
if [ 1 = "1" ]; then
```

اما، موقعی که ما عدد را به هیچ تنظیم نمودیم (**number=**)، پوسته بعد از بسط این مورد را دیده است:

```
if [ = "1" ]; then
```

که یک خطا است. این مابقی پیغام خطایی را که دریافت نموده‌ایم نیز توضیح می‌دهد. "**=**" یک عملگر دوگانی است، یعنی دوشناسه را برای عمل کردن روی آنها انتظار دارد ـ یکی در هر طرف. آنچه پوسته سعی می‌کند به ما بگوید، آن است که فقط یک شناسه وجود دارد و عملگر باید یگانی باشد (مانند "**!**") که فقط بر یک شناسه منفرد عمل می‌کند.

برای اصلاح این مشکل، سطر 5 را به اینصورت این تغییر بدهید:

```
if [ "$number" = "1" ]; then
```

اکنون وقتی پوسته بسط را انجام می‌دهد، مشاهده خواهد نمود:

```
if [ "" = "1" ]; then
```

که به طور صحیح مقصود ما را بیان می‌کند.

این مطلب مورد مهمی را طرح می‌کند که هنگام نوشتن اسکریپت‌هایتان به یاد داشته باشید. در نظر گرفتن آنکه اگر یک متغیر مساوی هیچ تنظیم شود چه پیش می‌آید.

نقل‌قول‌های مفقود

سطر 6 را با حذف کردن نقل‌قول انتهای سطر ویرایش کنید:

```
echo "Number equals 1
```

و دوباره اسکریپت را اجرا نمایید. باید این نتیجه را به دست‌بیاورید:

```
[me@linuxbox me]$ ./trouble.bash
./trouble.bash: line 8: unexpected EOF while looking for matching "
./trouble.bash: line 10 syntax error: unexpected end of file
```

اینجا هم مورد دیگری داریم که اشتباه در یک سطر موجب مشکل بعدی در اسکریپت می‌گردد. آنچه رخ می‌دهد آن است که پوسته در جستجو برای علامت نقل‌قولی است که بگوید انتهای رشته کجاست، اما قبل از پیدا کردن آن به انتهای فایل می‌رسد.

پیدا کردن این خطاها در یک اسکریپت طویل می‌تواند یک دردسر حقیقی باشد. این یکی از دلایلی است که شما موقعی که اسکریپت‌هایتان را می‌نویسید باید به طور پی‌درپی آنها را تست کنید. من همچنین احساس می‌کنم ویرایشگرهای متن با متمایز کردن (highlighting) ترکیب دستوری، پیدا کردن اینگونه اشکال‌ها را آسانتر می‌سازند.

جداسازی مشکل‌ها

پیدا کردن اشکال‌های برنامه‌های شما گاهی اوقات می‌تواند بسیار دشوار و گیج‌کننده باشد. در اینجا چند تکنیک هست که سودمند خواهد بود:

جدا کردن قطعه‌های کد به وسیله «توضیح کردن آنها». این شگرد با قرار دادن کاراکترهای توضیح در ابتدای سطرهای کد برای منع کردن پوسته از خواندن آنها انجام می‌گردد. خیلی اوقات این کار را با یک قطعه از کد انجام خواهید داد که ببینید آیا یک مشکل خاص برطرف می‌شود. با انجام این کار، شما می‌توانیدآن قسمتی از برنامه را که باعث یک مشکل می‌شود (یا نمی‌شود) مجزا کنید.

برای مثال، موقعی که ما در جستجوی نقل‌قول گمشده‌مان بودیم می‌توانستیم این کار را انجام بدهیم:

```
#!/bin/bash

number=1

if [ $number = "1" ]; then
    echo "Number equals 1
#else
#   echo "Number does not equal 1"
```

```
fi
```

با توضیح کردن عبارت **else** و اجرای اسکریپت، توانستیم نشان بدهیم که مشکل در عبارت **else** نیست ولواینکه پیغام خطا اشاره کرد که هست.

برای وارسی کردن فرضیات خود فرمانهای echo را به کار ببرید. همچنانکه در پیگردی اشکال‌ها تجربه به دست می‌آورید، کشف خواهید نمود که بیشتر اوقات اشکال‌ها در جایی که اول انتظار دارید آنها را پیدا کنید نیستند. یک مشکل معمول آن خواهد بود که شما بواسطه عملکرد برنامه‌تان یک فرض اشتباه بنا می‌کنید. شما مشاهده می‌کنید که مشکل در یک نقطه معین از برنامه ایجاد می‌شود و فرض می‌کنید که مشکل در آنجاست. به طوری که ما دیده‌ایم، بیشتر اوقات این فرض اشتباه است. برای مبارزه با این، شما باید در حالیکه اشکال‌یابی می‌کنید فرمانهای **echo** را در کدتان قرار بدهید، برای آنکه پیغام‌هایی تولید کند که تصدیق کنند برنامه آنطور عمل می‌کند که مورد انتظار است. دو نوع پیغام وجود دارد که شما باید درج کنید.

نوع اول به سادگی اعلام می‌کند که شما در برنامه به یک نقطه معین رسیده‌اید. ما این نوع را در بحث قبل‌تر خود در مورد stubbing دیدیم. این برای آگاهی از آن که جریان برنامه به روش مورد انتظارمان پیش می‌رود، مفید است.

نوع دوم مقدار متغیر (یا متغیرهای)مورد استفاده در یک محاسبه یا تست را نمایش می‌دهد. بیشتر اوقات تشخیص خواهید داد که قسمتی از برنامه شما عمل نمی‌کند زیرا چیزی که شما قبلاً در برنامه خود فرض کردید که صحیح است، در حقیقت صحیح نیست و بعداً باعث عمل نکردن برنامه شما می‌گردد.

تحت مراقبت قرار دادن اجرای اسکریپت

امکان این وجود دارد که **bash** به شما نشان بدهد موقعی که اسکریپت شما اجرا می‌شود، چه کاری در حال انجام است. برای این کار، یک "**-x**" به سطر اول اسکریپت خود اضافه کنید، مانند این:

```
#!/bin/bash -x
```

اکنون، وقتی اسکریپت را اجرا می‌کنید، **bash** هر سطر را (با بسطهای انجام شده) به طوریکه آن را اجرا می‌کند، نمایش خواهد داد. این شیوه ***tracing***(پیگردی) نامیده می‌شود. چیزی مانند این به نظر می‌رسد:

```
[me@linuxbox me]$ ./trouble.bash
+ number=1
+ '[' 1 = 1 ']'
+ echo 'Number equals 1'
Number equals 1
```

به طور جایگزین، می‌توانید فرمان **set** را در درون اسکریپت‌تان برای فعال و غیر فعال کردن پیگردی به کار ببرید. **set -x** را برای فعال کردن tracing و **set +x** را برای غیرفعال کردن آن استفاده کنید. برای مثال:

```
#!/bin/bash
```

```
number=1

set -x
if [ $number = "1" ]; then
    echo "Number equals 1"
else
    echo "Number does not equal 1"
fi
set +x
```


ورودی صفحه کلید و محاسبات

تا به حال، اسکریپت‌های ما فعل و انفعالی نبوده‌اند. یعنی، آنها به هیچ ورودی از کاربر نیاز نداشتند. در این درس، مشاهده خواهیم نمود چگونه اسکریپت‌های شما می‌توانند پرسشها را بپرسند، و پاسخ‌ها را به دست آورده استفاده کنند.

read

برای دریافت ورودی صفحه‌کلید، از فرمان **read** استفاده کنید. فرمان **read** ورودی از صفحه‌کلید را دریافت می‌کند و به یک متغیر تخصیص می‌دهد. این هم یک مثال:

```
#!/bin/bash

echo -n "Enter some text > "
read text
echo "You entered: $text"
```

به طوری که مشاهده می‌کنید، در سطر 3 ما یک اعلان نمایش دادیم. توجه کنید که "-n" ارایه شده به فرمان **echo** باعث نگاه‌داشتن اشاره‌گر در همان سطر می‌گردد، یعنی در انتهای اعلان یک تعویض سطر بیرون نمی‌دهد.

سپس، فرمان **read** را با "text" به عنوان شناسه‌اش احضار کردیم. آنچه این سطر انجام می‌دهد منتظر ماندن برای آنکه کاربر چیزی تایپ کند که با *carriage return* (کلید اینتر) دنبال شود و سپس تخصیص دادن آنچه تایپ گردیده به متغیر text است.

این هم اسکریپت در عمل است:

```
[me@linuxbox me]$ read_demo.bash
Enter some text >this is some text
```

```
You entered: this is some text
```

اگر شما به فرمان **read** نام یک متغیر را برای تخصیص ورودی‌اش ندهید، از یک متغیر محیط به نام REPLY استفاده می‌کند.

فرمان **read** چند گزینه خط فرمان نیز قبول می‌کند. دو مورد از جالب‌ترین آنها `-t` و `-s` هستند. گزینه `-t` که با یک تعداد ثانیه دنبال می‌شود یک زمان انتظار خودکار برای فرمان **read** فراهم می‌کند. این به معنای آن است که فرمان **read** اگر پس از تعداد ثانیه‌های مشخص شده پاسخی از کاربر دریافت نکرده باشد منصرف می‌شود. این گزینه در موردی می‌تواند به کار برود که یک اسکریپت حتی اگر کاربر به اعلان‌ها جواب ندهد باید (شاید با یک پاسخ ذخیره شده پیش‌فرض) ادامه بدهد. این هم گزینه `-t` در عمل:

```
#!/bin/bash

echo -n "Hurry up and type something! > "
if read -t 3 response; then
    echo "Great, you made it in time!"
else
    echo "Sorry, you are too slow!"
fi
```

گزینه `-s` باعث می‌گردد آنچه کاربر تایپ می‌کند نمایش داده نشود. این برای موقعی که شما کلمه عبور یا سایر اطلاعات محرمانه را از کاربر طلب می‌کنید مفید است.

محاسبات

چون ما روی کامپیوتر کار می‌کنیم، طبیعی است که انتظار داشته باشیم بتواند برخی محاسبات ساده را انجام بدهد. پوسته ویژگی‌هایی برای ***محاسبات صحیح*** فراهم می‌کند.

یک عدد صحیح چیست؟ به معنای تمام اعداد کاملی مانند 1, 2, 458, -2859 است. به معنای اعداد کسری مانند0.5, .333, یا 3.1415 نیست. اگر شما باید به اعداد کسری رسیدگی کنید، برنامه جداگانه‌ای به نام **bc** وجود دارد که یک زبان محاسب با دقت اختیاری را ارایه می‌کند. این برنامه می‌تواند در اسکریپت‌های پوسته به کار برود، اما فراتر از حوزه این آموزش است.

بیایید فرض کنیم شما می‌خواهید خط فرمان را به عنوان یک ماشین حساب ابتدایی به کار ببرید. می‌توانید کاری مانند این را انجام بدهید:

```
[me@linuxbox me]$ echo $((2+2))
```

چنانکه مشاهده می‌کنید، موقعی که یک عبارت حسابی را با پرانتزهای دوتایی احاطه می‌کنید، پوسته بسط حسابی انجام خواهد داد.

توجه نمایید که فضاهای سفید صرفنظر می‌شوند:

```
[me@linuxbox me]$ echo $((2+2))
4
[me@linuxbox me]$ echo $(( 2+2 ))
```

```
4
[me@linuxbox me]$ echo $(( 2 + 2 ))
4
```

پوسته می‌تواند یک طیف عملیات حسابی متعارفی (و نه غیر متعارفی) را انجام بدهد. این هم یک مثال:

```
#!/bin/bash

first_num=0
second_num=0

echo -n "Enter the first number --> "
read first_num
echo -n "Enter the second number -> "
read second_num

echo "first number + second number = $((first_num + second_num))"
echo "first number - second number = $((first_num - second_num))"
echo "first number * second number = $((first_num * second_num))"
echo "first number / second number = $((first_num / second_num))"
echo "first number % second number = $((first_num % second_num))"
echo "first number raised to the"
echo "power of the second number   = $((first_num ** second_num))"
```

ملاحظه کنید چطور "$" مقدم برای رجوع به متغیرهای داخل عبارت حسابی مانند "first_num + second_num" لازم نیست.

این برنامه را امتحان کنید و دقت نمایید با تقسیم چگونه رفتار می‌کند (به خاطر بیاورید که این تقسیمِ صحیح است) و با اعداد طویل چطور رفتار می‌کند. اعدادی که خیلی بزرگ باشند *سرریز می‌کنند* مانند کیلومترشمار در یک اتومبیل موقعی که شما از تعداد کیلومتری که برای شمارش آن تنظیم گردیده تجاوز می‌کنید. دوباره شروع می‌کند اما بواسطه چگونگی بیان کردن اعداد صحیح در حافظه، ابتدا تمام اعداد منفی را مرور می‌کند. تقسیم بر صفر (که از نظر ریاضی نامعتبر است) موجب یک خطا می‌شود.

من اطمینان دارم که شما چهار عملیات نخست به عنوان جمع، تفریق، ضرب و تقسیم را تشخیص می‌دهید، اما ممکن است آن مورد پنجم ناشناخته باشد. نماد "%" نشان‌دهنده باقیمانده است. این عمل تقسیم را انجام می‌دهد، اما به جای برگشت دادن خارج قسمت مانند تقسیم، باقیمانده را برگشت می‌دهد. در حالیکه، شاید این خیلی مفید به نظر نیاید، سودمند است، در واقع، موقع نوشتن برنامه‌ها بسیار سودمند است. برای مثال، وقتی باقیمانده یک عملیات صفر باشد، بیانگر آن است که عدد اول دقیقاً مضربی از دومی است. این مطلب می‌تواند بسیار مفید باشد:

```
#!/bin/bash

number=0
```

```
echo -n "Enter a number > "
read number

echo "Number is $number"
if [ $((number % 2)) -eq 0 ]; then
    echo "Number is even"
else
    echo "Number is odd"
fi
```

یا، در این برنامه که تعداد ثانیه‌های دلخواه را به ساعت و دقیقه قالب‌بندی می‌کند:

```
#!/bin/bash

seconds=0

echo -n "Enter number of seconds > "
read seconds

hours=$((seconds / 3600))
seconds=$((seconds % 3600))
minutes=$((seconds / 60))
seconds=$((seconds % 60))

echo "$hours hour(s) $minutes minute(s) $seconds second(s)"
```

کنترل جریان ـ بخش 2

آماده باشید غافلگیر نشوید. این درس، درس سنگینی خواهد بود!

انشعاب بیشتر

در درس قبلیِ کنترل جریان، در باره فرمان **if** و چگونگی استفاده از آن برای تغییر دادن جریان برنامه نسبت به وضعیت خروج یک فرمان، آموختیم. در اصطلاحات برنامه‌نویسی، این نوع کنترل جریان ***انشعاب*** نامیده می‌شود زیرا مانند پیمایش یک درخت است. شما در درخت به یک دوشاخه می‌رسید و ارزیابی یک شرط تعیین می‌کند به کدام شاخه بروید.

یک نوعِ دوم و پیچیده‌تر از انشعاب به نام ***case*** وجود دارد. یک ***case*** انشعاب چند شاخه است. بر خلاف انشعاب ساده که در آن شما یکی از دو مسیر ممکن را انتخاب می‌کنید، یک ***case*** چندین پیامد محتمل بر اساس ارزیابی یک کمیت را پشتیبانی می‌کند.

شما می‌توانید این نوع انشعاب را با چندین جمله **if** طرح‌ریزی کنید. در مثال پایین، ما ورودی کاربر را ارزیابی می‌کنیم:

```
#!/bin/bash

echo -n "Enter a number between 1 and 3 inclusive > "
read character
if [ "$character" = "1" ]; then
    echo "You entered one."
elif [ "$character" = "2" ]; then
    echo "You entered two."
elif [ "$character" = "3" ]; then
    echo "You entered three."
else
    echo "You did not enter a number between 1 and 3."
fi
```

خیلی دلچسب نیست.

خوشبختانه، پوسته یک راه‌حل بیشتر برازنده برای این مورد فراهم می‌کند. فرمان داخلی به نام **case** را فراهم می‌کند، که در آن می‌توانید یک برنامه هم‌ارز برنامه فوق بسازید:

```
#!/bin/bash

echo -n "Enter a number between 1 and 3 inclusive > "
```

```
read character
case $character in
    1 ) echo "You entered one."
        ;;
    2 ) echo "You entered two."
        ;;
    3 ) echo "You entered three."
        ;;
    * ) echo "You did not enter a number between 1 and 3."
esac
```

فرمان **case** دارای قالب زیر است:

```
case word in
    patterns ) commands ;;
esac
```

case به طور انتخابی جملات را در صورتیکه ***word*** با یک ***pattern*** مطابقت کند اجرا می‌کند. می‌توانید هر تعداد الگو (pattern) و جمله داشته باشید. الگوها می‌توانند متن لفظی یا کاراکتر عام باشند. می‌توانید الگوهای چندگانه که با کاراکتر " | " جدا گردیده‌اند داشته باشید. در اینجا یک مثال برای نشان دادن آنکه منظورم چیست:

```
#!/bin/bash

echo -n "Type a digit or a letter > "
read character
case $character in
                                # بررسی برای حروف
    [[:lower:]] | [[:upper:]] ) echo "You typed the letter $character"
                                ;;

                                # بررسی برای ارقام
    [0-9] )                     echo "You typed the digit $character"
                                ;;

                                # بررسی هر مورد دیگر
    * )                         echo "You did not type a letter or a digit"
esac
```

الگوی ویژه "*" را ملاحظه کنید. این الگو با هر چیزی منطبق خواهد گردید، بنابراین برای مواردی به کار می‌رود که با الگوهای قبلی مطابقت

نکرده‌اند. گنجاندن این الگو در انتها معقول است، به طوریکه می‌تواند برای تشخیص ورودی نامعتبر استفاده شود.

حلقه‌ها

آخرین نوع کنترل جریان برنامه که ما گفتگو خواهیم نمود *حلقه‌زنی* نامیده می‌شود. حلقه‌زنی اجرای تکرار شونده یک بخش از برنامه بر اساس وضعیت خروج یک فرمان است. پوسته سه فرمان برای حلقه‌زنی ارایه می‌کند: **while**، **until** و **for**. ما در این درس **while** و **until** و در درس آینده **for** را پوشش می‌دهیم.

فرمان **while** موجب می‌گردد مادامیکه که وضعیت خروج فرمانِ مشخص شده صحیح باشد، یک قطعه از کد بارها و بارها اجرا گردد. این هم یک مثال ساده از یک برنامه که از صفر تا ۹ را شمارش می‌کند:

```
#!/bin/bash

number=0
while [ "$number" -lt 10 ]; do
    echo "Number = $number"
    number=$((number + 1))
done
```

در سطر سوم، ما متغیری به نام number ایجاد کرده و مقدار صفر را به آن تخصیص می‌دهیم. سپس، حلقه **while** را شروع می‌کنیم. به طوریکه مشاهده می‌کنید، فرمانی تعیین کرده‌ایم که مقدار number را بررسی کند. ما در مثال خود بررسی می‌کنیم که آیا number دارای مقداری کمتر از 10 است.

به کلمه **do** در سطر چهارم و کلمه **done** در سطر هفتم توجه کنید. اینها قطعه کدی را که می‌خواهیم تا زمانی که وضعیت خروج صفر باقی بماند تکرار شود محصور می‌کنند.

در اکثر موارد، قطعه کدی که تکرار می‌شود باید کاری انجام بدهد که سرانجام وضعیت خروج را تغییر خواهد داد، در غیر اینصورت با موردی مواجه می‌شوید که ***حلقه بی‌پایان*** نامیده می‌شود، یعنی هرگز حلقه خاتمه نمی‌یابد.

در مثال، قطعه کد تکرار شونده مقدار number را بیرون می‌دهد (فرمان **echo** در سطر 5) و در سطر ششم number را یک واحد افزایش می‌دهد. هر نوبت که قطعه کد تکمیل می‌گردد، دوباره وضعیت خروج فرمان **]** ارزیابی می‌شود. بعد از دهمین تکرارِ حلقه، number ده مرتبه افزایش یافته است و فرمان **test** با وضعیت خروج غیرصفر خاتمه می‌یابد. در آن نقطه، جریان برنامه با جمله بعد از کلمه done ادامه می‌یابد. چون done آخرین سطر مثال ما است، برنامه پایان می‌پذیرد.

فرمان **until** دقیقاً به همان روش عمل می‌کند، غیر از آنکه، قطعه کد تا وقتی که وضعیت خروج فرمانِ تعیین شده غلط باشد تکرار می‌گردد. در مثال پایین، ملاحظه کنید که چطور عبارت داده شده به فرمان **test** برای رسیدن به همان نتیجه نسبت به مثال **while** تغییر کرده است:

```
#!/bin/bash

number=0
until [ "$number" -ge 10 ]; do
    echo "Number = $number"
```

```
    number=$((number + 1))
done
```

ساختن یک منو

یک روش رایجِ نمایش یک واسط کاربر برای برنامه‌های بر پایه متن، کاربرد یک منو است. یک منو فهرستی از انتخاب‌ها است که کاربر می‌تواند برگزیند.

در مثال زیر، ما با استفاده از دانش جدید در مورد حلقه‌ها و caseها برای ساختن یک برنامه ساده دارای منو استفاده می‌کنیم:

```
#!/bin/bash

selection=
until [ "$selection" = "0" ]; do
    echo "
    PROGRAM MENU
    1 - Display free disk space
    2 - Display free memory

    0 - exit program
"
    echo -n "Enter selection: "
    read selection
    echo ""
    case $selection in
        1 ) df ;;
        2 ) free ;;
        0 ) exit ;;
        * ) echo "Please enter 1, 2, or 0"
    esac
done
```

در این برنامه مقصود از حلقه **until** نمایش مجدد منو است، بعد از آن که یک انتخاب تکمیل گردیده باشد. حلقه تا زمانیکه انتخاب برابر "0" یعنی "exit" شود ادامه می‌یابد. ملاحظه کنید چگونه در برابر ورودی‌های کاربر که انتخاب معتبری نیستند دفاع کردیم.

برای اینکه این برنامه موقع اجرا نمای بهتری داشته باشد، می‌توانیم با افزودن یک تابع که بعد از تکمیل شدن هر انتخاب از کاربر بخواهد کلیدی را بزند، و قبل از اینکه دوباره منو نمایش داده شود صفحه نمایش را پاک کند، آن را ارتقاء بدهیم. این هم مثال ارتقاء یافته:

```
#!/bin/bash
```

```
press_enter()
{
    echo -en "\nPress Enter to continue"
    read
    clear
}

selection=
until [ "$selection" = "0" ]; do
    echo "
    PROGRAM MENU
    1 - display free disk space
    2 - display free memory

    0 - exit program
"
    echo -n "Enter selection: "
    read selection
    echo ""
    case $selection in
        1 ) df ; press_enter ;;
        2 ) free ; press_enter ;;
        0 ) exit ;;
        * ) echo "Please enter 1, 2, or 0"; press_enter
    esac
done
```

وقتی کامپیوتر شما هنگ می‌کند ...

همگی ما تجربه **هنگ کردن** برنامه کاربردی را داشته‌ایم. هنگ کردن موقعی است که به نظر می‌رسد یک برنامه به طور ناگهانی متوقف شده و غیر پاسخگو می‌شود. در حالیکه ممکن است شما تصور کنید برنامه متوقف شده است، در اکثر موارد، برنامه هنوز در حال اجرا است اما منطق برنامه در یک حلقه بی‌پایان گرفتار است.

این وضعیت را تصور کنید: شما دارای یک دستگاه خارجی هستید که به کامپیوتر شما متصل گردیده است، از قبیل یک گرداننده دیسک USB اما فراموش کرده‌اید آن را به جریان بیاندازید. شما سعی در استفاده از دستگاه می‌کنید اما به جای آن برنامه هنگ می‌کند. وقتی این مورد رخ می‌دهد، شما می‌توانید گفتگوی زیر را میان برنامه و رابط برای دستگاه مجسم نمایید:

برنامه کاربردی: آیا آماده هستی؟
رابط: دستگاه آماده نیست.

```
برنامه کاربردی:    آیا آماده هستی؟
رابط:              دستگاه آماده نیست.

برنامه کاربردی:    آیا آماده هستی؟
رابط:              دستگاه آماده نیست.

برنامه کاربردی:    آیا آماده هستی؟
رابط:              دستگاه آماده نیست.
```

و به همین ترتیب، برای همیشه.

نرم‌افزارهای خوب نوشته شده سعی می‌کنند با برقرار کردن یک ***timeout*** از این وضعیت پرهیز کنند. این به معنای آن است که حلقه تعداد کوشش‌ها را هم شمارش می‌کند و یا مدت زمان منتظر مانده برای وقوع یک مورد را محاسبه می‌کند. اگر از تعداد تلاش‌ها یا مدت زمان مجاز تجاوز شود، حلقه خارج می‌گردد و برنامه یک پیغام خطا تولید نموده و خارج می‌شود.

« کنترل جریان ـ بخش 3 فهرست مطالب کنترل جریان ـ بخش 2 »

پارامترهای مکانی

وقتی آخرین بار اسکریپت‌مان را ترک کردیم، چیزی مانند این بود:

```
#!/bin/bash

# sysinfo_page - اطلاعات سیستم HTML اسکریپتی برای تولید یک فایل

##### ثابت‌ها

TITLE="System Information for $HOSTNAME"
RIGHT_NOW=$(date +"%x %r %Z")
TIME_STAMP="Updated on $RIGHT_NOW by $USER"

##### توابع

system_info()
{
    echo "<h2>System release info</h2>"
    echo "<p>Function not yet implemented</p>"
```

```
}   #  system_info پایان تابع


show_uptime()
{
    echo "<h2>System uptime</h2>"
    echo "<pre>"
    uptime
    echo "</pre>"

}   #  show_uptime پایان تابع


drive_space()
{
    echo "<h2>Filesystem space</h2>"
    echo "<pre>"
    df
    echo "</pre>"

}   #  drive_space پایان تابع


home_space()
{
    # فقط کاربر ارشد می‌تواند این اطلاعات را به دست بیاورد

    if [ "$(id -u)" = "0" ]; then
        echo "<h2>Home directory space by user</h2>"
        echo "<pre>"
        echo "Bytes Directory"
        du -s /home/* | sort -nr
        echo "</pre>"
    fi

}   #  home_space پایان تابع



##### Main
```

```
cat <<- _EOF_
  <html>
  <head>
      <title>$TITLE</title>
  </head>
  <body>
      <h1>$TITLE</h1>
      <p>$TIME_STAMP</p>
      $(system_info)
      $(show_uptime)
      $(drive_space)
      $(home_space)
  </body>
  </html>
_EOF_
```

موارد بسیاری آماده به کار داریم، اما چند ویژگی دیگر نیز هست که من می‌خواهم اضافه کنم:

1. می‌خواهم نام فایل خروجی در خط فرمان مشخص شود، به علاوه یک نام فایل خروجی پیش‌فرض برای اینکه اگر نامی مشخص نگردد تنظیم کنم.

2. می‌خواهم یک وضعیت محاوره‌ای ارایه کنم که برای نام فایل استعلام نماید و اگر فایل موجود باشد کاربر را آگاه کرده و برای رونویسی آن نظر کاربر را جویا شود.

3. به طور طبیعی، می‌خواهم یک گزینه help نیز داشته باشیم که پیغام نحوه کاربرد را نمایش بدهد.

تمام این ویژگی‌ها، با کاربرد گزینه‌های فرمان و شناسه‌ها ارتباط دارند. برای مدیریت گزینه‌های سطر فرمان، ما یک وسیله پوسته به نام ***پارامترهای مکانی*** را به کار می‌بریم. پارامترهای مکانی یک گروه از متغیرهای خاص هستند (`$0` تا `$9`) که در بر دارنده محتویات سطر فرمان هستند.

بیایید سطر فرمان زیر را فرض کنیم:

```
[me@linuxbox me]$ some_program word1 word2 word3
```

اگر `some_program` یک اسکریپت پوسته بود، ما می‌توانستیم هر یک از اقلام سطر فرمان را بخوانیم زیرا پارامترهای مکانی محتوی این موارد می‌گردیدند:

- $0 محتوی "some_program"
- $1 محتوی "word1"
- $2 محتوی "word2"
- $3 محتوی "word3"

این هم یک اسکریپت که شما می‌توانید این مورد را آزمایش کنید:

```
#!/bin/bash

echo "Positional Parameters"
echo '$0 = ' $0
echo '$1 = ' $1
echo '$2 = ' $2
echo '$3 = ' $3
```

یافتن شناسه‌های سطر فرمان

بیشتر اوقات، شما نیاز خواهید داشت وجود شناسه‌ها را برای عمل کردن بر آنها بررسی کنید. چند روش برای انجام این کار وجود دارد. نخست، به سادگی می‌توانید بررسی کنید که آیا `$1` محتوی چیزی هست، مانند این:

```
#!/bin/bash

if [ "$1" != "" ]; then
    echo "Positional parameter 1 contains something"
else
    echo "Positional parameter 1 is empty"
fi
```

دوم، پوسته متغیری به نام `$#` را نگهداری می‌کند که شامل تعداد اقلام سطر فرمان به غیر از نام فرمان (`$0`) است.

```
#!/bin/bash

if [ $# -gt 0 ]; then
    echo "Your command line contains $# arguments"
else
    echo "Your command line contains no arguments"
fi
```

گزینه‌های سطر فرمان

همانطور که قبلاً گفتیم، بسیاری از برنامه‌ها مخصوصاً برنامه‌های پروژه گنو، از هر دو نوع گزینه خط فرمان کوتاه و بلند پشتیبانی می‌کنند. برای مثال، جهت نمایش یک پیغام راهنما برای بسیاری از این برنامه‌ها، شما می‌توانید یا از گزینه "`-h`" یا گزینه بلندتر "`--help`" استفاده کنید. نام

گزینه‌های بلند به طور معمول با دو خط تیره همراهی می‌گردد. ما در اسکریپت‌های خود این قرارداد را به کار خواهیم برد.

این هم کدی که برای پردازش سطر فرمان‌مان استفاده خواهیم نمود:

```
interactive=
filename=~/sysinfo_page.html

while [ "$1" != "" ]; do
    case $1 in
        -f | --file )           shift
                                filename=$1
                                ;;
        -i | --interactive )    interactive=1
                                ;;
        -h | --help )           usage
                                exit
                                ;;
        * )                     usage
                                exit 1
    esac
    shift
done
```

این کد کمی ماهرانه است، بنابراین هنگامی که من تلاش می‌کنم آن را تشریح نمایم بردباری کنید.

دو سطر نخست نسبتاً ساده هستند. ما متغیر `interactive` را طوری تنظیم کرده‌ایم که تهی باشد. این بیانگر آن است که وضعیت محاوره‌ای در خواست نگردیده است. سپس متغیر `filename` را برای در بر داشتن نام فایل پیش‌فرض تنظیم کردیم. اگر نام دیگری در سطر فرمان تعیین نشود، این نام فایل به کار خواهد رفت.

پس از اینکه دو متغیر تنظیم شدند، در صورتیکه کاربر هیچ گزینه‌ای به کار نبرد، ما تنظیمات پیش‌فرض را در اختیار داریم.

بعد، یک حلقه **while** می‌سازیم که در میان تمام اقلام سطر فرمان گردش می‌کنیم و هر یک را با **case** پردازش می‌کنیم. فرمان **case** هر گزینه ممکن را یافته و به طور مناسب پردازش خواهد نمود.

اکنون بخش ماهرانه. آن حلقه چگونه کار می‌کند؟ به جادوی **shift** استناد می‌کند.

shift یک فرمان داخلی است که روی پارامترهای مکانی عمل می‌کند. هر نوبت که شما فرمان **shift** را احضار می‌کنید، تمام پارامترهای مکانی را یکی به پایین جابه جا می‌کند. $2 می‌شود $1، $3 میشود $2، $4 می‌شود $3، و به همین ترتیب. این را امتحان کنید:

```
#!/bin/bash

echo "You start with $# positional parameters"
```

```
# حلقه تا موقعی که تمام پارامترها مصرف شده باشند
while [ "$1" != "" ]; do
    echo "Parameter 1 equals $1"
    echo "You now have $# positional parameters"

    # جا به جایی تمام پارامترها یکی به طرف پایین
    shift

done
```

گرفتن یک شناسهِ گزینه

گزینه "-f" ما به یک نام فایل معتبر به عنوان یک شناسه احتیاج دارد. ما دوباره **shift** را برای به دست آوردن مورد بعدی در خط فرمان و تخصیص آن به filename به کار می‌بریم. بعداً باید محتوای filename را برای حصول اطمینان از معتبر بودنش بررسی نماییم.

یکپارچه نمودن پردازش کننده سطر فرمان در اسکریپت

ما باید یک تابع نحوه استفاده اضافه کنیم و چند مورد را جابه جا کنیم تا این روال جدید را به صورت به هم پیوسته در اسکریپت‌مان داشته باشیم. همچنین مقداری کد تست برای بازبینی آنکه پردازشگر سطر فرمان به طور صحیح کار می‌کند اضافه خواهیم نمود. اکنون اسکریپت ما مانند این به نظر می‌آید:

```
#!/bin/bash

# sysinfo_page - اسکریپتی برای تولید یک فایل HTML اطلاعات سیستم

##### ثابت‌ها

TITLE="System Information for $HOSTNAME"
RIGHT_NOW=$(date +"%x %r %Z")
TIME_STAMP="Updated on $RIGHT_NOW by $USER"

##### توابع

system_info()
{
    echo "<h2>System release info</h2>"
    echo "<p>Function not yet implemented</p>"

}   #  پایان تابع system_info
```

```
show_uptime()
{
    echo "<h2>System uptime</h2>"
    echo "<pre>"
    uptime
    echo "</pre>"

}   #  show_uptime پایان تابع


drive_space()
{
    echo "<h2>Filesystem space</h2>"
    echo "<pre>"
    df
    echo "</pre>"

}   #  drive_space پایان تابع


home_space()
{
    # فقط کاربر ارشد می‌تواند این اطلاعات را به دست بیاورد

    if [ "$(id -u)" = "0" ]; then
        echo "<h2>Home directory space by user</h2>"
        echo "<pre>"
        echo "Bytes Directory"
        du -s /home/* | sort -nr
        echo "</pre>"
    fi

}   #  home_space پایان تابع


write_page()
{
    cat <<- _EOF_
    <html>
        <head>
```

```
        <title>$TITLE</title>
        </head>
        <body>
        <h1>$TITLE</h1>
        <p>$TIME_STAMP</p>
        $(system_info)
        $(show_uptime)
        $(drive_space)
        $(home_space)
        </body>
    </html>
_EOF_

}

usage()
{
    echo "usage: sysinfo_page [[[-f file ] [-i]] | [-h]]"
}



##### Main

interactive=
filename=~/sysinfo_page.html

while [ "$1" != "" ]; do
    case $1 in
        -f | --file )           shift
                                filename=$1
                                ;;
        -i | --interactive )    interactive=1
                                ;;
        -h | --help )           usage
                                exit
                                ;;
        * )                     usage
                                exit 1
    esac
    shift
done
```

```
# کد تست برای بازبینی صحت پردازش سطر فرمان

if [ "$interactive" = "1" ]; then
        echo "interactive is on"
else
        echo "interactive is off"
fi
echo "output file = $filename"


# نوشتن اسکریپت (به صورت توضیح نشان‌گذاری شده تا تست کردن کامل شود)

# write_page > $filename
```

افزودن وضعیت محاوره‌ای

وضعیت محاوره‌ای با کد پایین پیاده‌سازی می‌گردد:

```
if [ "$interactive" = "1" ]; then

    response=

    echo -n "Enter name of output file [$filename] > "
    read response
    if [ -n "$response" ]; then
        filename=$response
    fi

    if [ -f $filename ]; then
        echo -n "Output file exists. Overwrite? (y/n) > "
        read response
        if [ "$response" != "y" ]; then
            echo "Exiting program."
            exit 1
        fi
    fi
fi
```

ابتدا، بررسی می‌کنیم آیا وضعیت محاوره‌ای فعال هست، در غیر اینصورت کاری برای انجام نداریم. بعد، نام فایل را از کاربر سوال می‌کنیم. روشی را که اعلان بیان می‌شود ملاحظه کنید:

```
echo -n "Enter name of output file [$filename] > "
```

ما مقدار فعلی `filename` را نمایش می‌دهیم، چون این قسمت طوری کدنویسی می‌شود که اگر کاربر فقط کلید اینتر را بزند، مقدار پیش‌فرض `filename` استفاده خواهد شد. این مطلب در دو سطر بعدی جایی که مقدار `response` بررسی می‌شود انجام گردیده است. اگر `response` تهی نباشد، آنوقت مقدار `response` به `filename` تخصیص داده می‌شود. در غیر اینصورت `filename` تغییر نیافته با حفظ کردن مقدار پیش‌فرض آن باقی گذاشته می‌شود.

پس از اینکه دارای نام فایل خروجی هستیم، بررسی می‌کنیم که آیا از قبل موجود است. اگر باشد، به کاربر اخطار می‌دهیم. اگر پاسخ کاربر "y" نباشد، منصرف شده و خارج می‌شویم، وگرنه می‌توانیم پیش برویم.

« خطاها و سیگنالها و Trapها
فهرست مطالب
پارامترهای مکانی »

کنترل جریان ـ بخش 3

اکنون که در باره پارامترهای مکانی آموخته‌اید، وقت آن است که دستورالعمل باقیمانده کنترل جریان را پوشش بدهیم، **for**. مانند **while** و **until**، از **for** نیز برای ساختن حلقه‌ها استفاده می‌شود. **for** به این شکل کار می‌کند:

```
for variable in words; do
    commands
done
```

در اصل، **for** یک کلمه از فهرست کلمات را به متغییر مشخص شده تخصیص می‌دهد، و این کار را بارها و بارها تکرار می‌کند تا تمام کلمات مصرف بشوند. این هم یک مثال:

```
#!/bin/bash

for i in word1 word2 word3; do
    echo $i
done
```

در این مثال، رشته "word1" به متغیر `i` تخصیص داده می‌شود، سپس جمله `echo $i` اجرا می‌گردد، آنوقت به متغیر `i` رشته "word2" تخصیص داده می‌شود، و جمله `echo $i` اجرا می‌شود، و به همین ترتیب، تا تمام کلماتِ فهرست تخصیص یافته باشند.

مطلب جالب در باره **for** روشهای بسیاری است که شما می‌توانید فهرست کلمات را بسازید. تمام انواع بسطها می‌توانند به کار بروند. در مثال

بعدی، ما با استفاده از جایگزینی فرمان فهرستی از کلمات تشکیل می‌دهیم:

```
#!/bin/bash

count=0
for i in $(cat ~/.bash_profile); do
    count=$((count + 1))
    echo "Word $count ($i) contains $(echo -n $i | wc -c) characters"
done
```

در اینجا فایل `.bash_profile` را گرفته و تعداد کلمات در فایل و تعداد کاراکترهای هر کلمه را شمارش می‌کنیم.

بنابراین از پارامترهای مکانی چه استفاده‌ای می‌کنیم؟ خیلی خوب، یکی از ویژگی‌های **for** آن است که می‌تواند پارامترهای مکانی را به عنوان فهرست کلمات به کار ببرد:

```
#!/bin/bash

for i in "$@"; do
    echo $i
done
```

متغیر پوسته "$@" شامل لیستی از شناسه‌های سطر فرمان است. این شیوه بیشتر مواقع برای پردازش لیستی از فایلها در سطر فرمان استفاده می‌شود.

این هم یک مثال دیگر:

```
#!/bin/bash

for filename in "$@"; do
    result=
    if [ -f "$filename" ]; then
        result="$filename is a regular file"
    else
        if [ -d "$filename" ]; then
            result="$filename is a directory"
        fi
    fi
    if [ -w "$filename" ]; then
        result="$result and it is writable"
    else
        result="$result and it is not writable"
```

```
    fi
    echo "$result"
done
```

این اسکریپت را آزمایش کنید. برای دیدن آنکه کار می‌کند لیستی از فایلها یا یک کاراکتر عام مانند "*" به آن بدهید.

این هم یک نمونه اسکریپت دیگر. این یکی فایلهای داخل دو دایرکتوری را مقایسه می‌کند و فایلهایی از دایرکتوری اول را که در دایرکتوری دوم وجود ندارند لیست می‌کند.

```
#!/bin/bash

# cmp_dir - برنامه‌ای برای مقایسه دو دایرکتوری

# بررسی شناسه‌های مورد نیاز
if [ $# -ne 2 ]; then
    echo "usage: $0 directory_1 directory_2" 1>&2
    exit 1
fi

# کسب اطمینان برای آنکه هر دو شناسه دایرکتوری باشند
if [ ! -d $1 ]; then
    echo "$1 is not a directory!" 1>&2
    exit 1
fi

if [ ! -d $2 ]; then
    echo "$2 is not a directory!" 1>&2
    exit 1
fi

# directory_2 و مقایسه آن با  directory_1 پردازش هر فایل در
missing=0
for filename in $1/*; do
    fn=$(basename "$filename")
    if [ -f "$filename" ]; then
        if [ ! -f "$2/$fn" ]; then
            echo "$fn is missing from $2"
            missing=$((missing + 1))
        fi
    fi
done
```

```
echo "$missing files missing"
```

اکنون ما در جهت کار واقعی می‌خواهیم تابع home_space در اسکریپت‌مان را برای بیرون دادن اطلاعات بیشتر بهینه‌سازی کنیم. به یاد می‌آورید که نگارش قبلی چیزی مانند این بود:

```
home_space()
{
    # فقط کاربر ارشد می‌تواند این اطلاعات را به دست بیاورد

    if [ "$(id -u)" = "0" ]; then
    echo "<h2>Home directory space by user</h2>"
    echo "<pre>"
    echo "Bytes Directory"
        du -s /home/* | sort -nr
    echo "</pre>"
    fi

}   #  home_space پایان تابع
```

این هم نگارش جدید:

```
home_space()
{
    echo "<h2>Home directory space by user</h2>"
    echo "<pre>"
    format="%8s%10s%10s   %-s\n"
    printf "$format" "Dirs" "Files" "Blocks" "Directory"
    printf "$format" "----" "-----" "------" "---------"
    if [ $(id -u) = "0" ]; then
        dir_list="/home/*"
    else
        dir_list=$HOME
    fi
    for home_dir in $dir_list; do
        total_dirs=$(find $home_dir -type d | wc -l)
        total_files=$(find $home_dir -type f | wc -l)
        total_blocks=$(du -s $home_dir)
        printf "$format" $total_dirs $total_files $total_blocks
```

```
    done
    echo "</pre>"

}   #  پایان تابع home_space
```

این نگارش بهینه‌سازی شده یک فرمان جدید **printf** معرفی می‌کند، که برای تولید خروجی قالب‌بندی شده مطابق یک ***قالب رشته*** به کار می‌رود. **printf** از زبان برنامه‌نویسی C شروع شده و در بسیاری زبانهای برنامه‌نویسی دیگر از جمله C++، Perl، awk، java، PHP، و البته، bash پیاده سازی گردیده است. در باره قالب‌های رشته **printf** می‌توانید در این آدرس‌ها بیشتر بخوانید:

- GNU Awk User's Guide - Control Letters
- GNU Awk User's Guide - Format Modifiers

همچنین فرمان **find** را معرفی می‌کنیم. **find** برای جستجوی فایلها یا دایرکتوریهایی که با ضوابط معینی مطابقت دارند به کار می‌رود. در تابع home_space، ما فرمان **find** را برای لیست کردن دایرکتوریها و فایلهای معمولی در هر دایرکتوری خانگی به کار می‌بریم. با استفاده از فرمان **wc**، تعداد فایلها و دایرکتوریهای یافت شده را شمارش می‌کنیم.

مورد واقعاً جالب در باره home_space چگونگی رسیدگی کردن به مشکل دستیابی کاربر ارشد است. شما توجه خواهید داشت که ما با **id** برای کاربر ارشد بررسی می‌کنیم و مطابق خروجی از تست، رشته‌های متفاوتی به متغیر dir_list تخصیص می‌دهیم، که لیست کلمات برای حلقه **for** متعاقب آن می‌گردند. به این طریق، اگر یک کاربر دلخواه اسکریپت را اجرا کند، تنها دایرکتوری خانگی او لیست خواهد گردید.

یک تابع دیگر که می‌تواند از یک حلقه **for** استفاده کند، تابع ناتمام system_info ما است. می‌توانیم آن را به این شکل بسازیم:

```
system_info()
{
    #  /etc پیدا کردن هر فایل release در

    if ls /etc/*release 1>/dev/null 2>&1; then
        echo "<h2>System release info</h2>"
        echo "<pre>"
        for i in /etc/*release; do

            # چون نمی‌توانیم از طول فایل مطمئن باشیم،
            #           فقط سطر اول را نمایش می‌دهیم.

            head -n 1 $i
        done
        uname -orp
        echo "</pre>"
    fi

}   #  پایان تابع system_info
```

در این تابع، ما نخست تعیین می‌کنیم که آیا هیچ فایل release برای پردازش وجود دارد. فایلهای release شامل نام فروشنده و نگارش توزیع هستند. آنها در دایرکتوری /etc قرار داده می‌شوند. برای یافتن آنها، ما یک فرمان **ls** اجرا می‌کنیم و تمام خروجی‌اش را دور می‌اندازیم. فقط وضعیت خروج را می‌خواهیم. که اگر فایلی پیدا شده باشد صحیح خواهد بود.

بعد، ما HTML برای این قسمت از صفحه را ایجاد می‌کنیم، چون اکنون می‌دانیم که فایلهای release برای پردازش وجود دارند. برای پردازش فایلها، یک حلقه **for** را جهت عمل کردن روی هر کدام آغاز می‌نماییم. در داخل حلقه، ما از فرمان **head** برای باز گرداندن سطر اول هر فایل استفاده می‌کنیم.

سرانجام، فرمان **uname** را با گزینه‌های "o"، "r"، و "p" جهت به دست آوردن برخی اطلاعات اضافی سیستم به کار می‌بریم.

« خطاها و سیگنالها و Trapها 2 | فهرست مطالب | کنترل جریان ـ بخش 3 »

خطاها و سیگنالها و Trapها (ای وای!) ـ بخش 1

در این درس، می‌خواهیم به مدیریت خطاها در خلال اجرای اسکریپت‌هایتان نگاه کنیم.

تفاوت میان یک برنامه خوب و یک برنامه ضعیف بیشتر اوقات بر حسب ***نیرومندی*** برنامه اندازه‌گیری می‌شود. یعنی توانایی برنامه در مدیریت موقعیت‌هایی که چیزی درست کار نمی‌کند.

وضعیت خروج

همچنانکه از درس‌های قبل به یاد دارید، هر برنامه خوب نوشته شده هنگامی که پایان می‌پذیرد یک وضعیت خروج برگشت می‌دهد. اگر برنامه به طور موفقیت‌آمیز به پایان برسد، وضعیت خروج صفر خواهد بود. اگر وضعیت خروج مورد دیگری غیر از صفر باشد، آنوقت برنامه به طریقی مردود گردیده است.

بررسی نمودن وضعیت خروج برنامه‌هایی که شما در اسکریپت‌هایتان احضار می‌کنید بسیار با اهمیت است. همچنین اهمیت دارد که اسکریپت‌های شما هنگامی که به پایان می‌رسند یک وضعیت خروج معنادار برگشت بدهند. من یکبار مدیر سیستم یونیکسی داشتم که او یک اسکریپت برای یک سیستم تولید شامل دو سطر کد پایین نوشت:

```
# مثالی از یک ایده حقیقتاً نامساعد

cd $some_directory
rm *
```

چرا انجام چنین موردی یک روش نامساعد است؟ اگر هیچ عدم موفقیتی نباشد، نادرست نیست. این دو سطر دایرکتوری کاری را به دایرکتوری نام برده شده در $some_directory تغییر می‌دهد و فایلها در آن دایرکتوری را حذف می‌کند. این رفتارِ مورد انتظار است. اما اگر دایرکتوری نامبرده در $some_directory موجود نباشد چه می‌شود؟ در آن حالت، فرمان **cd** ناموفق می‌گردد و اسکریپت فرمان **rm** را در دایرکتوری کاری جاری اجرا می‌کند. رفتار مورد انتظار نیست!

در ضمن، اسکریپت مدیر سیستم بیچاره من به این نارساییِ زیاد اجازه داد و قسمت بزرگی از یک سیستم تولید را تخریب نمود. اجازه ندهید این

مورد برای شما رخ بدهد!

مشکل به وسیله اسکریپت آن بود که قبل از پیشروی با فرمان **rm** وضعیت خروج فرمان **cd** را بررسی نکرد.

بررسی کردن وضعیت خروج

چند روش وجود دارد که می‌توانید وضعیت خروج یک برنامه را به دست آورده و به آن واکنش نشان بدهید. نخست، شما می‌توانید محتوای متغیر محیط $? را معاینه کنید. $? محتوی وضعیت خروج آخرین فرمان اجرا شده خواهد بود. با کد زیر کارکرد آن را می‌توانید مشاهده کنید:

```
[me] $ true; echo $?
0
[me] $ false; echo $?
1
```

فرمانهای **true** و **false** برنامه‌هایی هستند که غیر از اینکه به ترتیب یک وضعیت خروج صفر و یک برگشت بدهند کاری انجام نمی‌دهند. با کاربرد آنها، ما می‌توانیم مشاهده کنیم چگونه متغیر دارای وضعیت خروجِ برنامه قبل می‌گردد.

بنابراین برای بررسی وضعیت خروج، می‌توانستیم اسکریپت را به این طریق بنویسیم:

```
# بررسی وضعیت خروج

cd $some_directory
if [ "$?" = "0" ]; then
        rm *
else
        echo "Cannot change directory!" 1>&2
        exit 1
fi
```

در این نگارش، ما وضعیت خروج فرمان **cd** را معاینه می‌کنیم و اگر صفر نباشد، یک پیغام خطا در خروجی استاندارد خطا چاپ نموده و اسکریپت را با وضعیت خروج 1 خاتمه می‌دهیم.

در حالیکه این یک راه حل کاری برای مشکل است، روشهای چابک‌تری که مقداری در تایپ کردن ما صرفه‌جویی خواهد نمود وجود دارند. در رویکرد بعدی ما سعی می‌کنیم به طور مستقیم از جمله `if` استفاده کنیم، چون وضعیت خروج فرمان داده شده را ارزیابی می‌کند.

با کاربرد `if`، می‌توانستیم آنرا به این شکل بنویسیم:

```
# یک روش بهتر

if cd $some_directory; then
        rm *
```

```
else
        echo "Could not change directory! Aborting." 1>&2
        exit 1
fi
```

در اینجا ما بررسی می‌کنیم ببینیم آیا فرمان cd موفق است. تنها آنوقت فرمان **rm** مجاز به اجرا می‌گردد، در غیر اینصورت یک پیغام خطا صادر و برنامه با وضعیت خروج 1 نشان‌دهنده آنکه یک خطا رخ داده است، خارج می‌شود.

یک تابع خروج خطا

چون بیشتر اوقات ما در برنامه‌هایمان خطاها را بررسی خواهیم نمود، نوشتن یک تابع که پیغام خطاها را نمایش خواهد داد، عقلانی است. این تابع از تایپ کردن بیشتر صرفه‌جویی می‌کند و تنبلی را رواج می‌دهد.

```
# یک تابع خروج خطا

error_exit()
{
        echo "$1" 1>&2
        exit 1
}

#  error_exit طرز استعمال

if cd $some_directory; then
        rm *
else
        error_exit "Cannot change directory!  Aborting."
fi
```

لیست‌های AND و OR

سر انجام، ما می‌توانیم اسکریپت‌هایمان را با عملگرهای کنترل AND و OR بیشتر ساده کنیم. برای روشن کردن چگونگی کار آنها، من از صفحه man **bash** نقل‌قول خواهم نمود:

> «عملگرهای کنترل && و || به ترتیب علامت لیست‌های ANDو لیست‌های ORهستند.یک لیست AND به این شکل است
>
> command1 && command2
>
> command2 در صورتی اجرا می‌گردد که اگر ***و فقط اگر***، command1 یک وضعیت خروج صفر برگشت بدهد.

یک لیست OR به این شکل است

```
command1 || command2
```

command2 در صورتی اجرا می‌شود که اگر و فقط اگر، command1 یک وضعیت خروج غیر صفر برگشت بدهد. وضعیت خروج لیست‌های AND و OR وضعیت خروج فرمان اجرا شده در لیست است.»

یک بار دیگر، ما می‌توانیم از فرمانهای **true** و **false** برای دیدن این کار استفاده کنیم:

```
[me] $ true || echo "echo executed"
[me] $ false || echo "echo executed"
echo executed
[me] $ true && echo "echo executed"
echo executed
[me] $ false && echo "echo executed"
[me] $
```

با استفاده از این شیوه، حتی می‌توانیم یک نگارش ساده‌تر بنویسیم:

```
# ساده‌تر از همه

cd $some_directory || error_exit "Cannot change directory! Aborting"
rm *
```

اگر در مورد خطا یک خروج لازم نباشد، آنوقت شما حتی می‌توانید این مورد را انجام بدهید:

```
# یک روش دیگر برای انجام آن در صورتیکه خارج شدن مطلوب نیست

cd $some_directory && rm *
```

می‌خواهم اشاره کنم که حتی با دفاع در برابر خطاها که ما در مثال‌هایمان برای استفاده از **cd** معرفی کرده‌ایم، بازهم این کد به واسطه یک خطای رایج برنامه‌نویسی آسیب‌پذیر است، یعنی، اگر نام متغیر شامل نام دایرکتوری با املای غلط نوشته شود چه موردی رخ می‌دهد؟ در آن حالت، پوسته متغیر را به عنوان متغیر تهی تفسیر می‌کند و **cd** موفق می‌شود، اما دایرکتوری به دایرکتوری خانگی کاربر تغییر می‌کند، بنابراین مواظب باشید!

بهینه‌سازی تابع خروج خطا

تعدادی بهبود وجود دارد که می‌توانیم برای تابع error_exit به وجود آوریم. من دوست دارم نام برنامه را برای روشن کردن اینکه خطا از کجا ناشی می‌شود پیوست کنم. این مطلب وقتی برنامه شما پیچیده‌تر می‌شود و دارای اسکریپت‌های راه‌اندازی کننده سایر اسکریپت‌ها باشید بیشتر اهمیت

می‌یابد. همچنین، به گنجاندن متغیر محیط LINENO توجه نمایید که به شما کمک می‌کند سطری از اسکریپت را که خطا در آن رخ داده است، به طور دقیق شناسایی کنید.

```
#!/bin/bash

# یک روال مدیریت خطای ماهرتر

#       من متغیری به نام PROGNAME در اسکریپت خود قرار می‌دهم که
#      نام برنامه‌ای که باید اجرا شود را نگهداری می‌کند. شما این
# کمیت را می‌توانید از اولین فقره سطر فرمان ($0) به دست بیاورید.

PROGNAME=$(basename $0)

error_exit()
{

#       ----------------------------------------
#      تابع برای خروج ناشی از خطای مهلک برنامه
#                          یک شناسه می‌پذیرد:
#             رشته شامل پیغام خطای توصیف کننده
#       ----------------------------------------


        echo "${PROGNAME}: ${1:-"Unknown Error"}" 1>&2
        exit 1
}

# مثال فراخوانی تابع error_exit به گنجاندن متغیر محیطِ
# LINENO توجه کنید. این متغیر محتوی شماره سطر جاری است


echo "Example of error with line number and message"
error_exit "$LINENO: An error has occurred."
```

استفاده از ابروها در داخل تابع error_exit یک مثال از ***بسط پارامتر*** است. در صورتیکه لازم است مطمئن باشید که نام متغیر از متن اطراف جدا می‌شود، می‌توانید آن را با ابروها احاطه کنید (مانند ${PROGNAME}). بعضی اشخاص طبق عادت آنها را پیرامون هر متغیری قرار می‌دهند. آن کاربرد در واقع یک مورد سلیقه‌ای است. در کاربرد دوم، ${1:-"Unknown Error"} به معنای این است که اگر parameter 1 یعنی $1 تعریف نشده است، رشته "Unknown Error" در محل آن جایگزین بشود. انجام تعدادی از دستکاری‌های سودمند رشته‌ای، با استفاده از بسط پارامتر مقدور است. در باره بسط پارامتر می‌توانید در صفحه مستندات **bash** تحت عنوان "EXPANSIONS" بیشتر بخوانید.

خطاها و سیگنالها و Trapها (ای وای!) ـ بخش 2

خطاها تنها طریقه‌ای نیستند که یک اسکریپت می‌تواند به طور غیرمنتظره خاتمه بیابد. شما باید به سیگنال‌ها نیز رسیدگی کنید. برنامه زیر را در نظر بگیرید:

```
#!/bin/bash

echo "this script will endlessly loop until you stop it"
while true; do
        : # Do nothing
done
```

بعد از اینکه این اسکریپت را راه‌اندازی کنید به ظاهر هنگ خواهد کرد. در واقع، مانند اکثر برنامه‌هایی که هنگ کرده به نظر می‌آیند، حقیقتاً در داخل یک حلقه گرفتار می‌شود. در این مورد، منتظر است تا فرمان **true** یک وضعیت خروج غیر صفر برگشت بدهد، که هرگز چنین کاری نمی‌کند. وقتی شروع شود، اسکریپت ادامه خواهد داشت تا bash سیگنالی دریافت کند که آن را متوقف خواهد نمود. شما می‌توانید چنین سیگنالی را با تایپ کردن Ctrl-c ارسال کنید که سیگنالی به نام SIGINT است (کوتاه شده SIGnal INTerrupt).

پاکسازی بعدی خودتان

بسیار خوب، بنابراین یک سیگنال می‌تواند از راه برسد و اسکریپت شما را خاتمه بدهد. سیگنال چرا اهمیت دارد؟ خوب، در بسیاری از موارد اهمیت ندارد و شما می‌توانید از سیگنالها صرفنظر کنید، اما در برخی موارد اهمیت خواهد داشت.

بیایید به یک اسکریپت دیگر نگاه کنیم:

```
#!/bin/bash

# برنامه‌ای برای چاپ یک فایل متن با سرایندها و پانویسها

TEMP_FILE=/tmp/printfile.txt

pr $1 > $TEMP_FILE

echo -n "Print file? [y/n]: "
read
if [ "$REPLY" = "y" ]; then
        lpr $TEMP_FILE
fi
```

این اسکریپت یک فایل متن تعیین شده در سطر فرمان را با فرمان **pr** پردازش می‌کند و نتیجه را در یک فایل موقت ذخیره می‌کند. بعداً، از کاربر سوال می‌کند که آیا می‌خواهد فایل چاپ شود. اگر کاربر "y" را تایپ نماید، آنوقت فایل موقت را برای چاپ به برنامه **lpr** عبور می‌دهد. (اگر شما در عمل دارای چاپگر متصل به سیستم خود نیستید می‌توانید **less** را جایگزین **lpr** نمایید.)

اکنون، من قبول دارم که این اسکریپت دارای مشکلات طراحی بسیار است. در حالیکه به یک نام فایل ارایه شده در سطر فرمان احتیاج دارد، بررسی نمی‌شود که آیا یک نام ارایه شده، و بررسی نمی‌کند که فایل در واقع موجود است. اما مشکلی که من می‌خواهم بر آن تمرکز نمایم این حقیقت است که وقتی اسکریپت خاتمه می‌یابد، فایلِ موقت را پشت سرش باقی می‌گذارد.

شیوه مناسب، حکم کردن به آن خواهد بود که وقتی اسکریپت خاتمه می‌یابد فایل $TEMP_FILE حذف بشود. این کار به آسانی با افزودن کد زیر به انتهای اسکریپت انجام می‌شود:

```
rm $TEMP_FILE
```

به نظر می‌رسد با این کد مشکل برطرف خواهد شد، اما اگر موقعی که اعلان "Print file? [y/n]:" ظاهر می‌گردد کاربر ctrl-c را تایپ کند چه پیش می‌آید؟ اسکریپت در فرمان **read** پایان می‌پذیرد و فرمان **rm** هرگز اجرا نمی‌گردد. به طور واضح، ما به واکنش در برابر سیگنالهایی همچون SIGINT هنگامی که ctrl-c تایپ می‌شود نیاز داریم.

خوشبختانه، bash یک شیوه برای انجام فرمانها در هنگامی که سیگنالها دریافت می‌شوند ارایه می‌کند.

trap

فرمان **trap** اجرای یک فرمان در موقعی که یک سیگنال توسط اسکریپت دریافت می‌شود را میسر می‌سازد. این فرمان به این شکل کار می‌کند:

```
trap arg signals
```

"signals" یک فهرست از سیگنالها برای گوش سپردن به آنها و "arg" یک فرمان است برای آنکه وقتی یکی از سیگنالها دریافت می‌شود اجرا گردد. برای اسکریپت چاپ خود، ما می‌توانیم مشکل سیگنال را به این طریق اداره کنیم:

```
#!/bin/bash

# برنامه‌ای برای چاپ یک فایل متن با سرایندها و پانویسها

TEMP_FILE=/tmp/printfile.txt

trap "rm $TEMP_FILE; exit" SIGHUP SIGINT SIGTERM

pr $1 > $TEMP_FILE

echo -n "Print file? [y/n]: "
read
```

```
if [ "$REPLY" = "y" ]; then
        lpr $TEMP_FILE
fi
rm $TEMP_FILE
```

در اینجا ما یک فرمان **trap** اضافه کردیم که اگر هر کدام از سیگنالهای لیست شده دریافت بشود فرمان "**rm $TEMP_FILE**" را اجرا خواهد نمود. سه سیگنال لیست شده، رایج‌ترین مواردی هستند که شما مواجه خواهید شد، اما تعداد بسیار بیشتری وجود دارد که می‌توانند تعیین بشوند. برای فهرست کامل، فرمان "**trap -l**" را تایپ کنید. علاوه بر فهرست کردن سیگنالها به وسیله نام آنها، ممکن است به طور جایگزین آنها را با شماره آنها مشخص کنید.

سیگنال شماره 9 از فضای بیرونی

یک سیگنال وجود دارد که شما نمی‌توانید trap کنید: SIGKILL یا سیگنال 9. کرنل هر پردازشی را که این سیگنال به آن ارسال شده بلافاصله خاتمه می‌دهد و هیچ مدیریت سیگنالی انجام نمی‌شود. چون این سیگنال همیشه یک برنامه گرفتار، هنگ، یا به طور دیگری فرو ریخته را خاتمه می‌دهد، وسوسه‌انگیز است به این فکر کنید که وقتی شما باید موردی را متوقف کرده و رهایش کنید، این راه آسانی برای خروج است. بیشتر اوقات مراجعه به فرمان زیر را که سیگنال SIGKILL ارسال می‌کند، مشاهده می‌کنید:

```
kill -9
```

در هر حال، با وجود اینکه ظاهراً آسان است، شما باید به یاد داشته باشید که وقتی این سیگنال را ارسال می‌کنید، هیچ پردازشی به وسیله برنامه انجام نمی‌شود. بیشتر اوقات این خوب است، اما برای بسیاری از برنامه‌ها اینطور نیست. مخصوصاً، بسیاری از برنامه‌های پیچیده (و برخی نه چندان پیچیده) برای جلوگیری از اجرای همزمان چند نمونه از برنامه ***فایلهای قفل*** تولید می‌کنند. وقتی به برنامه‌ای که از یک فایل قفل استفاده می‌کند یک سیگنال SIGKILL فرستاده می‌شود، موقع خاتمه یافتن امکان پاک کردن قفل را به دست نمی‌آورد. وجود فایل قفل از دوباره راه‌اندازی برنامه تا زمان حذف فایل قفل به طور دستی، پیش‌گیری خواهد نمود.

آگاه باشید. سیگنال SIGKILL را به عنوان آخرین چاره استفاده کنید.

یک تابع پاکسازی

در حالیکه فرمان **trap** مشکل را برطرف نموده است، می‌توانیم مشاهده کنیم که محدودیت‌هایی هم دارد. به طور مهمتر از همه، تنها یک رشته منفرد شامل فرمانی که موقع دریافت سیگنال باید اجرا بشود، قبول می‌کند. شما می‌توانستید زرنگی کنید و از "**;**" استفاده کرده، فرمانهای چندگانه را برای تحصیل رفتار پیچیده‌تر در رشته قرار بدهید، اما به طور صریح، ناخوشایند و زشت است. یک روش مناسب‌تر ایجاد یک تابع خواهد بود که وقتی شما می‌خواهید عملیاتی در انتهای اسکریپت خود انجام بدهید فراخوانی می‌گردد. من در اسکریپت‌هایم، به این تابع نام clean_up می‌دهم.

```
#!/bin/bash

# برنامه‌ای برای چاپ یک فایل متن با سرایندها و پانویسها

TEMP_FILE=/tmp/printfile.txt
```

```
clean_up() {

        # انجام برنامه خانه‌تکانی خروج
        rm $TEMP_FILE
        exit
}

trap clean_up SIGHUP SIGINT SIGTERM

pr $1 > $TEMP_FILE

echo -n "Print file? [y/n]: "
read
if [ "$REPLY" = "y" ]; then
        lpr $TEMP_FILE
fi
clean_up
```

استفاده از یک تابع پاکسازی ایده خوبی برای روالهای مدیریت خطای شما نیز هست. در هر صورت، وقتی برنامه شما خاتمه می‌یابد (به هر دلیل)، شما باید خودتان بعداً پاکسازی کنید. در اینجا نگارش تکمیل شده برنامه ما با مدیریت خطا و سیگنال بهبودیافته:

```
#!/bin/bash

# برنامه‌ای برای چاپ یک فایل متن با سرایندها و پانویسها

# Usage: printfile file

# ایجاد یک نام فایل موقتی که به دایرکتوری tmp محلی کاربر تنظیم می‌شود
#     و دارای یک نام است که در برابر حملات به فایلهای موقت مقاوم است.

if [ -d "~/tmp" ]; then
        TEMP_DIR=~/tmp
else
        TEMP_DIR=/tmp
fi
TEMP_FILE=$TEMP_DIR/printfile.$$.$RANDOM
PROGNAME=$(basename $0)

usage() {
```

```
        # نمایش پیغام نحوه کاربرد در خروجی استاندارد خطا
        echo "Usage: $PROGNAME file" 1>&2
}

clean_up() {

        # انجام برنامه خانه‌تکانی خروج
        # به طور اختیاری یک وضعیت خروج قبول می‌کند
        rm -f $TEMP_FILE
        exit $1
}

error_exit() {

        # نمایش پیغام خطا و خروج
        echo "${PROGNAME}: ${1:-"Unknown Error"}" 1>&2
        clean_up 1
}

trap clean_up SIGHUP SIGINT SIGTERM

if [ $# != "1" ]; then
        usage
        error_exit "one file to print must be specified"
fi
if [ ! -f "$1" ]; then
        error_exit "file $1 cannot be read"
fi

pr $1 > $TEMP_FILE || error_exit "cannot format file"

echo -n "Print file? [y/n]: "
read
if [ "$REPLY" = "y" ]; then
        lpr $TEMP_FILE || error_exit "cannot print file"
fi
clean_up
```

تولید بدون خطر فایلهای موقت

در برنامه فوق، اقداماتی برای کمک به محفوظ داشتن فایل موقتِ مورد استفاده در اسکریپت به عمل آمده. در یونیکس استفاده از یک دایرکتوری به نام /tmp برای قرار دادن فایلهای موقتی مورد استفاده برنامه‌ها، یک قرارداد است. هر کسی می‌تواند فایلها را در این دایرکتوری بنویسد. این مورد به طور طبیعی به برخی نگرانی‌های امنیتی منجر می‌گردد. در صورت امکان، از نوشتن فایلها در دایرکتوری /tmp پرهیز کنید. شیوه ترجیح یافته، نوشتن آنها در دایرکتوری محلی از قبیل دایرکتوری ~/tmp (یک دایرکتوری فرعی در دایرکتوری خانگی کاربر) است. در صورتیکه شما باید فایلها را در دایرکتوری /tmp بنویسید، باید اقداماتی برای اطمینان از غیر قابل پیش‌بینی بودن نام فایلها انجام بدهید. نام فایلهای قابل پیش‌بینی اجازه می‌دهند یک مهاجم پیوندهای نمادین به فایلهایی ایجاد کند که خود می‌خواهد شما بازنویسی کنید.

یک نام فایل خوب به شما کمک خواهد نمود معین کنید در فایل چه چیزی نوشته شده، اما به طور کامل قابل پیش‌بینی نخواهد بود. در اسکریپت فوق، سطر پایین از کد، فایل موقت $TEMP_FILE را ایجاد کرد:

```
TEMP_FILE=$TEMP_DIR/printfile.$$.$RANDOM
```

متغیر $TEMP_DIR بر اساس دسترس‌پذیری دایرکتوری، محتوی یکی از دایرکتوری‌های /tmp یا ~/tmp است. جاسازی نام برنامه در نام فایل شیوه رایجی است . ما این کار را با رشته "printfile" انجام داده‌ایم. سپس، از متغیر $$ پوسته برای جاسازی شماره شناسایی پردازش (pid) برنامه استفاده می‌کنیم. این به شناسایی آنکه کدام برنامه عهده‌دار فایل است بیشتر کمک می‌کند. به طور تعجب‌آور، شماره شناسایی پردازش به تنهایی برای ایمن کردن فایل به اندازه کافی غیرقابل پیش‌بینی نیست، بنابراین ما متغیر $RANDOM پوسته را برای پیوست کردن یک عدد تصادفی به نام فایل، اضافه می‌کنیم. با این شیوه یک نام فایل ایجاد می‌کنیم که هم قابل شناسایی و هم غیر قابل پیش‌بینی است.

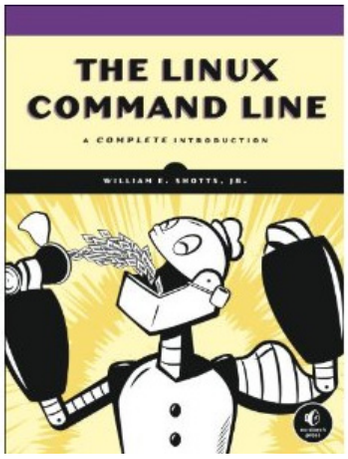
پایان سخن

این مطلب آموزشهای ***LinuxCommand.org*** را به پایان می‌رساند. من صمیمانه امیدوارم که آنها را هم سودمند و هم خوشایند یافته باشید. اگر چنین بود، ماجراجویی خط فرمان خود را با دانلود کتاب من ادامه بدهید.

The Linux Command Line

کتابی از William Shotts

جدید! اکنون ویرایش دوم اینترنتی در دسترس است!



برای کاربران جدید خط فرمان در نظر گرفته شده، این کتاب در ۵۳۷ صفحه همان مطالب LinuxCommand.org را پوشش می‌دهد اما با جزییات بسیار بیشتر. علاوه بر مبانی استفاده از خط فرمان و اسکریپت‌نویسی پوسته، ***The Linux Command Line*** شامل فصل‌هایی در باره بسیاری از برنامه‌های رایج مورد استفاده در خط فرمان، و نیز مباحث پیشرفته بیشتر است.

- **منتشره تحت مجوز** Creative Commons**، این کتاب در قالب PDF برای دریافت به طور آزاد در دسترس است.** آن را از اینجا دریافت کنید.
- **کتاب همچنین به صورت چاپی در دسترس است، توسط انتشارات** No Starch Press **چاپ گردیده.** در کتابفروشی‌های معتبر به فروش می‌رسد. همچنین No Starch Press نسخه‌های الکترونیکی آن را با e-readerهای محبوب ارایه می‌کند.
- **آن را در** کتابخانه منطقه‌تان **پیدا کنید.**

خوانندگان در باره ***The Linux Command Line*** چه می‌گویند!

«من کمی بیش از یک سال است که از لینوکس استفاده می‌کنم. تا جایی که می‌توانستم خواندم ازجمله Rute، و بسیاری دیگر. من در ۹۳ صفحه اول ِکتاب شما بیش از تمام آنها یاد گرفته‌ام!!!!»

«کتاب شما چه کمک بزرگی برای من بوده است! من در جستجوی کتابی مانند کتاب شما بودم، اما هرگز موفق نشدم موردی را پیدا کنم که خواننده را به طور کامل در استفاده کلی خط فرمان لینوکس همراهی نماید، شما موردی را سراغ دارید؟ اگر اشتباه نکنم، کتابهای زیادی به آموزش اسکریپت‌نویسی پوسته پرداخته‌اند، اما هیچ کدام برای استفاده منظم خط فرمان تنظیم نگردیده‌اند.»

«کتاب ِخارق‌العاده!! من اخیراً از سیستم عامل موروثی به لینوکس مهاجرت کرده‌ام و هراسان از کاربرد ترمینال در تلاش برای یافتن توزیعی بوده‌ام که رفتار آن سیستم‌عامل قدیمی را تقلید کند. من با این کتاب روبرو شدم و برای اولین بار ترمینال را باز کردم. این کتاب هرآنچه را که شما نیاز دارید در باره پوسته بدانید به شما آموزش می‌دهد و این کار را با سهولت انجام می‌دهد. با دادن یک پایه قوی به شما شروع می‌کند و با بنا کردن بر آن ادامه می‌دهد. سادگی و ساختار آموزنده آن برای تمام کاربران جدیدی که به لینوکس مهاجرت می‌کنند ایده‌آل است. حالا من همواره حتی برای ساده‌ترین امور در ترمینال هستم. این نه فقط علاقه ایجاد می‌کند، بلکه من اسکریپت‌نویسی پوسته را هم تمرین می‌کنم. کتاب ِعالی!!»

«من از خواندن کتاب لذت برده‌ام و مطالب بسیاری از هر فصل آموخته‌ام. نوشته شما بسیار واضح است، و من از دنبال کردن مثالهای شما لذت

بردم. من قبلاً چند کتاب لینوکس را به طور سطحی خواندم، اما هرگز وقت کافی که واقعاً برای آنها صرف کنم نداشتم. اما به واسطه آن تجربیات، فکر می‌کنم کتاب شما برای نوآموزان بسیار واضح‌تر و قابل فهم‌تر است.»

«وای! چه کتاب عالی. به طور آشکار تکیه‌گاه مستحکمی برای تمام آنهایی که می‌خواهند به طور تدریجی خودشان را از عادت به GUI رها کنند فراهم می‌نماید، یا شاید «دشوار را امکان پذیر می‌کند».»

«من فکر می‌کنم یکی از مهمترین دارایی‌های کتاب که اغلب کتابهای دیگر در باره لینوکس ندارند، لحن محاوره‌ای آن است. من احساس می‌کنم تقریباً در هر فصل، شما خواننده را به جویا شدن پرسش‌هایی از موادی که لزوماً پوشش داده نشده‌اند هدایت می‌کنید.»

نظرات بیشتر را در No Starch Press و Amazon بخوانید.

منابع

موارد محبوب از بلاگ

- Building An All-text Linux Workstation

یک مجموعه ۱۴ قسمتی که تولید یک ایستگاه کاری همه منظوره‌ای را که فقط برنامه‌های بر مبنای متن را به کار می‌برد، شرح می‌دهد.

- Getting Ready For Ubuntu 10.04

این مجموعه فرایند ارتقای یک سیستم لینوکس به وسیله انجام یک fresh install از یک نگارش به یک نگارش دیگر را مرور می‌کند. شیوه‌های پشتیبان‌گیری و نصب سیستم را پوشش می‌دهد.

- New Features In Bash 4.x

ویژگی‌های اضافه شده به چهارمین نگارش عمده bash را پوشش می‌دهد.

اسکریپت‌های پوسته

- **new_script**

یک تولید کننده الگوی اسکریپت پوسته bash. این اسکریپت را برای کمک به نوشتن اسکریپت‌های خودتان به کار ببرید. الگوهای تولید شده شامل توابع سودمند پوسته، مدیریت خطا و سیگنال، و تجزیه‌کننده شناسه و گزینه هستند.

- **my_cloud**

یک سیستم ابتدایی ذخیره ابری را با استفاده از هر میزبان راه‌دورِ در حال اجرا به عنوان یک خادم SSH، پیاده‌سازی می‌کند.

- **photo2mail**

فایلهای بزرگ تصویر (عکس‌ها) را برای استفاده به عنوان پیوست‌های پیغام‌های ایمیل، یادداشت‌های بلاگ، و غیره تغییر اندازه می‌دهد.

- **program_list**

یک لیست تشریحی از برنامه‌های موجود در یک دایرکتوری تولید می‌کند. مناسب برای سیاحت نمودن در سیستم شما.

سایت‌های دیگری که می‌توانید بهره‌مند گردید

Bash و اسکریپت‌نویسی

- The Bash Reference Manual

شاید در پاسخ به قابل استفاده نمودن موضوعات یافت شده در صفحه man bash، پروژه GNU راهنمای مرجع Bash را تولید کرده. شما می‌توانید آن را به عنوان ترجمه صفحه man bash به قالب قابل خواندن انسانی در نظر بگیرید. در حالیکه فاقد یک ن‌گرش آموزشی است و مثال‌های کاربردی را شامل نمی‌شود، برای خواندن بسیار آسان‌تر است و به‌گونه‌ای مفیدتر از صفحه man bash تنظیم گردیده است.

- Greg's Wiki

صفحه man bash و راهنمای مرجع Bash هر دو به طور وسیع ویژگی‌های موجود در bash را مستندسازی می‌کنند. اما، هنگامی که خواستار یک توصیف از رفتار bash هستیم، منابع متفاوتی لازم می‌گردند. مورد به مراتب شایسته‌تر Greg's Wiki است. این سایت مباحث متنوعی را پوشش می‌دهد، اما موارد مخصوصاً ارزشمند برای ما Bash FAQ که شامل بیش از یکصد پرسش در باره bash است که بارها پرسیده شده‌اند، Bash Pitfalls که بسیاری از مشکلات رایج رویاروی نویسندگان اسکریپت در bash را تشریح می‌کند، و Bash Guide، یک مجموعه آموزش‌های سودمند برای کاربران bash هستند. همچنین چند مورد مفرح rants برای خواندن وجود دارد.

- Bash Hacker's Wiki

همانند ویکی Greg، ویکی Bash Hackers گفتارهای مختلف بسیاری در ارتباط با bash، ویژگی‌ها و رفتار آن ارایه می‌کند. موارد گنجانده شده، تعدادی آموزش‌های مفید در زمینه شیوه‌های برنامه‌نویسی و موضوعات اسکریپت‌نویسی bash هستند. در حالیکه نوشتن در آن به طور گاه‌گاهی، و اندکی آشفته و بی‌نظم است، شامل اطلاعات سودمندی می‌باشد.

- Chet Ramey's Bash Page

Chet Ramey نگهدارنده فعلی bash است و او دارای صفحه خودش می‌باشد. در آن صفحه شما می‌توانید اطلاعات نگارش، آخرین اخبار، و سایر موارد را پیدا کنید. سودمندترین سند روی صفحه Bash نگارش Bash FAQ آن است. فایل NEWS شامل لیست فشرده‌ای از ویژگی های اضافه شده در هر نگارش bash است.

- Heiner's SHELLdorado

SHELLdorado یک سایت آموزشی اسکریپت‌نویسی پوسته است. آموزش‌هایی از مباحث متنوع اسکریپت‌نویسی و یک مجموعه بزرگ از اسکریپت‌ها و توابع را ارایه می‌دهد.

- Advanced Bash Scripting HOWTO

این یک نگاه تفصیلی به برنامه‌نویسی پوسته bash است. برخوردی اندیشمندانه همراه با مثال‌ها و تمرین‌های بسیار زیاد. یک مکمل مناسب برای آموزش‌های LinuxCommand.org.

- The Bash Prompt HOWTO

صفحه Bash Prompt HOWTO یک کنکاش جالب شگفت‌انگیزی است در مورد هر آنچه که شما می‌توانید با prompt انجام بدهید.

- Linux From Scratch

با آموزش The Linux From Scratch شما فرایند ساختن یک سیستم کامل لینوکس از کد منبع را طی خواهید نمود.

استانداردهای لینوکس

- The Linux Filesystem Hierarchy Standard

استاندارد سیستم فایل سلسله مراتبی لینوکس، طرح‌بندی استاندارد یک سیستم لینوکس را شرح می‌دهد. در صورتیکه می‌خواهید یاد بگیرید که چرا و چگونه لینوکس به طریقی که هست تنظیم می‌گردد، سایت مناسبی برای بازدید شما است.

- Linux Standard Base

Linux Standard Base یک پروژه اختصاصی برای توسعه استاندارد توزیع‌های لینوکس است.

جنگنده برای آزادی شما

- Free Software Foundation

بنیاد نرم‌افزار آزاد (FSF) به توسعه و ترویج حقوق استفاده، مطالعه، رونوشت برداری، ویرایش، و توزیع مجدد برنامه‌های کامپیوتری توسط کاربران اختصاص دارد.

- Electronic Frontier Foundation

EFF صرف نظر از خودِ تکنولوژی، برای محافظت از حقوق اساسی ما، جهت آموزش دادن مطبوعات، سیاست‌گذاران، و آحاد مردم پیرامون موضوعات آزادی‌های شخصی مرتبط با تکنولوژی کار می‌کند، و به عنوان یک مدافع آن آزادی‌ها عمل می‌نماید.

- Software Freedom Law Center

SFLC نمایندگی قانونی و سایر خدمات مرتبط با قانون را برای محافظت و پیشبرد نرم‌افزار آزاد و منبع‌باز فراهم می‌کند. در سال 2005, تاسیس گردید، اکنون این مرکز بسیاری از مهمترین و ماندگارترین پروژه‌های موفق منبع‌باز و آزاد را نمایندگی می‌کند.

- Against Monopoly

Against Monopoly وبلاگی است که نوشته‌های اقتصاد دانان و صاحبنظران نظریه بازی[1] را که مخالف مفاهیم گوناگون «مالکیت فکری» هستند نمایان می‌کند. حتماً کتاب آزاد و قابل دریافت *Against Intellectual Monopoly* نوشته Michele Boldrin و David K. Levine را بخوانید.

---

1. **مترجم:** نظریه بازی‌ها یک نظریه ریاضی منتسب به John von Neumann است که خط مشی و احتمال در شرایط رقابتی بازیها را بررسی می‌کند، که در آن شرایط تمام بازیگران دارای کنترل جهت‌دار هستند و هر کدام در جستجوی سودمندترین حرکات در ارتباط با سایرین می‌باشند. اکنون این نظریه شاخه‌ای از ریاضیات کاربردی است که موقعیت‌های استراتژیکی را مطالعه می‌کند که در آن موقعیت‌ها نقش‌آفرینان هر کدام اقدامات متفاوتی را برای حداکثرسازی منافع خود اتخاذ می‌نمایند.
این نظریه نخست به عنوان ابزاری برای درک رفتار اقتصادی ایجاد شد. در آغاز دهه70 میلادی برای توضیح رفتار حیوانی، شامل انواع و تکامل در اثر انتخاب طبیعی، به کار برده شد. در سالهای جنگ سرد به علت کاربردش در راهبرد نظامی، مخصوصاً مفهوم انهدام مسلم و قطعی دوجانبه، شاهد بیشترین رشد خود بود.
امروزه نظریه بازی در حوزه‌های آکادمیک بسیار متنوع از زیست‌شناسی تا فلسفه به کار می‌رود. اخیراً به علت کاربردش در هوش مصنوعی و مطالعه قیاسی مغز و سلسله اعصاب با ماشین‌های الکتریکی و مکانیکی، توجه علوم کامپیوتر را نیز به سمت خود جلب نموده است. [1]

new_script

این یک تولید کننده الگوی اسکریپت پوسته (یعنی اسکریپتی که اسکریپت‌ها را می‌نویسد) است. این اسکریپت برای ایجاد قسمت‌های استاندارد یک اسکریپت پوسته به کار می‌رود. چند پرسش در باره اسکریپتی که تولید می‌شود از کاربر سوال می‌کند و سپس اسکریپت حاصل را در یک فایل می‌نویسد. اسکریپت‌های تولید شده توسط `new_script` شامل روال‌های مدیریت خطا و سیگنال، یک تجزیه کننده گزینه و شناسه سطر فرمان، و یک help اصولی خط فرمان هستند. آموزش‌های کامل و مثال‌های آن را می‌توانید در اینجا پیدا کنید.

```
#!/bin/bash
# ------------------------------------------------------------------------
# new_script - Bash تولید کننده الگوی اسکریپ پوسته

# Copyright 2012, William Shotts <bshotts@users.sourceforge.net>

#  این برنامه نرم‌افزار آزاد است: شما می‌توانید آن را تحت شرایط نگارش
#    شماره 3 مجوز  GPL گنو یا به انتخاب خودتان هر نگارش پس از آن که
#  توسط بنیاد نرم‌افزارهای آزاد اعلام گردیده، توزیع و یا ویرایش کنید.

# این برنامه به امید آنکه سودمند باشد توزیع می‌گردد، اما بدون هرگونه
#  تعهد حتی تعهد ضمنی کیفیت یا صلاحیت برای یک مقصود خاص. برای جزییات
#  .ببینید (http://www.gnu.org/licenses/)  بیشتر مجوز GPL گنو را در

# new_script [-h|--help] [-q|--quiet] [-s|--root] [script] : نحوه کاربرد

# تاریخچه بازبینی:
# 2014-01-21  اصلاحات قالب‌بندی جزیی (3.2)
# 2014-01-12     تمیزکاری‌های مختلف (3.1)
# 2012-05-14     تولید شد
# ------------------------------------------------------------------------

PROGNAME=${0##*/}
VERSION="3.2"
SCRIPT_SHELL=${SHELL}

# ایجاد رشته‌های خوش‌نمای تاریخ
DATE=$(date +'%Y-%m-%d')
YEAR=$(date +'%Y')
```

```
# passwd دریافت نام واقعی کاربر از فایل
AUTHOR=$(awk -v USER=$USER \
  'BEGIN { FS = ":" } $1 == USER { print $5 }' < /etc/passwd)

# ساختن آدرس ایمیل از نام میزبان یا متغیر محیط REPLYTO اگر مشخص شده باشد
EMAIL_ADDRESS="<${REPLYTO:-${USER}@$HOSTNAME}>"

# تعریف آرایه‌ها برای گزینه‌ها و گزینه‌شناسه‌های سطر فرمان
declare -a opt opt_desc opt_long opt_arg opt_arg_desc


clean_up() { # انجام خانه‌تکانی قبل از خروج
  return
}

error_exit() {
  echo -e "${PROGNAME}: ${1:-"Unknown Error"}" >&2
  clean_up
  exit 1
}

graceful_exit() {
  clean_up
  exit
}

signal_exit() { # اداره سیگنال‌های trap شده
  case $1 in
    INT)
      error_exit "Program interrupted by user" ;;
    TERM)
      echo -e "\n$PROGNAME: Program terminated" >&2
      graceful_exit ;;
    *)
      error_exit "$PROGNAME: Terminating on unknown signal" ;;
  esac
}

usage() {
```

```
  echo "Usage: ${PROGNAME} [-h|--help ] [-q|--quiet] [-s|--root] [script]"
}

help_message() {
  cat <<- _EOF_
  ${PROGNAME} ${VERSION}
  Bash shell script template generator.

  $(usage)

  Options:

  -h, --help    Display this help message and exit.
  -q, --quiet   Quiet mode. No prompting. Outputs default script.
  -s, --root    Output script requires root privileges to run.

_EOF_
}

insert_license() {

  if [[ -z $script_license ]]; then
    echo "# All rights reserved."
    return
  fi
  cat <<- _EOF_

# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.

# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License at <http://www.gnu.org/licenses/> for
# more details.
_EOF_
}
```

```
insert_usage() {

  echo -e "usage() {\n  echo \"$usage_str\"\n}"
}

insert_help_message() {

  local arg i long

  echo -e "help_message() {"
  echo -e "  cat <<- _EOF_"
  echo -e "  \$PROGNAME ver. \$VERSION"
  echo -e "  $script_purpose"
  echo -e "\n  \$(usage)"
  echo -e "\n  Options:"
  i=0
  while [[ ${opt[i]} ]]; do
    long=
    arg=
    [[ ${opt_long[i]} ]] && long=", --${opt_long[i]}"
    [[ ${opt_arg[i]} ]] && arg=" ${opt_arg[i]}"
    echo -e "  -${opt[i]}$long$arg  ${opt_desc[i]}"
    [[ ${opt_arg[i]} ]] && \
      echo -e "    Where '${opt_arg[i]}' is the ${opt_arg_desc[i]}."
    ((++i))
  done
  [[ $root_mode ]] && \
    echo -e "\n  NOTE: You must be the superuser to run this script."
  echo -e "\n  _EOF_"
  echo -e "  return\n}"
}

insert_root_check() {

  if [[ $root_mode ]]; then
    echo -e "# Check for root UID"
    echo -e "if [[ \$(id -u) != 0 ]]; then"
    echo -e "  error_exit \"You must be the superuser to run this script.\""
    echo -e "fi"
  fi
```

```
}

insert_parser() {

  local i

  echo -e "while [[ -n \$1 ]]; do\n  case \$1 in"
  echo -e "    -h | --help)\n      help_message; graceful_exit ;;"
  for (( i = 1; i < ${#opt[@]}; i++ )); do
    echo -ne "    -${opt[i]}"
    [[ -n ${opt_long[i]} ]] && echo -ne " | --${opt_long[i]}"
    echo -ne ")\n      echo \"${opt_desc[i]}\""
    [[ -n ${opt_arg[i]} ]] && echo -ne "; shift; ${opt_arg[i]}=\"\$1\""
    echo " ;;"
  done
  echo -e "    -* | --*)\n      usage"
  echo -e "      error_exit \"Unknown option \$1\" ;;"
  echo -e "    *)\n      echo \"Argument \$1 to process...\" ;;"
  echo -e "  esac\n  shift\ndone"
}

write_script() {

################################################################################
#        آغاز الگوی اسکریپت
################################################################################
cat << _EOF_
#!$SCRIPT_SHELL
# ------------------------------------------------------------------------------
# $script_name - $script_purpose

# Copyright $YEAR, $AUTHOR $EMAIL_ADDRESS
$(insert_license)

# Usage: $script_name$usage_message

# Revision history:
# $DATE Created by $PROGNAME ver. $VERSION
# ------------------------------------------------------------------------------
```

```
PROGNAME=\${0##*/}
VERSION="0.1"

clean_up() { # Perform pre-exit housekeeping
  return
}

error_exit() {
  echo -e "\${PROGNAME}: \${1:-"Unknown Error"}" >&2
  clean_up
  exit 1
}

graceful_exit() {
  clean_up
  exit
}

signal_exit() { # Handle trapped signals
  case \$1 in
    INT)
      error_exit "Program interrupted by user" ;;
    TERM)
      echo -e "\n\$PROGNAME: Program terminated" >&2
      graceful_exit ;;
    *)
      error_exit "\$PROGNAME: Terminating on unknown signal" ;;
  esac
}

usage() {
  echo -e "Usage: \$PROGNAME$usage_message"
}

$(insert_help_message)

# Trap signals
trap "signal_exit TERM" TERM HUP
trap "signal_exit INT"  INT
```

```
$(insert_root_check)

# Parse command-line
$(insert_parser)

# Main logic

graceful_exit

_EOF_
################################################################################
#        پایان الگوی اسکریپت
################################################################################

}

check_filename() {

  local filename=$1
  local pathname=${filename%/*} # Equals filename if no path specified

  if [[ $pathname != $filename ]]; then
    if [[ ! -d $pathname ]]; then
      [[ $quiet_mode ]] || echo "Directory $pathname does not exist."
      return 1
    fi
  fi
  if [[ -n $filename ]]; then
    if [[ -e $filename ]]; then
      if [[ -f $filename && -w $filename ]]; then
        [[ $quiet_mode ]] && return 0
        read -p "File $filename exists. Overwrite [y/n] > "
        [[ $REPLY =~ ^[yY]$ ]] || return 1
      else
        return 1
      fi
    fi
  else
    [[ $quiet_mode ]] && return 0 # در وضعیت quiet نام فایل تهی قبول است
    return 1
```

```
  fi
}

read_option() {

  local i=$((option_count + 1))

  echo -e "\nOption $i:"
  read -p "Enter option letter [a-z] (Enter to end) > "
  [[ -n $REPLY ]] || return 1 # نگذاشتن عضو آرایه در صورتیکه REPLY تهی باشد
  opt[i]=$REPLY
  read -p "Description of option ------------------> " opt_desc[i]
  read -p "Enter long alternate name (optional) ----> " opt_long[i]
  read -p "Enter option argument (if any) ----------> " opt_arg[i]
  [[ -n ${opt_arg[i]} ]] && \
  read -p "Description of argument (if any)---------> " opt_arg_desc[i]
  return 0 # صرف نظر از نتیجه حاصل از بررسی فوق 0 اجبار به برگشت دادن
}

#       سیگنال‌های Trap
trap "signal_exit TERM" TERM HUP
trap "signal_exit INT"  INT

#       تجزیه سطر فرمان
quiet_mode=
root_mode=
script_license=
while [[ -n $1 ]]; do
  case $1 in
    -h | --help)
      help_message; graceful_exit ;;
    -q | --quiet)
      quiet_mode=yes ;;
    -s | --root)
      root_mode=yes ;;
    -* | --*)
      usage; error_exit "Unknown option $1" ;;
    *)
      tmp_script=$1; break ;;
  esac
```

```
  shift
done

#       منطق اصلی

if [[ $quiet_mode ]]; then
  script_filename="$tmp_script"
  check_filename "$script_filename" || \
    error_exit "$script_filename is not writable."
  script_purpose="[Enter purpose of script here.]"
else
       # دریافت نام اسکریپت
  script_filename=
  while [[ -z $script_filename ]]; do
    if [[ -n $tmp_script ]]; then
      script_filename="$tmp_script"
      tmp_script=
    else
      read -p "Enter script output filename: " script_filename
    fi
    if ! check_filename "$script_filename"; then
      echo "$script_filename is not writable."
      echo -e "Please choose another name.\n"
      script_filename=
    fi
  done

  #     هدف اسکریپت
  read -p "Enter purpose of script: " script_purpose

  #      اجازه نامه
  read -p "Include GPL license header [y/n]? > "
  [[ $REPLY =~ ^[yY]$ ]] && script_license="GPL"

  #      آیا به مزایای کاربر ارشد نیاز است؟
  read -p "Does this script require superuser privileges [y/n]? "
  [[ $REPLY =~ ^[yY]$ ]] && root_mode="yes"

  #     گزینه‌های سطر فرمان
  option_count=0
```

```
  read -p "Does this script support command-line options [y/n]? "
  [[ $REPLY =~ ^[yY]$ ]] \
    && while read_option; do ((++option_count)); done
fi

script_name=${script_filename##*/} # جدا کردن مسیر از نام فایل
script_name=${script_name:-"[Untitled Script]"} # اگر تهی باشد تنظیم به نام پیشفرض

#       گزینه "help" گنجانده شده به طور پیشفرض
opt[0]="h"
opt_long[0]="help"
opt_desc[0]="Display this help message and exit."

#       ایجاد پیغام نحوه کاربرد
usage_message=
i=0
while [[ ${opt[i]} ]]; do
  arg="]"
  [[ ${opt_arg[i]} ]] && arg=" ${opt_arg[i]}]"
  usage_message="$usage_message [-${opt[i]}"
  [[ ${opt_long[i]} ]] \
    && usage_message="$usage_message|--${opt_long[i]}"
  usage_message="$usage_message$arg"
  ((++i))
done

#       تولید اسکریپت
if [[ $script_filename ]]; then # نوشتن اسکریپت در فایل
  write_script > "$script_filename"
  chmod +x "$script_filename"
else
  write_script # نوشتن اسکریپت در خروجی استاندارد
fi
graceful_exit
```

my_cloud

اگر شما به یک میزبان راه‌دورِ اجرا کننده یک سرویس‌دهنده پوسته امن(ssh) دسترسی دارید، این اسکریپت یک «Dropbox عصر حجر» فراهم می‌کند. برای استفاده از این اسکریپت شما به یک دایرکتوری به نام `cloud` در داخل دایرکتوری خانگی خود روی میزبان راه‌دور نیاز دارید. آنوقت می‌توانید فایل‌ها را به دایرکتوری `cloud` روی میزبان راه‌دور کپی(قرار بدهید)، از آن کپی(دریافت کنید)، فایلهای آن را لیست کرده، و یا حذف نمایید.

مثال‌ها

```
me@linuxbox ~ $ my_cloud -c me@remotehost -l
```

فایل‌های دایرکتوری `cloud` روی میزبان راه‌دور را لیست می‌کند.

```
me@linuxbox ~ $ my_cloud -c me@remotehost -p .bashrc
```

فایل .bashrc روی میزبان محلی را به دایرکتوری `cloud` در میزبان راه‌دور کپی می‌کند.

```
me@linuxbox ~ $ my_cloud -c me@remotehost -g my_file
```

فایل my_file را از دایرکتوری `cloud` روی میزبان راه‌دور به دایرکتوری جاری در میزبان محلی کپی می‌کند.

```
me@linuxbox ~ $ my_cloud -c me@remotehost -d my_file
```

فایل my_file را از دایرکتوری `cloud` روی میزبان راه‌دور حذف می‌نماید.

توجه نمایید که اسکریپت فقط فایل‌ها میان سیستم‌ها رامنتقل می‌کند، نه دایرکتوری‌ها را کپی می‌کند، و نه همچون یک سرویس‌گیرنده واقعی ذخیره ابری هم‌گام سازی یک دایرکتوری با دایرکتوری دیگر را فراهم می‌کند.

نکته

برای بهبودبخشی به سهولت استفاده، می‌توانید یک مستعار به فایل .bashrc خودتان اضافه کنید برای آنکه نیاز به مشخص کردن ***user@host*** جهت میزبان‌ها را که بارها استفاده می‌کنید برطرف نمایید:

```
alias mc_remotehost='my_cloud -c me@remotehost'
```

```
#!/bin/bash
# ------------------------------------------------------------------------
# my_cloud - ذخیره و بازیابی فایل‌ها روی سرویس‌دهنده راه‌دور

# Copyright 2013, William Shotts <bshotts@users.sourceforge.com>
```

```
# این برنامه نرم‌افزار آزاد است: شما می‌توانید آن را تحت شرایط نگارش
#    شماره 3 مجوز GPL گنو یا به انتخاب خودتان هر نگارش پس از آن که
#  توسط بنیاد نرم‌افزارهای آزاد اعلام گردیده، توزیع و یا ویرایش کنید.

# این برنامه به امید آنکه سودمند باشد توزیع می‌گردد، اما بدون هرگونه
#  تعهد حتی تعهد ضمنی کیفیت یا صلاحیت برای یک مقصود خاص. برای جزییات
#  بیشتر مجوز GPL گنو را در (http://www.gnu.org/licenses/) ببینید.

#       پیش‌نیازها:
# my_cloud انتظار دارد که میزبان‌های راه‌دور دارای یک دایرکتوری
#          به نام cloud در داخل دایرکتوری خانه کاربر باشند.

#       نحوه کاربرد:
# my_cloud -h|--help
# my_cloud -c|--cloud user@host -l|--list
# my_cloud -c|--cloud user@host -g|--get file...
# my_cloud -c|--cloud user@host -p|--put file...
# my_cloud -c|--cloud user@host -d|--delete file...

# تاریخچه بازبینی:
# 2013-12-30  ایجاد گردید new_script اسکریپت 3.0 به وسیله نگارش
# ------------------------------------------------------------------------

PROGNAME=${0##*/}
VERSION="0.1"

clean_up() { # انجام خانه‌تکانی قبل از خروج
  return
}

error_exit() { # مدیریت خطای مهلک
  echo -e "${PROGNAME}: ${1:-"Unknown Error"}" >&2
  clean_up
  exit 1
}

graceful_exit() { # خروج عادی
  clean_up
  exit
```

```
}

signal_exit() { # اداره سیگنال‌های trap شده
  case $1 in
    INT)    error_exit "Program interrupted by user" ;;
    TERM)   echo -e "\nnew_script: Program terminated" >&2 ; graceful_exit ;;
    *)      error_exit "new_script: Terminating on unknown signal" ;;
  esac
}

usage() {
  echo -e "Usage: $PROGNAME [-h ]|[-c user@host [-l]|[-g|-p|-d file...]]"
}

help_message() {
  cat <<- _EOF_
  $PROGNAME ver. $VERSION
  Store and retrieve files on a remote server

  $(usage)

  Options:
  -h, --help              Display this help message and exit.
  -c, --cloud user@host   Remote server login, where 'user@host'
                          is the login name and host.
  -l, --list              List files on remote server
  -g, --get file...       Get file(s) from remote server
  -p, --put file...       Put file(s) on remote server
  -d, --delete file...    Delete file(s) on remote server

_EOF_
  return
}

set_mode() {
  if [[ $mode == "empty" ]]; then
    mode=$1
  else
    error_exit "Only one mode (-l, -g, -p, -d) is allowed."
  fi
```

```
}

#        Trap سیگنال‌های
trap "signal_exit TERM" TERM HUP
trap "signal_exit INT"  INT

#        تجزیه سطر فرمان
mode=empty
file_list=
while [[ -n $1 ]]; do
  case $1 in
    -h | --help)    help_message; graceful_exit ;;
    -c | --cloud)   shift; user_host="$1" ;;
    -l | --list)    set_mode list ;;
    -g | --get)     set_mode get ;;
    -p | --put)     set_mode put ;;
    -d | --delete)  set_mode delete ;;
    -* | --*)       usage; error_exit "Unknown option $1" ;;
    *)  # پردازش شناسه‌ها
        case $mode in
          get) file_list="$file_list $user_host:cloud/$1"
            ;;
          put) [[ -f "$1" ]] && file_list="$file_list $1"
            ;;
          delete) file_list="$file_list $1"
            ;;
        esac
        ;;
  esac
  shift
done

# منطق اصلی

# خروج در صورتیکه میزبان راه‌دور تعیین نشده باشد
[[ -n "$user_host" ]] || error_exit "You must specify a user@host (-c)."
host=${user_host##*@}

case $mode in
  list )  echo -e "\n### Files on host ${host}: ###"
```

```
    ssh $user_host 'ls -lA cloud'
    ;;
  get ) echo -e "\n### Getting $file_list from host $host ###"
    scp -p $file_list .
    ;;
  put ) echo -e "\n### Putting $file_list on host $host ###"
    scp -p $file_list $user_host:cloud
    ;;
  delete ) echo -e "\n### Deleting $file_list from host $host ###"
    ssh $user_host "cd cloud && rm $file_list"
    ;;
esac
graceful_exit
```

صفحه اول
بخش منابع »

photo2mail

امروزه دوربین‌های دیجیتالی به تولید فایلهای بزرگ تصویرها گرایش دارند. فایلهایی که برای پیوست نمودن به پیغام‌های الکترونیکی یا سایر اشکال مخابره اینترنتی نامناسب هستند. این اسکریپت باخواندن یک یا چندفایل تصویر و سپس نوشتن فایل‌های جدید با اندازه مناسب‌تر این مشکل را حل می‌کند. این کار به کمک برنامه **convert** فراهم شده با بسته **ImageMagick** انجام می‌شود.

مثال‌ها

```
me@linuxbox ~ $ photo2mail image.jpg
```

در ساده‌ترین شکل، **photo2mail** فایل image.jpg را می‌خواند و یک فایل جدید تصویر به نام image-1024.jpg در همان دایرکتوری اصلی ایجاد می‌کند.

```
me@linuxbox ~ $ photo2mail -s 800 image.jpg
```

به طور پیش‌فرض، اسکریپت تصویرها را طوری تغییر اندازه می‌دهد که برای محدود شدن در داخل یک جعبه چهارگوش به اندازه 1024 پیکسل مناسب شود. سایر اندازه‌ها با استفاده از گزینه -s ، می‌توانند تعیین بشوند.

```
me@linuxbox ~ $ photo2mail -d ../resized image.jpg
```

با استفاده از گزینه -d ، می‌توان برای خروجی یک دایرکتوری جایگزین تعیین نمود.

```
me@linuxbox ~ $ photo2mail -j image.png
```

به طور پیش‌فرض، **photo2mail** تصویر تغییر یافته را با همان قالب فایل اصلی می‌نویسد. گزینه -j برنامه **photo2mail** را وادار به نوشتن یک فایل JPEG می‌نماید.

```
#!/bin/bash
# ----------------------------------------------------------------------
# photo2mail - تغییر اندازه تصویرها برای پیوست به ایمیل

# Copyright 2013, William Shotts <bshotts@users.sourceforge.net>

# این برنامه نرم‌افزار آزاد است: شما می‌توانید آن را تحت شرایط نگارش
#    شماره 3 مجوز GPL گنو یا به انتخاب خودتان هر نگارش پس از آن که
# توسط بنیاد نرم‌افزارهای آزاد اعلام گردیده، توزیع و یا ویرایش کنید.

# این برنامه به امید آنکه سودمند باشد توزیع می‌گردد، اما بدون هرگونه
# تعهد حتی تعهد ضمنی کیفیت یا صلاحیت برای یک مقصود خاص. برای جزییات
# بیشتر مجوز GPL گنو را در (http://www.gnu.org/licenses/) ببینید.
```

```
#  photo2mail [-h|--help] [--options] file... : نحوه کاربرد

#       گزینه‌ها :
# -d, --directory dir   دایرکتوری برای تصویرهای خروجی، دایرکتوری
#                              پیش‌فرض همان دایرکتوری مبدا می‌باشد.
# -j, --jpeg                اجبار به این‌که تصویر خروجی صرف‌نظر از
#                              قالب تصویر مبدا در قالب JPEG باشد.
# -s, --size size         اندازه محدوده جعبه تصویر خروجی. اندازه
#                                        پیش‌فرض 1024  پیکسل است.

#   این برنامه فایل‌های تغییر اندازه داده شده تصویرها را با استفاده از
#   برنامه convert از بسته ImageMagick تولید می‌کند. تصویرهای خروجی در
#   همان دایرکتوری تصویرهای اصلی ایجاد می‌گردند و برای آسان شناخته شدن
#  دارای اندازه تصویر پیوست شده در نام‌هایشان هستند. شناسه size اندازه
#      اندازه عرض به پیکسل، مجدوده جعبه چهارگوش دربرگیرنده تصویر است.

# تاریخچه بازبینی:
# 2014-01-12  تمیزکاری‌های بیشتر (1.2)
# 2014-01-04  تمیزکاری‌های مختلف (1.1)
# 2013-01-11  به وسیله نگارش 3.0.1 اسکریپت new_script ایجاد گردید
# ------------------------------------------------------------------------

PROGNAME=${0##*/}
VERSION="1.2"
DEFAULT_SIZE=1024
REQUIRED_PROGS="identify convert"

clean_up() { # انجام خانه‌تکانی قبل از خروج
  return
}

error_exit() {
  echo -e "${PROGNAME}: ${1:-"Unknown Error"}" >&2
  clean_up
  exit 1
}

error_warning() {
  echo -e "${PROGNAME}: Warning - $1" >&2
  return
```

```
}

graceful_exit() {
  clean_up
  exit
}

signal_exit() { # اداره سیگنال‌های trap شده
  case $1 in
    INT)    error_exit "Program interrupted by user" ;;
    TERM)   echo -e "\n$PROGNAME: Program terminated" >&2
            graceful_exit ;;
    *)      error_exit "$PROGNAME: Terminating on unknown signal" ;;
  esac
}

usage() {
  echo -e "Usage: $PROGNAME [-h|--help] [--options] file..."
}

help_message() {
  cat << _EOF_
$PROGNAME ver. $VERSION
Resize images for use as email attachments

$(usage)

Options:
-d, --directory dir Directory for output images. Default
                    is same directory as source.
-j, --jpeg          Force output image to be JPEG regardless
                    of source image format.
-s, --size size     Size of output image bounding box. Default
                    is 1024 pixels.

_EOF_
  return
}

#       سیگنال‌های Trap
```

```
trap "signal_exit TERM" TERM HUP
trap "signal_exit INT"  INT

size=$DEFAULT_SIZE
f_ext=
f_path=

#       تجزیه سطر فرمان
while [[ -n $1 ]]; do
  case $1 in
    -d | --directory) shift; f_path="$1" ;;
    -h | --help)      help_message; graceful_exit ;;
    -j | --jpeg)      f_ext="jpg" ;;
    -s | --size)      shift; size="$1" ;;
    -* | --*)         usage; error_exit "Unknown option $1" ;;
    *)                break ;;
  esac
  shift
done

#       منطق اصلی

# Check validity of options
[[ "$size" =~ ^[0-9]+$ ]] \
  || error_exit "output size must be an integer."
[[ -z "$f_path" || -d "$f_path" ]] \
  || error_exit "output directory '$f_path' does not exist."

# کسب اطمینان از آن‌که برنامه‌های مورد نیاز ImageMagick نصب شده‌اند
for i in $REQUIRED_PROGS; do
  type "$i" &> /dev/null \
    || error_exit "required program '$i' not found."
done

# حلقه پردازش
for input_file in "$@"; do
  if [[ -r "$input_file" ]] && identify "$input_file" &> /dev/null; then
    filename="${input_file##*/}"
    if [[ -z "$f_path" ]]; then
      path="${input_file%/*}"
```

```
    else
      path="$f_path"
    fi
    [[ "$filename" == "$path" ]] && path="."
    base="${filename%.*}"
    if [[ -z "$f_ext" ]]; then
      ext="${filename##*.}"
    else
      ext="$f_ext"
    fi
    output_file="$path/$base-$size.$ext"
    convert "$input_file" -resize ${size}x${size} "$output_file" \
      || error_warning "Cannot convert '$input_file'. Skipping..."
  else
    error_warning "'$input_file' not a valid image file. Skipping..."
  fi
done

graceful_exit
```

صفحه اول
بخش منابع »

program_list

آیا هیچ از آن همه فایلهای داخل /usr/bin تعجب کرده‌اید؟ البته که تعجب کرده‌اید! بسیار خوب، شما می‌توانستید یک **whatis** روی هر فایل در دایرکتوری(که در یک سیستم معمولی تعداد آنها می‌تواند یکهزار یا بیشتر باشد) انجام بدهید، یا می‌توانید اسکریپت زیر را اجرا کنید.

program_list برنامه‌ای است که یک لیست تفسیری از برنامه‌های یک دایرکتوری(پیش‌فرض آن /usr/bin است) با نمایش شرح هر برنامه تولید می‌کند. همچنین، نام بسته‌ای را که برنامه به آن تعلق دارد نیز لیست می‌کند، یک سرنخِ مفید دیگر برای تعیین آنچه برنامه انجام می‌دهد.

برنامه خروجی‌اش را در سه قالب متفاوت به خروجی استاندارد ارسال می‌کند، متن ساده(پیش‌فرض)، کمیت‌های جداشده با tab (عالی برای پردازش بعدی لیست با سایر ابزارها یا بارگذاری آن داخل یک صفحه‌گسترده)، و قالب Markdown برای تبدیل مستقیم به HTML و سایر قالب‌ها.

program_list باید در هر سیستم بر مبنای rpm (از قبیل Red Hat، CentOS، Fedora، وغیره) یا بر پایه deb (مانند Debian، Ubuntu، و غیره) کار کند. به هر حال توجه داشته باشید که به سبب سرعت پایین جستجوی بسته روی اکثر سیستم‌ها، اجرای **program_list** می‌تواند زمان زیادی(تا نیم ساعت یا بیشتر) نسبت به سرعت سیستم شما و تعداد فایل‌های موجود، صرف کند.

علاوه بر دایرکتوری پیش‌فرض /usr/bin، برای دایرکتوری /bin، /sbin، و /usr/sbin نیز مناسب است.

مثال‌ها

```
me@linuxbox ~ $ program_list > program_list.txt
```

یک لیست متن ساده از /usr/bin تولید می‌کند و آن را در فایلی به نام `program_list.txt` ذخیره می‌کند.

```
me@linuxbox ~ $ program_list -t /usr/sbin > program_list_usr_sbin.tsv
```

یک لیست با کمیت جدا شده با tab از /usr/sbin تولید و آن را در فایلی به نام `program_list_usr_sbin.tsv` ذخیره می‌کند.

```
#!/bin/bash
# ------------------------------------------------------------------------
# program_list - تولید یک لیست تفسیری از برنامه‌ها

# Copyright 2014, William Shotts <bshotts@users.sourceforge.net>

# این برنامه نرم‌افزار آزاد است: شما می‌توانید آن را تحت شرایط نگارش #
#    شماره 3 مجوز GPL گنو یا به انتخاب خودتان هر نگارش پس از آن که #
# توسط بنیاد نرم‌افزارهای آزاد اعلام گردیده، توزیع و یا ویرایش کنید. #

# این برنامه به امید آنکه سودمند باشد توزیع می‌گردد، اما بدون هرگونه #
# تعهد حتی تعهد ضمنی کیفیت یا صلاحیت برای یک مقصود خاص. برای جزییات #
# بیشتر مجوز GPL گنو را در (http://www.gnu.org/licenses/) ببینید. #
```

```
# /usr/bin این برنامه یک لیست تفسیری از برنامه‌های دایرکتوری
#  یا دایرکتوری تعیین شده کاربر) شامل نام فایل، نام بسته‌ای)
#  man که بواسطه آن نصب شده است و شرح مختصر اخذ شده از صفحه
#  برنامه اگر در دسترس باشد، تولید می‌کند. قالب لیست می‌تواند
#  متن ساده(قالب پیش‌فرض)، کمیت‌های جدا شده با tab مفید برای)
#  .باشد Markdown وارد کردن لیست به سایر برنامه‌ها)، یا قالب

#  program_list [-h|--help]                  نحوه کاربرد :
#  program_list [[-m|--markdown]|[-t|--tabs]] [directory]

#  تاریخچه بازبینی:
# 2014-01-27      در برابر نام‌های ناخوشایند تقویت گردید(ver. 1.1)
# 2014-01-17  ایجاد گردید new_script اسکریپت 3.0.1 به وسیله نگارش
# ------------------------------------------------------------------------

PROGNAME=${0##*/}
VERSION="1.1"

clean_up() { # انجام خانه‌تکانی قبل از خروج
  return
}

error_exit() { # مدیریت خطاهای مهلک
  echo -e "${PROGNAME}: ${1:-"Unknown Error"}" >&2
  clean_up
  exit 1
}

graceful_exit() {
  clean_up
  exit
}

signal_exit() { # شده trap اداره سیگنال‌های
  case $1 in
    INT)
      error_exit "Program interrupted by user" ;;
    TERM)
      echo -e "\n$PROGNAME: Program terminated" >&2
      graceful_exit ;;
```

```
    *)
      error_exit "$PROGNAME: Terminating on unknown signal" ;;
  esac
}

usage() {
  echo -e "Usage: $PROGNAME [-h|--help]|[[-m|-t] [directory]]"
}

help_message() {
  cat <<- _EOF_
  $PROGNAME ver. $VERSION
  Produce an annotated listing of programs in a directory

  $(usage)

  Options:
  -h, --help      Display this help message and exit.
  -m, --markdown  Output Markdown formatted text (with pandoc
                  extensions).
  -t, --tabs      Output tab-separated values.

  directory is optional. Default is /usr/bin.

_EOF_
  return
}

set_mode() { # تنظیم قالب خروجی
  if [[ $mode == "empty" ]]; then
    mode=$1
  else
    error_exit "Only one mode (-m or -t) is allowed."
  fi
}

string() { # نوشتن یک رشته کاراکتر "char" که "width" مرتبه تکرار شده

  local -i width="$1"
  local char="$2"
```

```
  head -c "$width" < /dev/zero | tr '\0' "$char"
}

find_package() { # جستجو برای نام بسته بر مبنای توزیع

  local filename="$1" raw_package

  case $distro_style in
    debian)
      raw_package="$(dpkg-query -S "$filename" 2> /dev/null | tail -1)"
      echo "${raw_package%:*}"
      ;;
    redhat)
      rpm -qf "$filename" 2> /dev/null
      ;;
    *)
      error_exit "Unsupported distribution."
      ;;
  esac
}

# Trap سیگنال‌های
trap "signal_exit TERM" TERM HUP
trap "signal_exit INT"  INT

# تجزیه سطرفرمان
mode=empty
program_directory=/usr/bin
while [[ -n "$1" ]]; do
  case "$1" in
    -h | --help)
      help_message
      graceful_exit
      ;;
    -m | --markdown)
      set_mode markdown
      ;;
    -t | --tabs)
      set_mode tsv
```

```
      ;;
    -* | --*)
      usage
      error_exit "Unknown option $1"
      ;;
    *)
      program_directory="$1"
      break
      ;;
  esac
  shift
done

# منطق اصلی

declare -a filenames packages descriptions
declare -i index=1 max_fn_len=0 fn_len=0 col1_width col2_width
distro_style="unknown"

# تعیین نوع سیستم بسته‌بندی
[[ -x /usr/bin/apt-get ]] && distro_style="debian"
[[ -x /bin/rpm || -x /usr/bin/rpm ]] && distro_style="redhat"

# بررسی آن‌که program_directory معتبر است
[[ -d "$program_directory" ]] || \
  error_exit "$program_directory cannot be read"

# بار گیری آرایه با نام فایل‌ها، بسته‌ها، و توضیح‌ها
while IFS= read -r i; do
  filenames[index]="${i##*/}"
  fn_len=${#filenames[index]}
  # تعیین طولانی‌ترین نام‌فایل برای محاسبه عرض ستون
  [[ $fn_len -gt $max_fn_len ]] && max_fn_len=$fn_len

  packages[index]="$(find_package "$i")"

  # گرفتن توضیح برنامه، دور ریختن ابتدای آن و
  #          بزرگ کردن حرف ابتدای اولین کلمه.
  raw_description="$(whatis "${filenames[index]}" 2>/dev/null | head -1)"
  raw_description="${raw_description##*' - '}"
```

```
  descriptions[index]="${raw_description^*}"

  ((++index))
done < <(find "$program_directory" -mindepth 1 -maxdepth 1 -executable \
          -not -type d | sort -u)

#  Markdown درج سرآیند
if [[ $mode == "markdown" ]]; then
  markdown_header="Programs in $program_directory"
  echo -e "$markdown_header\n$(string ${#markdown_header} "=")\n\n"
  ((max_fn_len += 4)) # اجازه برای افزودن کاراکترهای اضافی به نام فایل‌ها
fi

# محاسبه عرض ستون‌ها
col1_width=$((max_fn_len + 1))
col2_width=$((80 - col1_width))

# Markdown درج جدول
if [[ $mode == "markdown" ]]; then
  echo "$(string $max_fn_len '-') $(string $col2_width '-')"
fi

for ((i=1; i<index; ++i)); do
  case $mode in
    empty)
      printf "%-${max_fn_len}s Package:%s\n" \
        "${filenames[i]}" \
        "${packages[i]}"
      # اندازه کردن توضیح برای ستون دوم
      echo "${descriptions[i]}" | \
        fold -s -w $col2_width | \
        pr -T -o $col1_width
      echo
      ;;
    tsv)
      printf "%s\t%s\t%s\n" \
        "${filenames[i]}" \
        "${packages[i]}" \
        "${descriptions[i]}"
      ;;
```

```
    markdown)
      printf "%-${max_fn_len}s Package:%s\n\n" \
        "**${filenames[i]}**" \
        "${packages[i]}"
      echo "${descriptions[i]}" | \
        fold -s -w $col2_width | \
        pr -T -o $col1_width
      echo
      ;;
  esac
done

# Markdown بستن جدول
[[ $mode == "markdown" ]] && echo -e "$(string 80 '-')\n"

graceful_exit
```

صفحه اول
بخش منابع »

ترجمه از یادداشت تاریخ Sunday, October 14, 2012 وبلاگ مولف در اینجا

new_script Version 3

محبوب‌ترین اسکریپت در ***LinuxCommand.org*** اسکریپت *new_script* بود، یک اسکریپت پوسته bash که اسکریپت‌های پوسته bash تولید می‌کرد. یعنی، یک تولید کننده الگوی اسکریپت. چند ماه قبل من *new_script* را برای نوسازی و استفاده بهتر از ویژگی‌های bash، شامل آرایه‌ها، بازنویسی کردم. طول نگارش جدید کمتراز نصف طول نگارش قبلی است و بازهم کار بیشتری انجام می‌دهد.

استفاده از *new_script* با تولید یک الگوی سفارشی شده شامل یک تجزیه‌کننده برای شناسه‌ها و گزینه‌های سطرفرمان اسکریپت شما، مقدار زیادی از کار شما را صرفه‌جویی خواهد نمود. الگو همچنین تعدادی روال کمکی سودمند از قبیل اداره‌کننده سیگنال و خطا را شامل می‌گردد. اسکریپت تولید شده کاملاً کاربردی است(اگرچه کاری انجام نمی‌دهد) و آماده وارد کردن منطق خاص برنامه شما می‌باشد.

Synopsis

```
new_script [-h|--help] [-q|--quiet] [-s|--root] [script]
```

توضیحات

new_script به طور پیش‌فرض، برای جزییاتی در باره اسکریپتی که باید تولید گردد از کاربر استعلام خواهد نمود:

```
Enter script output filename >
```

انتخاب یک نام برای فایل اسکریپت تولید شده.

```
Enter purpose of script >
```

وارد کردن یک سطر شرح اسکریپت.

```
Include GPL license header [y/n]? >
```

با وارد کردن «y» یک سرآیند اجازه‌نامه GPL داخل قطعه توضیح در ابتدای اسکریپت درج می‌گردد.

```
Does this script require superuser privileges [y/n]? >
```

با وارد کردن «y» یک روتین درج می‌گردد که قبل از اجازه دادن به اجرای اسکریپت تحقیق می‌کند که کاربر دارای مزایای کاربر ارشد باشد.

```
Does this script support command-line options [y/n]? >
```

اگر مایل به پشتیبانی گزینه‌های سطرفرمان برای اسکریپت خود هستید، پاسخ «y» بدهید.

```
Option 1:
```

```
Enter option letter [a-z] (Enter to end) >
```

اگر گزینه‌های سطرفرمان مورد درخواست باشند، یک گروه از اعلان‌ها ظاهر خواهند شد.نخست، شکل کوتاه گزینه(کاراکتر منفرد) از شما پرسیده می‌شود. برای تعیین گزینه یک حرف در محدوده a-z وارد کنید یا برای پایان دادن به حلقه ورود گزینه چیزی وارد نکنید(م: فقط اینتر بزنید).

```
Description of option ------------------->
```

یک شرح کوتاه برای گزینه وارد کنید. این توضیح در پیغام «help» استفاده خواهد شد.

```
Enter long alternate name (optional) ---->
```

در صورتیکه مطلوب شما است یک کلمه واحد برای نام گزینه بلند وارد کنید. برای مثال، در صورتیکه دارای یک گزینه «h» برای تابع help هستید، می‌توانید یک نام از قبیل «help» برای جایگزین بلند آن تعیین کنید. وقتی اسکریپت شما اجرا می‌شود، شما می‌توانید تابع help را با به‌کار بردن گزینه -h یا --help در سطرفرمان، فراخوانی کنید.

توجه نمایید که *new_script* همیشه گزینه‌های پیش‌فرض «h» و «help» را به طور خودکار ایجاد می‌کند.

```
Enter option argument (if any) ---------->
```

اگر گزینه شما یک شناسه نیاز دارد، در اینجا یک کلمه منفرد توصیف‌کننده برای آن وارد کنید. برای مثال، اگر شما در حال ایجاد اسکریپتی به نام *my_script* بودید که هم یک فایل ورودی و هم یک فایل خروجی را پشتیبانی می‌کرد، شما می‌باید برای هر کدام یک گزینه با شناسه‌های مربوط به هر یک داشته باشید، مانند:

```
my_script -i infile -o outfile
```

بعد از اینکه وارد نمودن اطلاعات گزینه را تمام کنید، *new_script* الگوی اسکریپت سفارشی شما را تولید خواهد کرد. اسکریپت شما شامل یک روال سودمند اداره کننده خطا به نام ***error_exit*** خواهد بود، که به این صورت استفاده می‌شود:

```
error_exit error_message
```

که در آن *error_message* یک رشته شامل یک پیغام خطای توصیفی است. تابع **error_exit** رشته *error_message* را در خروجی استاندارد خطا بیرون می‌دهد، تابع **clean_up** را برای اجرای هر تمیزکاری لازم فراخوانی می‌کند، سپس با یک وضعیت خروج 1 خارج می‌شود. این هم یک مثال:

```
error_exit "Something bad happened!"
```

مثال‌های دیگر را می‌توانید داخل کد خود اسکریپت *new_script* مشاهده کنید.

گزینه‌ها

برنامه *new_script* گزینه‌های خط فرمان زیر را پشتیبانی می‌کند:

- -h, --help

نمایش یک پیغام راهنمایی و خروج.

- -q, --quiet

وضعیت Quiet. برای کسب اطلاعات اسکریپت به کاربر اعلان نمی‌دهد واسکریپت پیش‌فرض را در خروجی استاندارد بیرون می‌دهد.

- -s, --root

برای اجرای اسکریپت تولید شده مزایای کاربر ارشد لازم است.

همچنین می‌توانید نام فایل مطلوب برای اسکریپت تولید شونده را در خط‌فرمان بعداز همه گزینه‌های تعیین شده به کار ببرید.

نصب

می‌توانید اسکریپت را در اینجا پیدا کنید. برای نصب آن، فقط متن را انتخاب کرده و به داخل یک ویرایشگر متن کپی و فایل حاصل را ذخیره نمایید. اگر احساس زبردستی می‌نمایید، می‌توانید به طور مستقیم از طریق خط فرمان آن را در یک فایل بنویسید. همانند قبل متن را انتخاب و کپی کنید، یک ترمینال باز کرده فرمان زیر(با فرض اینکه شما دارای یک دایرکتوری ***bin*** در دایرکتوری خانه خود هستید، در غیر اینصورت آنطور که لازم است تطبیق بدهید) را در آن تایپ کنید:

```
cat > ~/bin/new_script
```

با فشردن دکمه میانی ماوس اسکریپت را در ترمینالPaste کنید و Ctrl-d را تایپ نمایید. هنگامی که اسکریپت را نوشته‌اید، به این طریق مجوز اجرا به آن بدهید:

```
chmod +x ~/bin/new_script
```

« بخش منابع | صفحه اول | «new_script