www.takbook.com

www.takbook.com

راهنمای اسکریپت نویسی لینوکس

بهمن عرب‌رضائی

ویرایش ۲.۰
بهار ۱۴۰۲

www.takbook.com

راهنمای اسکریپت نویسی لینوکس

۱ معرفی ۱

۲ مقدمه ۲

۳ معرفی ساختار لینوکس ۵

۱.۳ دایرکتوری / ۷

۳.۲ دایرکتوری /bin/ ۷

۳.۳ دایرکتوری /boot/ ۷

۳.۴ دایرکتوری /dev/ ۷

۳.۵ دایرکتوری /etc/ ۷

۳.۶ دایرکتوری /home/ ۷

۳.۷ دایرکتوری /sbin/ ۷

۳.۸ دایرکتوری /lib/ ۷

۳.۹ دایرکتوری /media/ ۸

۳.۱۰ دایرکتوری /mnt/ ۸

۳.۱۱ دایرکتوری /opt/ ۸

۳.۱۲ دایرکتوری /tmp/ ۸

۳.۱۳ دایرکتوری /proc/ ۸

۳.۱۴ دایرکتوری /usr/ ۹

۳.۱۵ دایرکتوری /srv/ ۹

۴ شروع اسکریپت نویسی ۱۰

۱.۴ متغیرهای محیطی ۱۲

۵ کاراکترهای ویژه ۱۵

۵.۱ کاراکتر # ۱۵

۵/۲ کاراکتر ; ۱۵

۵/۳ کاراکتر ;; ۱۵

www.takbook.com

۴.۵ کاراکتر “ ۱۶

۵/۵ کاراکتر ‘ ۱۶

۶.۵ کاراکتر ` ۱۶

۷.۵ کاراکتر \ ۱۷

۵/۸ کاراکتر / ۱۷

۵/۹ کاراکتر: ۱۷

۱۰.۵ کاراکتر $ ۱۸

۱۱.۵ کاراکتر >, < , << , >> ۱۸

۶ متغیرها ۱۹

۶.۱ معرفی ۱۹

۶.۲ محدوده متغیرها ۲۱

۷ دستورات شرطی ۲۳

۱.۷ عبارت شرطی ۲۴

۸ حلقه ها ۲۶

۱.۸ حلقه for ۲۶

۸.۲ سری مقادیر ۲۷

۸.۳ خروج و ادامه حلقه ۲۹

۴.۸ حلقه while ۳۱

۹ توابع ۳۴

۱.۹ تعریف ۳۴

۹.۲ بازنویسی دستورات ۳۷

۹.۳ ارسال پارامتر غیر مستقیم Indirect ۳۹

۴.۹ توابع بازگشتی Recursive ۴۰

۱۰ آرایه ها ۴۳

۱۰.۱ تعریف ارایه ۴۴

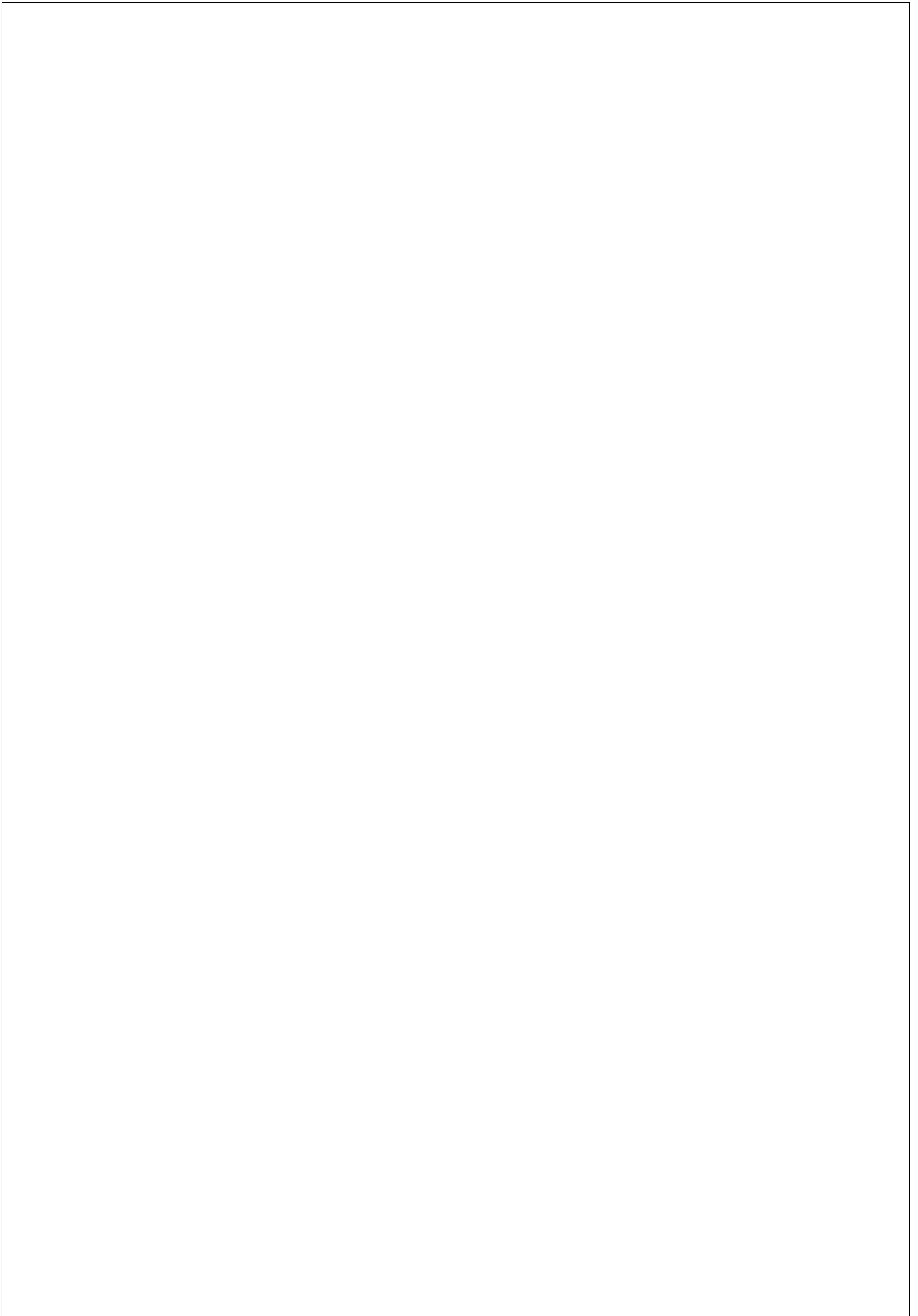
www.takbook.com

۱۰.۱.۱ با ایندکس ۴۴
۱۰.۱.۲ تعریف تمام اعضا ۴۴
۱۰.۱.۳ ترکیبی ۴۵
۱۰.۲ عملگرهای آرایه ۴۵
۱۰.۲.۱ دسترسی به اعضای آرایه ۴۵
۱۰.۲.۲ طول آرایه ۴۷
۳.۲.۱۰ زیر مجموعه ۴۷
۱۰.۲.۴ جایگزینی ۴۸
۱۰.۲.۵ حذف ایتم ۴۹
۱۱ دستورات تست ۵۱
۱۱.۱ درست چیست؟ ۵۱
۱۱.۲ ساختار (( )) ۵۳
۳.۱۱ ساختار if-elif-fi ۵۶
۱۱.۴ دستور case-esac ۵۷
۵.۱۱ دستور test ۵۹
۶.۱۱ ساختار [[ ]] ۶۱
۱۲ عبارات منظم Regular Expression ۶۳
۱.۱۲ قوانین عبارات منظم ۶۳
۱۳ رشتهها و متغیرها ۶۷
۱۳.۱ رشته ها ۶۷
۱۳.۱.۱ طول رشته ۶۷
۱۳.۱.۲ طول زیر رشته ۶۸
۱۳.۱.۳ ایندکس زیر رشته ۶۸
۱۳.۱.۴ استخراج زیر رشته ۶۸
۵.۱.۱۳ حذف زیر رشته ۶۹

www.takbook.com

۱۳.۱.۶ جایگذاری زیررشته ۷۰
۱۳.۲ متغیرها ۷۱
۱۳.۲.۱ فرمت کامل ۷۱
۱۳.۲.۲ مقدار پیش فرض ۷۲
۱۳.۲.۳ مقدار جایگزین ۷۳
۱۳.۲.۴ پیام خطا ۷۳
۱۳.۲.۵ فراخوانی غیر مستقیم ۷۴

www.takbook.com



www.takbook.com

۱ معرفی

هدف این کتاب معرفی دنیای جذاب اسکریپت نویسی لینوکس و همچنین سیستم عامل لینوکس می باشد.
این کتاب بدون پشتیبان مالی در حال تدوین می باشد لذا در صورتی که این کتاب برای شما مفید بوده و مایل بودید مبلغ ۲۰هزار تومان، یا هر مبلغ دیگر، را به کارت زیر نزد بانک تجارت واریز نمایید

شماره کارت :

5859-8310-1442-1274

شبا:

IR840180000000002624053055

به نام : بهمن عرب‌رضائی

www.takbook.com

۲ مقدمه

امروزه لینوکس یکی از محبوب ترین سیستم عامل‌های موجود می باشد. این سیستم عامل در ابزارهای متعددی همانند تلفن های همراه، ابزارهای صنعتی، کامپیوترهای خانگی و ... استفاده می شود.

لینوکس نسل گرافیکی سیتم عامل یونیکس می باشد همانند ویندوز که در واقع نسل گرافیکی سیستم عامل DOS بود. یونیکس اولین بار توسط دنیس ریچی Dennis Ritchie، کن تامسون Ken Thompson، داگلاس مکیلروی Douglas McIlroy و جو اوسانا Joe Ossanna توسعه داده شد. این سیستم عامل سیستم عامل مبتنی بر خط فرمان بود به این معنا که کنترل سیستم عامل و اجرای برنامه از طریق دستورات که در خط فرمان صادر می شد، صورت می گرفت، همانند سیستم عامل داس. اما قابلیت بسیار مهمی که در این سیستم عامل وجود داشت فعال بودن سیستم عامل در زمان اجرای برنامه ها بود. به عبارت دیگر سیستم عامل داس سیستم عامل تک فرآیند Single Process بود یعنی در زمان اجرای یک فرایند سیستم عامل هیچ گونه دخالتی در کنترل منابع همانند حافظه، پردازشگر و ... نداشت و فرآیند در حال اجرا می توانست تمامی منابع را مصرف کرده و حتی باعث خرابی Crash کل سیستم گردد. اما در سیستم عامل یونیکس، سیستم عامل همواره فعال بوده و اجرای فرآیند را کنترل می نمود و در نتیجه همزمان امکان اجرای چندین فرایند وجود داشت لذا این سیستم عامل یک سیستم عامل چند فرایندی Multi Process بود.

در سیستم عامل یونیکس می توانستید اجرای یک فرایند را به پشت صحنه Back Ground ارسال کنید و مجدد خط فرمان را در اختیار بگیرید و یا حتی چندین خط فرمان در اختیار داشته باشید. در واقع خود خط فرمان نیز یک فرایند بود که سیستم عامل اجرا می نمود پس شما می توانستید با استفاده از دکمه‌ها Alt+F1 الی Alt+F12 خط فرمان های مختلف را فراخوانی کنید و در هر کدام دستورات مختلفی را صادر کنید.

این قابلیت باعث می شد سیستم عامل یونیکس سیستم عامل بسیار قدرتمندی باشد اما امکان اجرای آن، در آن زمان، بر روی سیستم های شخصی PC (Personal Computer) وجود نداشت.

در کنار مزیت چند فرایندی، خط فرمان یونیکس از مفسر interpreter بسیار قوی برخوردار بود که می توانستید مجموعه ای از دستورات خط فرمان را در یک فایل نوشته و به صورت یک فرمان در خط فرمان اجرا نمود. به این قابلیت در یونیکس Bash Script Programming گفته می‌شد شبیه قابلیت Batch File در سیستم عامل داس. تفاوت بسیار مهم آن با Batch File های در داس این بود که در داس شما معمولا فقط دنباله ای از دستورات را می نوشتید و برای کنترل روند اجرا همانند دستورات شرطی، حلقه های، فراخوانی توابع و ... امکاناتی در اختیار نداشتید، البته در نسخه های اولیه داس، و در صورت نیاز باید با هنر

www.takbook.com

خود و استفاده از دستورات غیر متعارفی مانند برچسب گذاری label این امکانات را شبیه سازی می‌کردید که خود باعث ناخوانایی شدید در برنامه نوشته شده می‌گریدد.

اما در آن سمت در یونیکس به همان شکل زبان‌های برنامه نویسی امکان استفاده از دستورات کنترل روند برنامه همانند شرط‌ها، حلقه ها و فراخوانی توابع را در خود داشت و علاوه بر آن به راحتی می توانستید خروجی دستورات را به عنوان ورودی به دستور بعدی ارسال کنید که به آن لوله‌گذاری piping گفته می شد.

این قابلیت‌ها باعث شد که مدیران سیستم ها به شدت به Bash Script نویسی علاقمند شوند و بسیاری از روال های مدیریتی خود را در قالب این فایل‌ها برنامه نویسی کرده و در کنار آن استفاده از سرویس برنامه‌ریزی فرایندها Crontab قدرت خارق العاده ای به مدیران سیستم اعطا نمود

این قابلیت در ادامه در سیستم عامل لینوکس حفظ شد و امروزه مهارت در Bash Scriptنویسی یکی از الزامات هر مدیر سیستم و حتی توسعه گرها و برنامه نویس‌های سیستم عامل لینوکس گردیده است.

امروزه برای لینوکس خط فرمان‌ها یا شل های متعددی وجود دارد اولین خط فرمان را آقای کن تامسون با نام Thompson Shell معرفی کرد. در سال ۱۹۷۷ خط فرمان Bourne shell را آقای Stephen Bourne معرفی نمود که به شدت مورد استقبال قرار گرفت و امروزه بیشترین کاربرد را دارد.

معروف ترین خط فرمان ها عبارتند از :

- Bourne Shell
- Bourne Again Shell [Bash]
- C Shell
- Korn Shell
- TC Shell

اما شاید بتوان گفت آنچه باعث فدرتمندتر شدن خط فرمان Bash شده است موارد ذیل می باشد:

۱. قدرت ویرایش فرمان ها
۲. نامحدود بودن تعداد رخدادهای قابل ذخیره
۳. امکان معرفی نام مستعار برای دستورات
۴. نامحدود بودن طول آرایه‌ها

www.takbook.com

برای یادگیری و تست برنامه‌های مورد استفاده می توانید از سایت زیر نیز کمک بگیرید

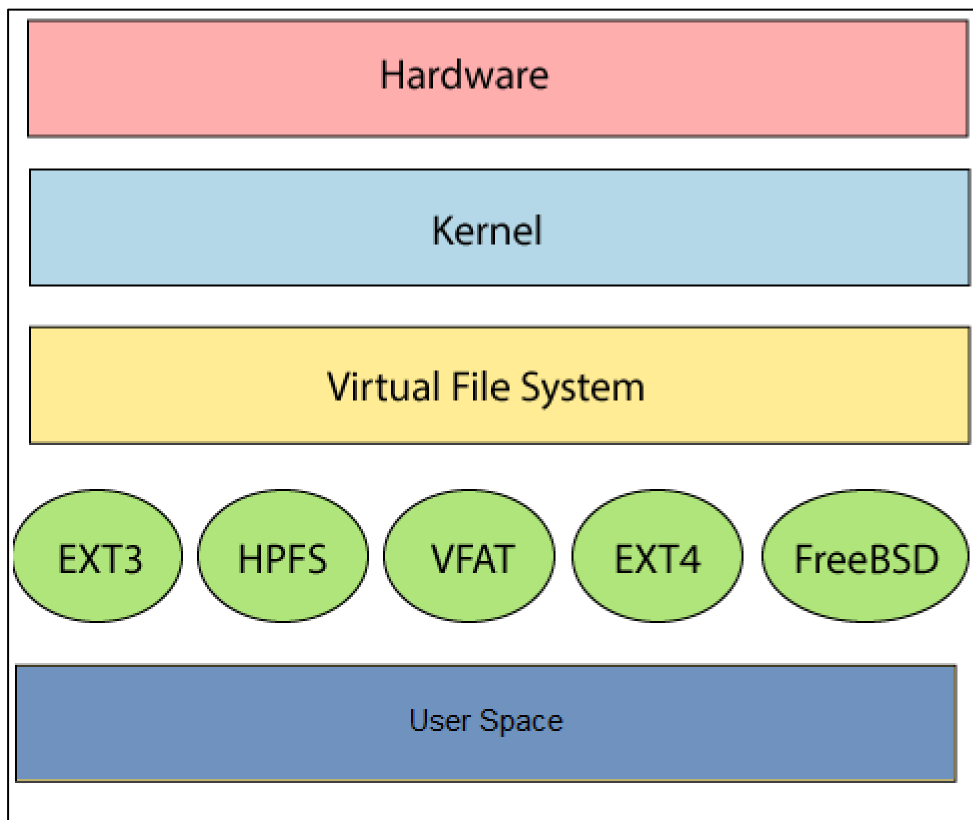
۱. https://www.jdoodle.com/test-bash-shell-script-online
۲. https://rextester.com/l/bash_online_compiler

www.takbook.com

۳معرفی ساختار لینوکس

قبل از اینکه شروع به یادگیری و معرفی شل اسکریپت کنیم می بایست اندکی با ساختار فایل ها در لینوکس آشنا شویم. برخلاف ویندوز در لینوکس پارتیشن بندی Partitioning به شکل محسوس برای کاربر وجود ندارد و عملا پارتیشن ها در پشت صحنه می باشند. آنچه که کاربر با آن مواجه می شود ساختار دایرکتوری می باشد. اما چه رابطه ای بین پارتیشن و دایرکتوری وجود دارد؟

مهمترین موضوع در لینوکس این است که همه چیز در لینوکس فایل می باشد. بنا براین در لینوکس دایرکتوری هم فایل است و پارتیشن هم فایل است. در واقع در سطح فایل سیستم File System برای هر فایلی یک مشخصه به نام inode وجود دارد. بنابراین در سطح فایل سیستم تمامی المان هایی که ذخیره می شوند با inode شناخته شده و به عنوان فایل تفسیر می شود و در زمان نمایش به کاربر است که می تواند به صورت دایرکتوری، پارتیشن، سخت افزار و ... تفسیر و نمایش داده شود.

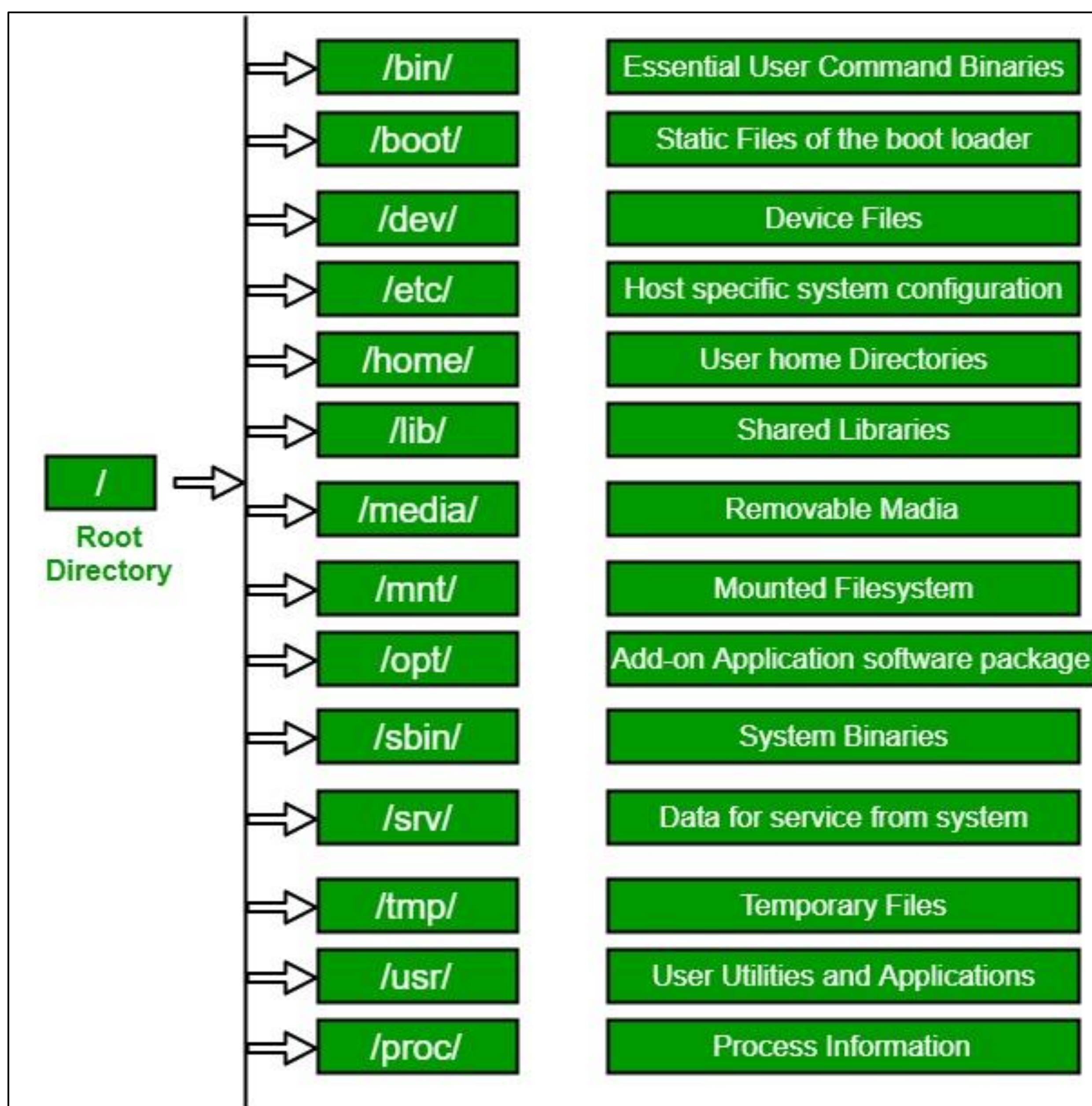


شکل ۱ - ساختار فایل سیستم

www.takbook.com

همانطور که در شکل ۱ دیده می شود فایل سیستم های مختلف همه بر روی لایه ای به نام VFS قرار دارند و این لایه است که با کرنل در ارتباط است. همانطور که گفته شد در کرنل همه اطلاعات به صورت فایل ذخیره می شود در نتیجه فایل سیستم های مخلف همانند Ext4, Ext4 و ... مهمترین کاری که انجام می دهند نحوه نمایش اطلاعات به کاربر می باشد

در سطح کاربر ساختار دایرکتوری مشابه شکل ۲ در اغلب نسخه های لینوکس قابل رویت است



شکل ۲- ساختار دایرکتوری در لینوکس

در ادامه به صورت مختصر هر کدام از این دایرکتوری ها معرفی می شوند

www.takbook.com

۳/۱ دایرکتوری /

این دایرکتوری، ریشه و جد همه دایرکتورها است و بالاتر از آن دایرکتوری دیگری وجود ندارد در مقایسه با ویندوز همانند My Computer می باشد

۳/۲ دایرکتوری /bin/

در این دایرکتوری فایل اجرایی پایه که مورد نیاز کاربر و عملیات سیستم می باشد، قرار دارد. از جمله این فایل‌های اجرایی فایل bash می باشد که یکی از خطوط فرمان های معروف لینوکس است. همچنین دستوراتی مانند cp, mv, date و... همه در این مسیر تعریف شده اند

۳/۳ دایرکتوری /boot/

این دایرکتوری حاوی فایل‌های بوت boot می باشد. مهم ترین فایلی که در این دایرکتوری قرار دارد فایلی با نامی مشابه ...-vmlinuz-XX.XX.XX می‌باشد که عملا کرنل لینوکس می‌باشد همچنین فایل‌های مربوط به boot loader نیز در همین دایرکتوری، به عنوان مثال دایرکتوری به نام grub، وجود دارد. امروزه یکی از معروف ترین بوت لودرها grub می باشد

۳/۴ دایرکتوری /dev/

تمامی سخت افزار های موجود به صورت یک فایل در این دایرکتوری تعریف شده اند. سخت افزارهایی همانند هارد، رم، کارت شبکه، کیبورد و ... این دقیقا همان چیزی است که قبلا نیز ذکر شد که در لینوکس همه چیز فایل است

۳/۵ دایرکتوری /etc/

تمامی برنامه ها فایل های تنظیمات خود را در این دایرکتوری ذخیره می کنند. به صورت استاندارد هر برنامه در این مسیر دایرکتوری به نام خود ایجاد می‌کند و فایل های تنظیمات را در آن ذخیره می نماید

۳/۶ دایرکتوری /home/

برای هر کاربری که در سیستم ایجاد می شود دایرکتوری با همان نام در این مسیر ایجاد می گردد و دسترسی و صاحب آن همان کاربر خواهد بود. در زمان نصب لینوکس پیشنهاد می گردد این دایرکتوری بر روی پارتیشن مجزا نصب گردد

۳/۷ دایرکتوری /sbin/

این دایرکتوری همانند دایرکتوری bin می باشد با این تفاوت که دسترسی به برنامه های این دایرکتوری صرفا در اختیار کاربرهای مدیر می باشد

۳/۸ دایرکتوری /lib/

www.takbook.com

اغلب برنامه‌های اجرایی فایل‌های برای اجرا نیاز دارند که با نام کتابخانه library شنا خته می شوند. این فایل ها در لینوکس با پسوند so. و a. ذخیره می شوند. به صورت استاندارد برنامه‌ها در زمان اجرا در مسیرهای از پیش تعریف شده به دنبال کتابخانه‌ها می گردند که دایرکتوری lib/ یکی از این مسیرهای از پیش تعریف شده می باشد. در ادامه نحوه تعریف مسیرهای اختصاصی نیز تشریح خواهد شد

۳/۹ دایرکتوری /media/

در برخی مواقع کاربر دستگاه جانبی مانند flash memory, DVD Drive و ... را به سیتم متصل می کند در این زمان سیستم عامل دایرکتوری با نام مناسب برای این دستگاه در این مسیر ایجاد می نماید و به کاربر امکان دسترسی به محتویات دستگاه را می دهد

۳/۱۰ دایرکتوری /mnt/

نام این دایرکتوری مخف کلمه mount (نگاشت) می باشد. نگاشت فرایندی در لینوکس می باشد که به کمک آن دسترسی به فضای فایل ها را بر روی دایرکتوری ایجاد می نماید. مثلا جهت دسترسی به فایل هایی که در یک پارتیشن ذخیره می شود باید پارتیشن به دایرکتوری در سیستم نگاشت شود؛ به صورت استاندارد این دایرکتوری ها در این مسیر ایجاد می گردند

۳/۱۱ دایرکتوری /opt/

دایرکتوری که برخی نرم افزارها به عنوان مسیر نصب از آن نصب می کنند شبیه فولدر program files در ویندوز

۳/۱۲ دایرکتوری /tmp/

همانطور که از نام آن می توان حدس زد دایرکتوری موقت است. به عبارتی تمامی فایل هایی که در این دایرکتوری ذخیره می شوند موقت است و معمولا با راه اندازی مجدد سیستم این فایل ها از بین می روند

۳/۱۳ دایرکتوری /proc/

اطلاعات آماری فرایندها و همچنین اطلاعات آماری سیستم عامل در این دایرکتوری ذخیره می شود. اطلاعاتی مانند حجم ترافیک مصرف شده بر روی کارت های شبکه، حجم حافظه موجود و میزان حافظه مصرف شده و .... فایل های این دایرکتوری اغلب به صورت تنها خواندنی می باشند و قابل تغییر نمی باشد و

www.takbook.com

اگر هم به هر صورت تغییری در آن ها ایجاد شود این تغییرات ناپایدار بوده و با راه اندازی مجدد سیستم از بین می رود. معمولا این دایرکتوری بر روی ram سیستم ایجاد می شود و در نتیجه فایل ها بعد از راه اندازی مجدد از بین رفته و مجدد ساخته می شوند

۳/۱۴ دایرکتوری /usr/

ابزارهایی که کاربران نصب می کنند معمولا در این دایرکتوری ذخیره می شود. البته به صورت عادی کاربران دسترسی به این دایرکتوری ندارند و مدیر است که اجازه دسترسی به این دایرکتوری را دارد اما مدیران، نرم افزارها و ابزارهایی که برای کاربران نصب می شود را در این مسیر قرار می دهند

۳/۱۵ دایرکتوری /srv/

سرویس‌هایی که به سایر دستگاه ها در شبکه ارائه خدمت می کنند در این دایرکتوری قرار می گیرند که معروف ترین آن، سرویس های وب Web Server می باشد مانند سرور Apache

www.takbook.com

۴شروع اسکریپت نویسی

دنیای برنامه نویسی، دنیای مهیجی است که یادگیری آن با صرف زمان و تمرین فراوان به دست می آید. در این بین برخی زبان ها ظرایف بسیار زیادی در برنامه نویسی دارند که یادگیری آن ها مستلزم صرف زمان بیشتر، تمرین و کسب تجربه بیشتر می‌باشد، زبان هایی مانند C&C++. برخی زبان‌های دیگر با صرف زمان کمتر و سرعت بیشتر قابلیت یادگیری دارند همانند اسکریپت نویسی Bash Shell.

زبان های اسکریپتی غالبا از نوع زبان های دوم هستند و لذا یادگیری زبان های اسکریپتی بسیار سریع و لذت بخش می باشد.

براساس روال کتاب های آموزشی معرفی زبان را با برنامه سلام شروع می کنیم

```
hello.sh
#!/bin/sh
#This is first shell script
echo "Hello World!"
```

کد بالا را در فایلیT مثلا hello.shT ذخیره کنید. دقت کنید که حتما پسوند فایل شما sh باشد. البته سیستم عامل لینوکس به پسوند فایل حساسیت ندارد اما پسوندها حداقل دو کارکرد مهم دارند

۱. معرفی محتوی فایل
۲. کمک به سیسم عامل در شناسایی برنامه مناسب جهت خواندن یا اجرای فایل

بنابراین عادت به انتساب پسوند به فایل ها عادت بسیار خوبی می باشد.

پس از ذخیره فایل برای اجرای آن به دو شکل می توان اقدام کرد.

۱. انتساب خاصیت اجرایی به فایل و اجرای ان

```
#> chmod +x ./hello.sh
#>./hello.sh
```

۲. اجرا با استفاده از خود برنامه bash

```
#>sh ./hello.sh
```

www.takbook.com

خط اول با دستور #!/bin/sh نوشته شده است. در این دستور دو نکته وجود دارد

۱. شروع خط با نویسه # : در ادامه اشاره خواهد شد که نویسه # از نویسه‌ها ویژه است و معنای خاص دارد. این نویسه در ابتدای خط به معنای توضیحات comment می باشد. خطی که با این نویسه شروع شود به عنوان توضیحات است و اجرا نمی‌گردد در نتیجه هر آنچه نوشته شود هیچ خطایی در اجرا ایجاد نمی کند.
اما توضیحات این خط نیر توضیحات خاصی است.

۲. بلافاصله پس از نویسه # نویسه ! درج شده است. در این حالت خاص نویسه ! به عنوان یک ابر متن Hyper Text شناخته می شود. ابر متن، نوعی نشانه گذاری متن است که برنامه های مختلف قادر به شناسایی و تفسیر آن می باشند. در اینجا نویسه ! مشخص می کند که مسیر پس از آن، مسیر فایل مفسر می باشد و ویرایشگر با استفاده از آن بدون در نظر گرفتن پسوند فایل، قادر به نمایش متن به صورت رنگی می باشد. به عبارت دیگر با حذف پسوند خواهید دید که همچنان ویرایشگر محتویات فایل را به درستی تشخیص داده و رنگی نمایش می دهد

خط بعدی فرمان نمایش در خروجی می باشد. فرمان echo به صورت سنتی در تمامی خطوط فرمان‌ها در سیستم عامل‌های مختلف و حتی در زبان های اسکریپتی به معنای ارسال مقدار ورودی فرمان به خط فرمان یا همان خروجی استاندارد می‌باشد. این دستور بسیار پرکاربرد است

به این ترتیب اولین اسکریپت نوشته و اجرا شد. اما اسکریپت ها نیز مانند هر برنامه دیگری در لینوکس خروجی عددی دارند. در خط فرمان هر دستوری که اجرا شود علاوه بر خروجی هایی که در خط فرمان نمایش می دهد خروجی نیز به عنوان نتیجه نهایی در متغیر محیطی به نام $? ذخیره می کند. این خروجی به صورت سنتی اگر مقدار 0 باشد به معنای اجرای موفقیت آمیز بوده و در غیر این صورت شماره خطا خواهد بود.

در اسکریپت‌ها چنانچه خروجی مشخص نشود همیشه مقدار 0 برگدانده می شود

```
#> ./hello.sh
#>echo $?
0
#>
```

www.takbook.com

چنانچه بخواهید مقدار خروجی متفاوتی به خط فرمان اعلام شود می توانید از دستور exit <value> استفاده کنید

```
hello.sh
#!/bin/sh
# this second test script
echo "Hello Again!"
exit 10

#>./hello.sh
#>echo $?
10
```

۴/۱ متغیرهای محیطی

متغیرهای محیطی، متغیرهایی در سطح سیستم عامل یا خط فرمان می‌باشند که سایر برنامه‌ها به آنها دسترسی دارند. معمولا این متغیرها با اهداف زیر استفاده می گردند

۱. ارسال ورودی به برنامه های دیگر
۲. اعلان تنظیمات پیش فرض
۳. نگهداری اطلاعات سیستمی، جهت استفاده سایر برنامه‌ها مانند نسخه سیستم عامل

فراخوانی متغیرهای محیطی در زمان اجرا با متغیرهای محلی تفاوتی ندارد یعنی با همان نویسه $ می توان متغیرهای محیطی را فراخوانی کرد مثلا echo $HOME که مسیر دایرکتوری اختصاصی کاربر فعلی در آن ذخیره شده است

جهت مشاهده لیست متغیرهای محیطی در سطح فرمان می توان از دستور env استفاه نمود

www.takbook.com

```
#> env
SHELL=/bin/bash
LANGUAGE=en_US:
LC_ADDRESS=az_IR
LC_NAME=az_IR
LC_MONETARY=az_IR
PWD=/home/bahman
LOGNAME=bahman
XDG_SESSION_TYPE=tty
LOCAL_SIPA=127.0.0.1:6159
HOME=/home/bahman
LC_PAPER=az_IR
LANG=en_US.UTF-8
LESSCLOSE=/usr/bin/lesspipe %s %s
XDG_SESSION_CLASS=user
LC_IDENTIFICATION=az_IR
TERM=xterm
LESSOPEN=| /usr/bin/lesspipe %s
USER=bahman
LOCAL_SERVER=127.0.0.1:6158
SHLVL=1
LC_TELEPHONE=az_IR
LC_MEASUREMENT=az_IR
BLUGW_CONFIG_PATH=/home/bahman/blugw/
XDG_SESSION_ID=144
XDG_RUNTIME_DIR=/run/user/1000
SSH_CLIENT=192.168.30.1 51352 22
```

این لیست در سیستم عامل‌های مختلف متفاوت خواهد بود اما برخی از متغیرها مشترک است

جهت تعریف یک متغیر محیطی جدید حتما باید از دستور export استفاده نمود

www.takbook.com

```
#> export  USER_NAME="bahman"

#>env

…

USER_NAME=bahman

…
```

www.takbook.com

۵کاراکترهای ویژه

در تمامی زبان‌های برنامه نویسی برخی از کاراکترها معنایی خاصی دارند و در زمان استفاده از آن ها باید دقت کرد. دقیقا مانند کلمات کلید زبان. همانطور که برخی کلمات در زبان های برنامه نویسی کلیدی هستد و به عنوان دستور تفسیر می‌شوند و برنامه نویس مجاز به استفاده از آنها به عنوان نام متغیر و تابع نمی باشد برخی کاراکترها نیز معنای خاص دارند و در هنگام استفاده از آن ها باید دقت کرد که این کاراکتر در آن جا چه معنی دارد. در ادامه به بررسی این کاراکترهای می پردازیم

۵/۱ کاراکتر

کاراکتر توضیحات comment. اگر در ابتدای خط باشد کل این خط توضیحات محسوب شده و اجرایی نخواهد بود

```
# This is comment line
echo "This is not comment #"
```

دقت شود که این کاراکتر در ابتدای خط به معنای توضیحات است اما در یک رشته دیگر معنای خاصی ندارد همچنین در انتهای دستور نیز با رعایت یک فاصله نیز باعث می شود ادامه خط به عنوان توضیحات در نظر گرفته شود

```
echo "This is not comment" # but this is comment
```

۵/۲ کاراکتر ;

در اسکریپت نویسی هر خط یک فرمان می باشد و انتهای خط به معنای پایان فرمان می باشد. بنابراین انتهای خط به عنوان جداکننده دستورات تلقی میگردد اما در برخی مواقع نیاز می شود در یک خط بیش از یک فرمان نوشت در این هنگام می توان از کاراکتر ; استفاده کرد. این کاراکتر نقش انتهای دستور و شروع دستور جدید را ایفا می کند

۵/۳ کاراکتر ;;

در دستور case به معنای پایان حالت تعریف شده و شروع حالت بعدی می باشد

www.takbook.com

```
case "$variable" in
abc) echo "\$variable = abc" ;;
xyz) echo "\$variable = xyz" ;;
esac
```

۵/۴ کاراکتر "

تعیین کننده ابتدا و انتهای رشته می باشد. تمامی کارکترهای قرار گرفته بین دو علامت " تشکیل رشته می‌دهند. دقت شود که در اسکریپت نویسی به دو صورت می توان رشته را تعریف کرد در این روش امکان استفاده از کاراکترهای ویژه در محتویات رشته می‌باشد. در صورت استفاده از کاراکتر ویژه در این نوع رشته-ها، این کاراکترها به عنوان دستور تلقی گردیده و بخشی از رشته نخواهد بود

```
echo "This is string to out put"
var="This is variable"
echo "This is string to out put $var" # This is string to output This is variable
```

۵/۵ کاراکتر '

دقیقا کاربردی همانند " دارد با این تفاوت که در رشته‌هایی که با این کاراکتر تعریف می شوند کاراکترهای ویژه دیگر به عنوان دستور تلقی نمی‌گردند.به توضیحات خط آخر توجه شود

```
echo "This is string to out put"
var="This is variable"
echo 'This is string to out put $var' # output: This is string to out put $var
```

۵/۶ کاراکتر `

کاراکتر backquotes به صورت دوتایی استفاده می شود و رشته ای که در بین آن قرار می گیرد به عنوان یک دستور اجرایی تلقی می گردد که خروجی دستور به جای ارسال به خروجی استاندارد در متغیر منتسب شده ذخیره می گردد

```
var=`echo "This is command"` # output: no out put
echo $var # output: This is command
```

www.takbook.com

۵/۷ کاراکتر \

کاراکتر فرار در تمامی زبان‌ها وجود دارد. در زمانی که می خواهید در یک رشته کاراکتر ویژه به عنوان فرمان تلقی نگردد قبل از آن از این کاراکتر استفاده می‌شود. در نتیجه دیگر مفسر، کاراکتر بعد از آن را تفسیر فرمان نکرده و مستقیم به خروجی ارسال می کند

```
var=”This is variable”
echo “This is string to out put \$var” # output: This is string to out put $var
```

۵/۸ کاراکتر /

این کاراکتر در رشته هایی که مسیر فایل یا دایرکتوری را مشخص می کنند به عنوان جدا کننده مسیر به کار می رود.

۵/۹ کاراکتر:

معادل دستور NOP در زبان های برنامه نویسی می باشد. از دیدگاه برنامه نویسی همیشه مقدار بازگشتی آن true می باشد.

```
:
echo $? # output: 0
```

همچنین جهت ایجاد حلقه های بی نهایت

```
while :
do
operation-1
operation-2
…
done
```

در دستورات شرطی، صرفنظر کردن از اجرای دستورات یک بلاک، مثلا if

www.takbook.com

```
if condition
then :
else
take some action
fi
```

۵/۱۰ کاراکتر $

متغیرها در اسکریپت دو حالت دارند یا در حال مقدار دهی هستند یا در حال استفاده در فرمان. در حالت اول متغیر بدون پیشوند و پسوند استفاده می شود اما در حالت دوم باید با پیشوند $ استفاده شود. بنابراین علامت $ یعنی رشته بعد متغیری است که مقدار آن برگردانده خواهد شد

```
var="bahman" # assignment
sum="Hello " + $var # read var
echo $sum # Hello bahman
```

۵/۱۱ کاراکتر >> , << , < ,>

کاراکترهای تغییر مسیر خروجی. این کاراکترها باعث می شوند خروجی دستورات به جای چاپ شدن در خروجی استاندارد به جایی دیگر ارسال شوند. تفاوت > و < با دو کاراکتر دیگر >> و << در این است که دو کارکتر >> , << به معنای اضافه شدن خروجی به محتوی مقصد فعلی و دو کاراکتر < , > به معنای جایگزینی محتوی مقصد با خروجی جاری می باشد

```
echi "Hello" > /tmp/test # create test file and write Hello to it as content of file
echo "Hello" >> /tmp/test # add Hello to end of file
```

www.takbook.com

۶متغیرها

۶/۱ معرفی

متغیرها یکی از اجزای اساسی در برنامه نویسی و اسکریپت نویسی می باشند. در نیای برنامه نویسی، برنامه نویس همواره در حال ذخیره و بازیابی اطلاعات پایه می‌باشد. به عنوان مثال زمانی که برنامه‌ای قرار است دو عدد را با هم جمع کند ابتدا این دو عدد را باید از کاربر دریافت کرده و سپس با هم جمع نماید، دریافت کردن اعداد از کاربر یعنی ذخیره در حافظه برای محاسبات آتی، همچنین پس از محاسبه حاصل جمع باید نتیجه جهت استفاده های آتی در جایی ذخیره گردد. ذخیره در حافظه به کمک متغیرها صورت می‌گیرد و به وسیله متغیر برنامه نویس اطلاعات را در حافظه ذخیره و بازیابی می‌نماید. مثلا در مثال ذکر شده جهت ذخیره ورودی‌ها می توان نام متغیرهای را input1 و input2 در نظر گرفت و جواب جمع را در متغیر sum ذخیره نمود.

در نامگذاری متغیرها باید از اسامی معنادار استفاده نمود به صورتی که در زمانی خواندن کد به راحتی بتوان هدف از تعریف متغیر را درک کرد همچنین هر زبان برنامه نویسی محدودیت‌های خاص خود را در نامگذاری متغیرها دارد که باید رعایت نمود . در اسکریپت نویسی نام متغیرها فقط شامل اعدا، حروف و نویسه _ می باشد سایر نویسه‌ها در نام متغیر مجاز نبود و باعث خطا می گردد.

در زمان تعریف متغیر نوع متغیر مشخص نمی‌شود و هر نوع اطلاعاتی اعم از اعداد، رشته ها و ... را می توان در آن ها ذخیره نمود.

در زمان مقداردهی همچ فاصله یا نویسه space نباید قبل یا بعد از نویسه = قرار داشته باشد

```
# valid variable definition var1=10
var2 =10 # invalid: use space before =
var2= 10 #invalid: use space after =
var#1=10 #invalid: use char # in naming not allowed
```

در زمان تعریف متغیر تنها نام آن ذکر می شود اما در زمان ارجاع به متغیر باید از نویسه $ قبل از نام استفاده گردد در غیر اینصورت نام متغیر به عنوان یک رشته تلقی خواهد شد

www.takbook.com

```
var1=10
echo $var1 # output: 10
echo var1 # output: var1
```

```
var1=bah
var2=man
echo $var1+$var2 #output: bah+man
echo var1+var2 #output: var1+var2
```

```
var1=10
var2=10
expr $var1 + $var2 # output: 20
expr var1 + var2  #output: error invalid variable reference
```

```
for c in `seq 1 10`
do
echo "Number: $c" # output: print Number:1 Number:2 ….. 
Number:10
echo "Number: c" # output: print Number:c Number:c ….. 
Number:c
done
```

فرم نوشتن متغیر به صورت $variable، فرم کوتاه شده می باشد. فرم طولانی به صورت ${variable} می باشد. معمولا زمانی از فرم طولانی استفاده می شود که فرم کوتاه ایجاد خطا نماید

www.takbook.com

```
var1="bahman"
echo $var1 # output: bahman
echo ${var1} # output: bahman
echo "var1=$var1" # output: var1=bahman
```

متغیری که مقداردهی نشده باشد حاوی مقدار پوچ یا به اصطلاح NULL می باشد. در زبان اسکریپت نویسی از آنجاییکه اشاره گر به حافظه وجود ندارد مقدار پوچ یعنی رشته تهی.

چنانچه متغیری مقدار دهی شده باشد می توان با دستور unset آن را پوچ نمود

```
echo "var1=$var1" # output: var1=
var1="Linux"
echo "var1=$var1" # output var1=Linux
unset var1
echo "var1=$var1" # output: var1=
```

۶،۲ محدوده متغیرها

مهمترین موضوع درباره متغیرها، بعد از نوع متغیر، محدوده متغیر scope of variable است. محدوده متغیر یعنی خطوطی از برنامه که در آنجا امکان استفاده و فراخوانی متغیر وجود دارد. در زبان های برنامه نویسی برای برنامه بلاک block تعریف می شود مثلا در زبان سی بلاک برنامه بین دو نویسه {} قرار می گیرد و محدوده تعریف متغیر در همان بلاکی است که تعریف شده. به عبارت دیگر متغیر در خارج از بلاک خود قابل فراخوانی نبوده و فراخوانی آن با خطا مواجه می شود.

موضوع محدوده متغیر در عین سادگی می تواند موضوع پیچیده‌ای باشد و برنامه نویس را به زحمت بیندازد. اما در اسکریپت نویسی برای محدوده متغیرها یک قانون وجود دارد **همه متغیرها عمومی Global هستند.** پس هر متغیری که تعریف شود در سراسر فایل یا برنامه قابل رویت و فراخونی می باشد. این قانون کار را بسیار ساده می کند و دیگر نیازی برای تعیین محدوده تعریف متغیر نیست

اما زمان هایی نیاز است که محدوده متغیر تعریف شود و متغیر خارج از محدوده مورد نظر قابل فرخوانی و رویت نباشد. در اسکریپت نویسی متغیرها را تنها در محدوده توابع می توان محدود نمود. به عبارت دیگر

www.takbook.com

مانند سایر زبان ها بلاک کد تعریف نشده است و صرفا می توان متغیر را در محدوده تابع محلی نمود. برای تعریف یک متغیر به صورت محلی می بایست قبل از تعریف و آن از کلمه کلیدی local استفاده نمود

```
local var_name=<var_value>
```

به مثال زیر دقت کنید

```
local_variables.sh
#!/bin/bash
# Experimenting with variable scope
var_change () {
local var1='local 1'
echo Inside function: var1 is $var1 : var2 is $var2
var1='changed again'
var2='2 changed again'
}
var1='global 1'
var2='global 2'
echo Before function call: var1 is $var1 : var2 is $var2
var_change
echo After function call: var1 is $var1 : var2 is $var2


#> ./local_variables.sh
Before function call: var1 is global 1 : var2 is global 2
Inside function: var1 is local 1 : var2 is global 2
After function call: var1 is global 1 : var2 is 2 changed again
```

www.takbook.com

۷دستورات شرطی

دستورات شرطی در برنامه نویسی، دستوراتی هستند که روند اجرای برنامه را تغییر می دهند. مثلا اگر ساعت قبل از ۱۲ ظهر است عبارت "صبح بخیر" ارسال شود و در غیر اینصورت عبارت "عصر بخیر" ارسال گردد. پس عبارت شرطی روند اجرای برنامه را تغییر می دهد که متعاقبا باعث تغییر در خروجی برنامه نیز می تواند بشود

ساختار دستور شرطی به شکل زیر می باشد

```
if  [ condition expression ]
then
command
command
…
else
command
command
…
fi
```

دستور شرط ابتدا مقدار condition expression را بررسی می کند چنانچه مقدار آن برابر true بود دستورات بین then تا else را اجرا می کند و اگر مقدار آن false بود دستورات بین else تا fi را اجرا می کند.

- در اسکریپت نویسی انتهای برخی دستورات بلاکی با کلمه دستور به صورت معکوس مشخص می شود مثلا if-fi یا case-esac

توجه شود که حتما می بایست قبل و بعد از نویسه‌های [,] فاصله باشد و این نویسه‌ها نباید به صورت چسبیده به کلمات قبل یا بعد استفاده شوند

www.takbook.com

```
if [ $name = "bahman" ]
then
echo "You are authorized"
else
echo "You are not authorized"
fi

if [$name = "bahman"]  #synatx error
then
echo "You are authorized"
else
echo "You are not authorized"
fi
```

۷/۱ عبارت شرطی

همانطور که در مثال قبل دیده شد دستور شرطی همراه عبارت شرطی می‌باشد. عبارت شرطی عبارتی است محاسباتی که نتیجه آن یکی از دو مقدار بولی true یا false خواهد بود. در واقع جبر بولین که در ریاضیات مطرح می شود پایه عبارات شرطی است لذا توصیه می شود در حد آشنایی جبر بولین، ریاضیات مطالعه شود تا تسلط بیشتری در عبارات شرطی به دست آورید

غالب عبارات شرطی مورد استفاده در اسکریپت نویسی به شرح ذیل می باشند

۱. و منطقی : and دو عبارت که با هم و منطقی می شوند؛ تنها زمانی مقدار true برمی گردانند که هر دو مقدار true داشته باشند. دو عبارت با استفاده از عملگر a- با هم و منطقی می شوند مثلا $exp1 -a $exp2
۲. یا منطقی : and دو عبارت که با هم یا منطقی می شوند تنها زمانی مقدار false برمی گردانند که هر دو مقدار false داشته باشند. دو عبارت با استفاده از عملگر o- با هم یا منطقی می شوند مثلا $exp1 -o $exp2
۳. نقیض منطقی : نقیض منطقی مقدار یک عبارت را معکوس می‌کند یعنی از true به false و بر عکس. مثلا اگر مقدار عبارت exp1 برابر true باشد نقیض آن false خواهد بود. عملگر نقیض عملگر واحد می‌باشد یعنی برای محاسبه فقط یک عملوند یا ورودی می پذیرد

در ادامه جدول محاسبات عملگرهای منطقی ارائه شده است

www.takbook.com

!	T
T	F
F	T

جدول نقیض منطقی

-o	T	F
T	T	T
F	T	F

جدول یا منطقی

-a	T	F
T	T	F
F	F	F

جدول و منطقی

جدول ۱ - جدول محاسبات منطقی

در عبارت های شرطی اغلب مواقع هدف مقایسه دو متغیر می باشد که براساس آن تصمیم گیری شود. در زمان مقایسه دو متغیر، یکی از دو حالت زیر همواره قابل توجه است

۱. متغیرها از جنس عدد می باشند : در این صورت برای مقایسه دو عدد باید از عملگرهای زیر استفاده شود
   ۱/۱. lt- برای مقایسه کوچکتر بودن، مشابه عملگر < در ریاضی
   ۱/۲. le- برای مقایسه کوچکتر یا برابر بودن، مشابه ≤
   ۱/۳. eq- برای مقایسه برابری، مشابه == در برخی زبان های برنامه نویسی
   ۱/۴. gt- برای مقایسه بزرگتری، مشابه >
   ۱/۵. ge- برای مقایسه بزگتری یا برابری، مشابه ≥
   ۱/۶. ne- عدم برابری دو مقدار، مشابه =! در برخی زبان های برنامه نویسی

۲. متغیرها از جنس رشته می باشند. در این صورت عملگرهای زیر قابل استفاده می باشند
   ۲/۱. n- برای بررسی عدم صفر بودن طول رشته
   ۲/۲. z- جهت بررسی صفر بودن طول رشته یا رشته تهی
   ۲/۳. = مشابه بودن دو رشته
   ۲/۴. =! عدم تشابه دو رشته

www.takbook.com

۸حلقه ها

۸/۱ حلقه for

یکی دیگر از روندها در برنامه نویسی حلقه می باشد. حلقه به معنی تکرار اجرای دستورات به تعداد مشخصی می باشد. مثلا جمع کردن تعدادی عدد و محاسبه حاصل جمع آن ها.

```
#!/bin/sh
a="10 20 30 40 50"
sum=0
for d in $a
do
    sum=$((sum + d))
done
echo "sum=$sum"
```

در مثال بالا از دستور حلقه for استفاده شده است. این دستور حلقه بدنه خود را که دستورات بین do و done است به تعداد مشخصی اجرا می کند. این تعداد برابر با تعداد اعضای آرایه اشاره شده می‌باشد. در این مثال آرایه شامل اعداد 50 40 30 20 10 می‌باشد پس بدنه حلقه که دستور sum=$((sum +d)) می باشد ۵ بار اجرا می شود. علاوه بر بدنه حلقه متغیری نیز در دستور وجود دارد که به آن شمارنده counter نیز می گویند در این مثال متغیر d شمارنده حلقه می باشد. شمارنده در هر بار اجرا مقداری برابر با یکی از اعضای آرایه را به خود می گیرد. بنابر این حلقه بالا به شکل زیر اجرا می گردد

دستور	مقدار شمارنده	شماره مرحله
sum=$((sum + 10))	10	۱
sum=$((sum + 20))	20	۲
sum=$((sum + 30))	30	۳
sum=$((sum + 40))	40	۴
sum=$((sum + 50))	50	۵

ساختار حلقه for مطابق شرح زیر می باشد

www.takbook.com

```
for var in <list>
do
<commands>
done
```

و یک مثال دیگر

```
for_loop.sh
#!/bin/bash
# Basic for loop
names='Stan Kyle Cartman'
for name in $names
do
echo $name
done
echo All done
```

نتیجه اجرا

```
#> ./for_loop.sh
Stan
Kyle
Cartman
All done
#>
```

۸/۲ سری مقادیر

در حلقه for می توان از سری مقادیر نیز استفاده نمود

www.takbook.com

```
#!/bin/bash
# Basic range in for loop
for value in {1..5}
do
echo $value
done
echo All done
```

در این مثال {1..5} به معنای اعداد ۱ الی ۵ می باشد و اگر می خواستیم برای آن پله افزایش یا کاهش در نظر بگیریم می توانستیم بنویسیم {1..5..1} یعنی از ۱ الی ۵ با گام یک .

مثال دیگر {10..0..2} یعنی از ۱۰ تا ۰ با گام ۲. مثال زیر

```
for_loop.sh
#!/bin/bash
# Basic range with steps for loop
for value in {10..0..2}
do
echo $value
done
echo All done
```

خروجی اجرای فایل بالا به صورت زیر خواهد بود

```
#>./for_loop.sh
10
8
6
4
2
0
All done
```

www.takbook.com

یه مورد جذاب از کاربرد سری مقادیر لیست فایل های دایرکتوری می باشد

```
convert_html_to_php.sh
#!/bin/bash
# Make a php copy of any html files
for value in $1/*.html
do
cp $value $1/$( basename -s .html $value ).php
done
```

در این مثال از تمامی فایل ها با پسوند html یک کپی با پسوند php ایجاد خواهد کرد

۸/۳ خروج و ادامه حلقه

در خیلی از مواقع نیاز می شود که اجرای حلقه در نقطه‌ای قبل از پایان بدنه حلقه متوقف شده و از حلقه خارج شده یا اجرا به ابتدای بدنه برگردد. برای خروج از حلقه در هر نقطه دلخواه از دستور break استفاده می‌شود و برای بازگشت به ابتدای بدنه و اجرای دستورت با مقدار بعدی شمارنده از دستور continue استفاده می شود

www.takbook.com

break_5.sh

```
#!/bin/bash
# Make a backup set of files
for value in {1..10}
do
echo $value
if [ $value -gt 5 ]
then
break
   fi
done
echo "All Done"
```

نتیجه اجرای برنامه:

```
#>./break_5.sh
1
2
3
4
5
6
All Done
```

همانطور که دیده می شود اجرای حلقه تا انتهای سری نرفته و بعد از ۶ متوقف شده است

www.takbook.com

```
break_5.sh
#!/bin/bash
# Make a backup set of files
for value in {1..10}
do
if [ $value -gt 5 -a $value -lt 9 ]
then
continue
   fi
echo $value
done
echo "All Done"
```

نتیجه اجرای برنامه:

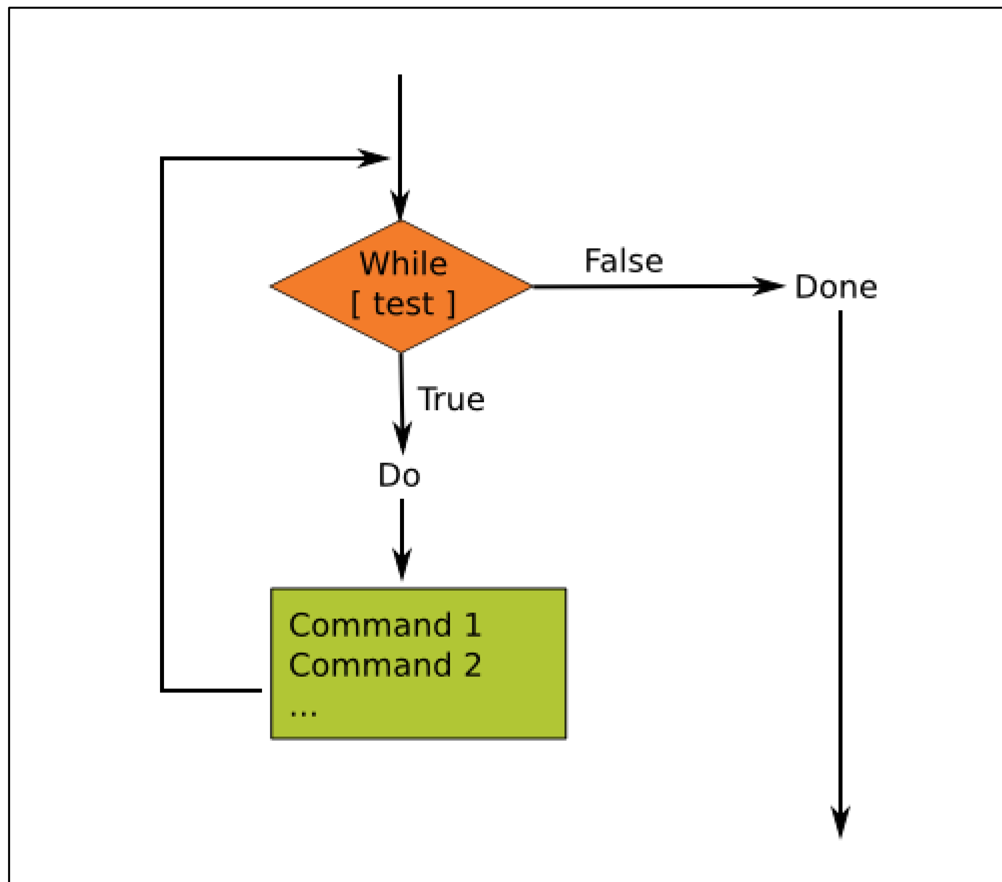
```
#>./break_5.sh
1
2
3
4
5
6
9
10
All Done
```

با توجه به شرط حلقه، اجرای حلقه برای مقادیر ۷ و ۸ صرفنظر شده و در خروجی دیده نمی شوند

۸,٤ حلقه while

www.takbook.com

حلقه while جهت اجرای بدنه به تعداد نامعلوم می باشد. در واقع برخلاف حلقه for که تعداد دفعات اجرا مشحص می باشد در این حلقه یک شرط تعیین کننده پایان دفعات اجرا می باشد



شکل ۳- حلقه while

while_loop.sh

```
#!/bin/bash
# Basic while loop
counter=1
while [ $counter -le 10 ]
do
echo $counter
((counter++))
done
echo All done
```

www.takbook.com

حلقه بالا تا زمانی که متغیر counter کمتر از ۱۰ باشد اجرا می گردد

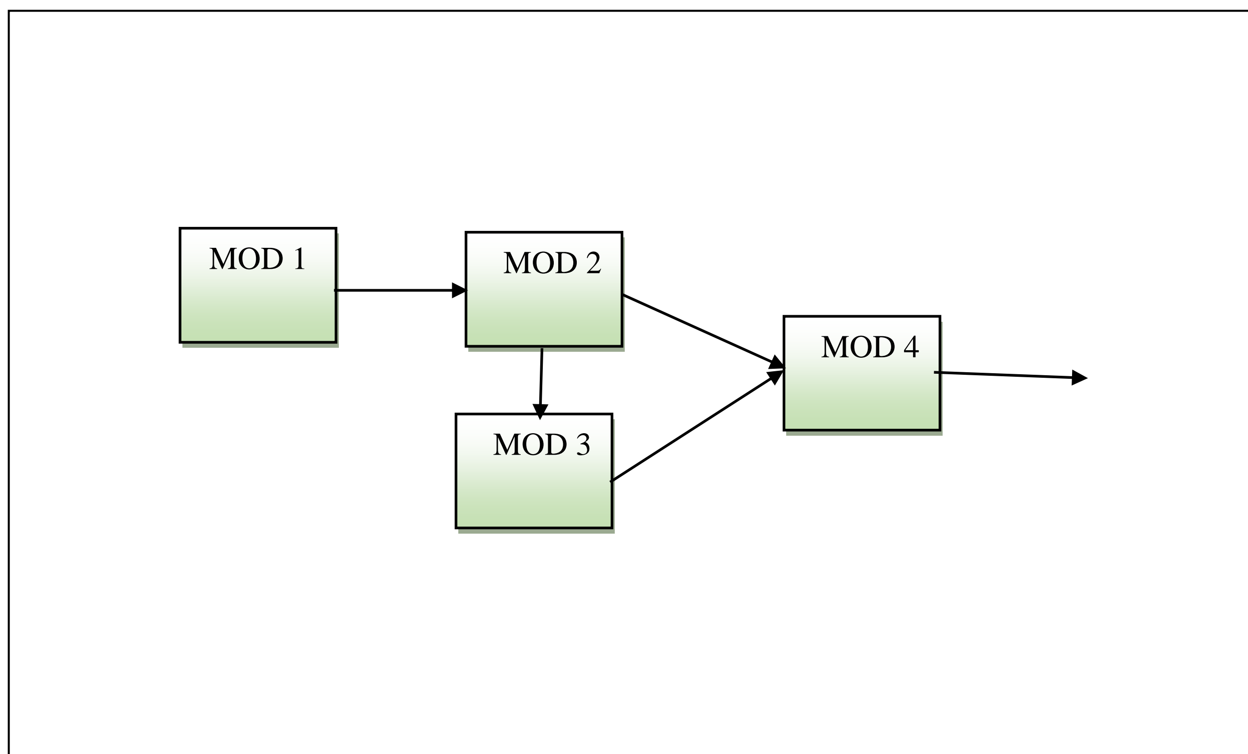
```
#> ./while_loop.sh
1
2
3
4
5
6
7
8
9
10
All done
```

www.takbook.com

۹ توابع

۹,۱ تعریف

توابع یکی از بهترین روش های کدنویسی پیمانه ای Modular Programming است. در این روش برنامه به اجزای مستقلی تقسیم می شود که هر جز می تواند مستقل از باقی اجرا شود. هر جز تعدادی ورودی دریافت کرده و پس از اجرا و پردازش تعدادی خروجی تولید می کند و با قرار دادن متوالی این قطعات برنامه اصلی شکل می گیرد



شکل ۴ - نمایی از برنامه نویسی پیمانه ای

بنابراین می توان تابع را به صورت زیربرنامه نیز تعریف کرد. در دنیای برنامه‌نویسی به زیربرنامه procedure یا روال می گویند. روال با تابع یک تفاوت اساسی دارد. تابع حتما باید خروجی داشته باشد مثلا تابع محاسبه فاکتوریل که یک عدد را به عنوان ورودی گرفته و خروجی آن عددی، که برابر فاکتوریل ورودی است، تولید می شود اما در روال یا زیربرنامه الزامی به وجود خروجی وجود ندارد. مثال یک روال می تواند ایجاد یک فایل در صورت عدم وجود باشد. به این ترتیب این روال وجود یا عدم وجود فایلی را بررسی کرده و در صورت عدم وجود آن را ایجاد می کند و در نتیجه خروجی خاصی تولید نمی شود.

www.takbook.com

در دنیای برنامه نویسی همواره توصیه می شود روال ها نیز به صورت تایع پیاده سازی شود یعنی حتما خروجی داشته باشد. حداقل خروجی که می توان تصور کرد مقدار بولین به عنوان عملکرد موفق یا ناموفق زیرنامه است

در ادامه ساختار تعریف تابع آورده شده است.

```
function_name () {
   list of commands
}
```

همانطور که دیده می شود برای تعریف تابع نام تابع کافی است. بر خلاف زبان های برنامه نویسی نیازی به تعریف پارمترهای ورودی نیست. در دامه نحوه ارسال و استفاده از پارامترهای ورودی بیان خواهد شد. در اینجا دیده می شود به راحتی با نوشتن نام تابع و قرار دادن دو علامت () بلافاصله پس از آن تابع تعریف شده است

```
HelloFun.sh
#!/bin/sh
# Define your function here
Hello () {
   echo "Hello World"
}

# Invoke your function
Hello

#>./HelloFun.sh
Hello World
#>
```

فراخوانی تابع نیز مطابق مثال بالا با نام تابع به تنهایی انجام می گردد و نیازی به پیشوند و یا پسوند ندارد.

www.takbook.com

اما برای ارسال پارامتر به تابع، نیازی به تعریف پارامترها در زمان تعریف تابع نیست و پارامترها به صورت $1, $2, …, $n در تابع قابل فراخوانی و استفاده هستند. در زمان فراخوانی تابع نیز پارامترها با ترتیب در جلوی نام تابع آورده و با علامت جای خالی space از همدیگر جدا می شوند

HelloFunParam.sh

```
#!/bin/sh
# Define your function here
Hello () {
   echo "Hello World $1 $2"
}
# Invoke your function
Hello Zara Ali


#>./HelloFunParam.sh
Hello World Zara Ali
```

فراخوانی توابع به صورت تو در تو نیز امکان پذیر است

www.takbook.com

```
NestedFunc.sh
#!/bin/sh
# Calling one function from another
number_one () {
   echo "This is the first function speaking..."
   number_two
}


number_two () {
   echo "This is now the second function speaking..."
}


# Calling function one.
number_one


#>./NestedFunc.sh
This is the first function speaking...
This is now the second function speaking...
```

۹/۲ بازنویسی دستورات

یکی از تکنیک های جذاب استفاده از توابع بازنویسی دستورات لینوکس می باشد. به عنوان مثال فرض کنید در اسکریپتی که می نویسید همیشه نیاز دارد دستور ls به همراه پارمترهای lh فراخوانی شود. در حالت عادی در خط فرمان از دستور زیر می توان به این منظور استفاده کرد

www.takbook.com

```
alias ls='ls -lh'
#> ls
total 12k
drwx------ 2 bahman bahman 4.0K ۱۳:۳۸ ۱۸ آوریل Acquirer
drwx------ 2 bahman bahman 4.0K ۱۳:۳۸ ۱۸ آوریل api
drwx------ 2 bahman bahman 4.0K ۱۱:۳۲ ۱۹ آوریل Cards
```

دستور alias برای تعریف دستورات مستعار جدید به کار می رود. در مثال بالا به خط فرمان اعلام می شود که دستور ls از این به بعد مستعار ls -lh می باشد و در نتیجه اجرای دستور ls معادل ls -lh می باشد

البته می توان نام مستعار جدید نیر تعریف نمود

```
alias list='ls -lh'
#> list
total 12k
drwx------ 2 bahman bahman 4.0K ۱۳:۳۸ ۱۸ آوریل Acquirer
drwx------ 2 bahman bahman 4.0K ۱۳:۳۸ ۱۸ آوریل api
drwx------ 2 bahman bahman 4.0K ۱۱:۳۲ ۱۹ آوریل Cards
```

با روشی مشابه دستور alias می توان در اسکریپت نیز دستورات خط فرمان را بازنویسی نمود

```
override.sh
#!/bin/bash
# Create a wrapper around the command ls
ls () {
command ls -lh
}
ls
```

www.takbook.com

در قطعه کد بالا دستور ls در خط انتهایی فراخوانی تابع ls می باشد که در این تابع با استفاده از دستور command اصل فرمان ls اجرایی می گردد

۹/۳ ارسال پارامتر غیر مستقیم Indirect

در زبان های برنامه نویسی ارسال پارامتر به تابع به چندین روش امکان پذیر است که تمامی این روش های در دو دسته کلی قابل بررسی می باشند

۱. ارسال مقدار به عنوان پارامتر
۲. ارسال آدرس به عنوان پارامتر

در زبان‌های مبتنی بر اسکریپت غالبا امکان استفاده از روش دوم وجود ندارد. روش ارسال آدرس قدرت برنامه‌نویسی را افزایش می دهد اما به تسلط کافی بر زبان برنامه نویسی نیز نیاز دارد.

در زبان های مبتنی بر اسکریپت از روش های معادل استفاده می شود که به شما امکان ارسال پارامتر شبیه ارسال آدرس را می دهد. روشی که در اسکریپت نویسی لینوکس معرفی شده است ارجاع غیر مستقیم می باشد. در این روش متغیر حاوی نام متغیر دیگری است و در زمان ارسال می توان به جای مقدار متغیر ، مقدار متغیری که نام آن ذخیر شده است ارسال شود مثلا متغیر Name=var1 در اینجا var1 مقدار متغیر Name در نظر گرفته نمی شود بلکه در برنامه متغیری به نام var1 نیز تعریف می شود مثلا var1=10 حالا اگر در دستور echo ${!Name} اجر شود خواهید دید که به جای var1 در خروجی 10 دیده می شود یعنی دستور echo در ورودی خود مقدار متغیر var1 را خواهد دید. به مثال زیر دقت کنید

www.takbook.com

ind-func.sh

```
#!/bin/bash
# Passing an indirect reference to a function.
echo_var ()
{
echo "$1"
}
message=Hello
Hello=Goodbye
echo_var "$message"          # Hello
# Now, let's pass an indirect reference to the function.
echo_var "${!message}"       # Goodbye
echo "-------------"


#>./ind-func.sh
Hello
Goodbye
-------------
```

دقت کنید که متغیر message متغیر عادی است که می توان مقدار آن را در خروجی نوشت، دستور echo "$messgae"، در همان حال می توان آن را به عنوان اشاره‌گر به متغیر دیگر، Hello، نیز در نظر گرفت اما باید حتما متغیر دوم در برنامه تعریف شده باشد در غیراینصورت خطا ایجاد خواهد شد.

بنابراین نویسه ! قبل از نام متغیر یعنی ارسال غیر مستقیم و البته همانطور مه در مثال دیده می شود جهت وضوح برنامه بهتر است در اینطور مواقع از فرم کامل نام متغیر، ${var_name}، استفاده شود.

۹/۴ توابع بازگشتی Recursive

توابع بازگشتی، توابعی هستند که خود را فراخوانی کنند. معمولا این فراخوانی با بررسی شرط خاصی انجام می شود چرا که اگر فراخوانی بدون شرط باشد تا بی نهایت ادامه پیدا خواهد کرد که امان پذیر نمی باشد.

یک مثال کلاسیک از توابع بازگشتی محاسبه فاکتوریل می باشد

```
5! = 4 * 3 * 2 * 1 = 120
```

www.takbook.com

در محاسبه فاکتوریل عملیات ضرب آنقدر ادامه پیدا می کن تا به عدد ۱ یا ۰ برسیم، بر اساس تعریف فاکتوریل اعداد ۰ و ۱ برابر ۱ می باشد

```
1! = 1
0! = 1
```

پس می توان محاسبه فاکتوریل را به صورت بازگشتی تعریف کرد. **فاکتوریل هر عدد برابر است با حاصل ضرب آن عدد در فاکتوریل عدد کوچکتر از خود**. این تعریف، مفهوم تابع بازگشتی را به خوبی در ذهن متبادر می کند. در ادامه برنامه محاسبه فاکتوریل آورده شده است.

www.takbook.com

```
#!/bin/bash
# factoril.sh : recursive
# ----------algorithm-------------
# Fact(0) =1
# else
#   Fact(j) = j * Fact(j-1)
#
# ---------------------------------
Fact ()
{
  num=$1   # Doesn't need to be local. Why not?
  if [ $num -lt 1 ]
  then
    echo 1  # First two terms are 1 ... see above.
  else
pnum=$(($num – 1))
pterm=$(Fact $pnum)
     echo "$(($num * $pterm))"
  fi
}

Fact 5
echo
```

www.takbook.com

۱۰ آرایه ها

همانطور که قبلا گفته شد در برنامه نویسی همواره در حال ذخیره و بازیابی اطلاعات هستید. برای این منظور از متغیرها استفاده می شود. در برخی مواقع اطلاعاتی که در حال ذخیره یا بازیابی هستند به صورت رشته ای یا مجموعه‌ای می باشند. مثال معدل دانش آموزان یک کلاس، جمعیت شهرهای یک کشور و ...

به این نوع اطلاعات در دنیای برنامه نویسی ارایه گفته می شود. آرایه در برنامه نویسی به صورت سطری تصویر می گردد

جمعیت شهرهای کشور	۶۰	۵۰	۴۰	۳۰	۲۰	۱۰
معدل دانش آموزان	۱۸.۲۵	۱۹.۳۵	۱۷.۳۵	۱۷.۶۳	۱۸.۳۵	۱۹.۳۲

شکل ۱ - نمونه اطلاعات آرایه‌ای

به نمونه‌های بالا آرایه های یک بعدی گفته می شود در واقع آرایه یک بعدی را می توان یک سطر یا یک ستون تصور کرد. می توان آرایه دو بعدی، جدول، سع بعدی و بیشتر نیز تعریف نمود که در اینجا مورد مطالعه نخواهند بود.

برای دسترسی به مقادیر آرایه از ایندکس استفاده می شود. مقادیر آرایه مثا از سمت چپ ایندکس گذاری می شوند. ایندکس آرایه می تواند از صفر یا یک یا هر عدد دیگری شروع شود اما در زبان برنامه نویسی از

	۶۰	۵۰	۴۰	۳۰	۲۰	۱۰
ایندکس آرایه	۶	۵	۴	۳	۲	۱

شکل ۲ - ایندکس آرایه

صفر یا یک شروع می شود. اغلب زبان های از صفر شروع می کنند برخی زبان ها نیز از یک. در اسکریپت-های لینوکس از صفر شروع می شود.

www.takbook.com

۱۰,۱ تعریف ارایه

۱۰,۱,۱ با ایندکس

در زبان اسکریپت نویسی لینوکس تنها از آرایه های یک بعدی پشتیبانی می شود. جهت تعریف آرایه در جلوی نام متغیر باید از نویسه‌ها [] استفاده شود و در بین دو نویسه ایندکس نوشته می شود. جهت دسترسی به مقدار یک ایندکس نیز باید از حالت کامل نام متغیر استفاده شود ${var_name[index]}

```
#!/bin/bash
area[11]=23
area[13]=37
area[51]=UFOs
#  Array members need not be consecutive or contiguous.
echo "area[11] = ${area[11]}"  #  {curly brackets} needed.
echo "area[13] = ${area[13]}"
echo "Contents of area[51] are ${area[51]}."
echo "area[43] = echo ${area[43]}"
echo "(area[43] unassigned)"
# Sum of two array variables assigned to third
area[5]=`expr ${area[11]} + ${area[13]}`
echo "area[5] = area[11] + area[13]"
echo "area[5] = ${area[5]}"
```

در مثال بالا تعریف آرایه به صورت مستقیم با استفاده از ایندکس انجام شده است. برای تعریف نیازی به تعریف تمامی ایندکس ها نیست در اسکریپت ایندکس های تعریف نشده به صورت خالی در نظر گرفته می شود مانند ایندکس ۴۳ در مثال

۱۰,۱,۲ تعریف تمام اعضا

روش دیگر تعریف ارایه ، تعریف تمام اعضا می باشد. در این روش اعضای متغیر بین دو پرانتز و با جدا کننده فضای خای space تعریف می شود (item1 item2 item3 ….)

www.takbook.com

```
# Another array,
# Another way of assigning array variables...
# array_name=( XXX YYY ZZZ ... )
area=( zero one two three four )
echo"area[0] = ${area[0]}"
# first element of array is [0], not [1]
echo "area[1] = echo ${area[1]}"  # [1] is second element
```

۱۰/۱/۳ ترکیبی

در این روش اعضای ارایه بین دو پرانتر تعریف می شوند اما تمامی ایندکس ها تعریف نمی شود. مثلا برای تعریف دو ایندکس ۱۰ و ۲۰ به صورت ([10]=item1 [20]=item2) عمل می شود. به مثال زیر توجه کنید

```
# Yet another array,
# Yet another way of assigning array variables...
# array_name=([xx]=XXX [yy]=YYY...)
area3=([17]=seventeen [24]=twenty-four)
echo "area3[17] = ${area3[17]}"
echo "area3[24] = ${area3[24]}"
```

۱۰،۲ عملگرهای آرایه

زمانی که متغیر از جنس آرایه باشد علاوه بر خواندن ایندکس مشخص از آن برخی اطلاعات دیگر نیز مورد نیاز واقع می شود. همانند طول آرایه، حذف یک عضو و .... برای به دست آوردن این اطلاعات توابع و عملگرهایی تعریف می شوند که به آنها عملگرهای آرایه گفته می شود.

۱۰،۲،۱ دسترسی به اعضای آرایه

یکی از روش های دسترسی به آرایه استفاده از عملگر [] که پیشتر نیز به آن اشاره شد. در اینجا با یک مثال مجددا یادآوری می شود

www.takbook.com

```
#!/bin/bash
# Lines of the poem (single stanza).
Line[1]="I do not know which to prefer,"
Line[2]="The beauty of inflections"
Line[3]="Or the beauty of innuendoes,"
Line[4]="The blackbird whistling"
Line[5]="Or just after."


tput bold
for index in 1 2 3 4 5    # Five lines.
do
  printf "     %s\n" "${Line[index]}"
done
```

در مثال بالا جهت چاپ در خروجی دو دستور جدید معرفی و استفاده شده است.

- tput bold : این دستور فونت خط فرمان را به حالت برجسته Bold تغییر می دهد
- printf : جهت ساختاردهی به رشته خروجی استفاده می شود. با استفاده از این روش می توان ساختار خروجی را در قالب رشته بیان نمود و متغیرهایی که به خروجی ارسال می شوند در ادامه به ترتیب به عنوان پارامتر ورودی ارسال می گردند. در فصول بعدی این تابع با جزییات تشریح خواهد گردید. در مثال فعلی رشته "%s\n   " به معنای چاپ تعدادی فضای خالی سپس متغیر رشته-ای %s و بعد از انتهای خط \n می باشد. متغیر رشته ای نیز به عنوان پارامتر دوم "${Line[index]}" بعد از آن ارسال شده است

روش دوم استفاده از عملگر : می باشد. این عملگر نیز برای دسترسی به مجموعه ای از ایندکس ها می باشد اما به صورت 0: مقدار اولین ایندکس را برمی گرداند. در ادامه این عملگر به صورت کامل بررسی خواهد شد

www.takbook.com

```
#!/bin/bash
array=( zero one two three four five )
# Element 0   1   2    3     4    5
echo ${array[0]}        #  zero
echo ${array:0}         #  zero
echo ${array:1}         #  error
```

۱۰,۲,۲ طول آرایه

برای محاسبه طول در آرایه از عملگر # استفاده می شود. این عملگر طول کل ارایه یا ایندکس اشاره شده را برمی گرداند

```
#!/bin/bash
array=( "zero" "one" "two" "three" "four" "five" )
# Element 0   1   2    3     4    5
echo ${#array[0]}  # length of item 0  = 4
echo ${#array[6]}  # length of item 7 that = 0

echo ${#array[@]}  # length of array = 6
```

۱۰,۲,۳ زیر مجموعه

با استفاده از عملگر :[@] می توان زیرمجموعه ای از آرایه را به دست آورد. این عملگر دو عملوند به عنوان ورودی دریافت می کند. ایندکس شروع و طول مثلا ۱:۳[@] یعنی از ایندکس ۱ به طول ۳ ایتم. اگر تنها [@] به عنوان عملگر استفاه شود کل آرایه برگردانده می شود

www.takbook.com

```
#!/bin/bash
arrayZ=( one two three four five)
echo ${arrayZ[@]}       # one two three four five
echo ${arrayZ[@]:1:3}  # two three four
echo ${arrayZ[@]:3}     # four five
```

۱۰,۲,٤ جایگزینی

عملگر جایگزینی امکان تغییر مقادیر ارایه را می دهد. با استفاده از عملگر [old expr]/[new expr]/ می توان در تمامی مقادیر به دنبال old expr جستجو کرد و با مقدار new expr جایگزین نمود.

```
#!/bin/bash
arrayZ=( one two three four five fivefive)
echo ${arrayZ[@]/fiv/XYZ}  # one two three four XYZe XYZefive
echo ${arrayZ[@]//fiv/XYZ} # one two three four XYZe XYZeXYZe
```

همانطور که در مثال دیده می شود چنانچه عملگر به صورت /exp/exp استفاده شود در هر ایتم اولین تطابق جایگزین می شود اما چنانچه عملگر به صورت //exp/exp استفاده شود در هر ایتم تمامی تطابق‌های جایگزین خواهند شد

چنانچه در عملگر عبارت اول تهی باشد، عملگر تطابق را پاک خواهد کرد

```
#!/bin/bash
arrayZ=( one two three four five five)
echo ${arrayZ[@]//fi/}  # one two three four ve ve
```

همچنین می توان تطابق را در ابتدای یا انتها عبارت بررس نمود. چنانچه عبارت مورد جستجو به صورت #<exp> بیان شود به معنی آن است که تطابق زمانی واقع می شود که رشته exp در ابتدای عبارت دیده شود و اگر عبارت مورد جستجو به صورت %<exp> بیان شود تطابق در صورتی که در انتهای عبارت رخ دهد تایید می گردد

www.takbook.com

```
#!/bin/bash
arrayZ=( one two three four five five)
echo ${arrayZ[@]/#iv/XYZ}   # one two three four five five
echo ${arrayZ[@]/#fi/XYZ}   # one two three four XYZve XYZve
echo ${arrayZ[@]/%fi/XYZ}   # one two three four five five
echo ${arrayZ[@]/%ve/XYZ}   # one two three four fiXYZ fiXYZ
```

- برای عبارت جایگزین از فراخونی تابع نیز می توان استفاده کرد

```
#!/bin/bash
arrayZ=( one two three four five five)


replacement() {
    echo -n "!!!"
}


# on!!! two three!!! four fiv!!! fiv!!!
echo ${arrayZ[@]/%e/$(replacement)}
```

۱۰٫۲٫۵ حذف ایتم

برای حذف ایتم از عملگر (تابع) unset استفاده می شود. این تابع مقدار ایتم را گرفته و از آرایه حذف می کند مثلا arrayZ[1] ایتم ۱ را حذف می کند و اگر بدون ایندکس فراخوانی شود آرایه حذف، پاک، می شود

www.takbook.com

```
#!/bin/bash
color=(red blue green brown yellow)
echo ${color[@]}  # red blue green brown yellow
unset color[1]
echo ${color[@]} # red green brown yellow
unset color
echo ${color[@]} #
```

www.takbook.com

۱۱دستورات تست

دستورات شرطی بخش قابل توجه‌ای از برنامه را به خود اختصاص می دهند چرا که همواره در برنامه های در حال تصمیم گیری جهت ادامه روند برنامه هستیم. در بخش های قبلی دستور if-else-fi معرفی گردید اما جهت تسریع در نوشتن اسکریپت در نسخه های جدیدتر شل ها ساختارهای جدیدی معرفی شده است که نوشتن شرطها را در برنامه سریعتر و خوانا تر می کند

۱۱/۱ درست چیست؟

ابتدا به برنامه زیر دقت کنید

truth.sh

```
#!/bin/bash
echo
echo "Testing \"0\""
if [ 0 ]      # zero
then
  echo "0 is true."
else          # Or else ...
  echo "0 is false."
fi           # 0 is true.
echo


echo "Testing \"1\""
if [ 1 ]      # one
then
  echo "1 is true."
else
  echo "1 is false."
fi           # 1 is true.
echo
echo "Testing \"-1\""
if [ -1 ]     # minus one
then
  echo "-1 is true."
else
  echo "-1 is false."
fi           # -1 is true.
echo
echo "Testing \"NULL\""
if [ ]        # NULL (empty condition)
then
  echo "NULL is true."
else
  echo "NULL is false."
fi           # NULL is false.
echo
echo "Testing \"xyz\""
if [ xyz ]    # string
then
```

www.takbook.com

```
  echo "Random string is true."
else
  echo "Random string is false."
fi          # Random string is true.
echo
echo "Testing \"\$xyz\""
if [ $xyz ]   # Tests if $xyz is null, but...
            # it's only an uninitialized variable.
then
  echo "Uninitialized variable is true."
else
  echo "Uninitialized variable is false."
fi          # Uninitialized variable is false.
echo
echo "Testing \"-n \$xyz\""
if [ -n "$xyz" ]          # More pedantically correct.
then
  echo "Uninitialized variable is true."
else
  echo "Uninitialized variable is false."
fi          # Uninitialized variable is false.
echo
xyz=         # Initialized, but set to null value.
echo "Testing \"-n \$xyz\""
if [ -n "$xyz" ]
then
  echo "Null variable is true."
else
  echo "Null variable is false."
fi          # Null variable is false.
echo
# When is "false" true?
echo "Testing \"false\""
if [ "false" ]            #  It seems that "false" is just a string ...
then
  echo "\"false\" is true." #+ and it tests true.
else
  echo "\"false\" is false."
fi          # "false" is true.
echo
echo "Testing \"\$false\""  # Again, uninitialized variable.
if [ "$false" ]
then
  echo "\"\$false\" is true."
else
  echo "\"\$false\" is false."
fi          # "$false" is false.
```

و اما خروجی برنامه

```
#> ./truth.sh
```

www.takbook.com

```
Testing "0"
0 is true.

Testing "1"
1 is true.

Testing "-1"
-1 is true.

Testing "NULL"
NULL is false.

Testing "xyz"
Random string is true.

Testing "$xyz"
Uninitialized variable is false.

Testing "-n $xyz"
Uninitialized variable is false.

Testing "false"
"false" is true.

Testing "$false"
"$false" is false.
```

همانطور که دیده می شود در اسکریپت نویسی اعداد همیشه معادل درست true هستند و هر رشته هم معادل درست می باشد اما متغیری که مقداردهی نشده باشد معادل غلط خواهد بود و همچنین NULL.

همچنین به تفاوت false و $false دقت کنید اولی چون رشته است پس درست است اما دومی $false متغیر نادرست است پس نادرست است. بنابراین همواره در ارزیابی منطقی یک عبارت این موضوع باید مد نظر باشد

۱۱/۲ ساختار (( ))

عبارت 2+2 در اسکریپت خطاست و اگر به صورت x=2+2 نوشته شود یعنی تعریف متغیر x به صورت رشته ای که مقدار آن نیز 2+2 می باشد. پس چگونه مقدار 2+2 محاسبه شود؟

برای محاسبات ریاضی چند روش وجود دارد که در فصل های بعدی به سایر روش ها نیز اشاره خواهد شد اما در این قسمت استفاده از ساختار (( )) معرفی می گردد. برای مثال بالا باید عبارت به صورت

www.takbook.com

"let "x = ((2 + 2)) نوشته شود. دستور let رشته ورودی را صورت عبارت محاسباتی در نظر گرفته و ارزیابی می کند. در مثال مذکور دو عدد با هم جمع شده و در متغیر x ذخیره می شود.

یک مثال دیگر از محاسبات منطقی؛ عبارت "let "num = ((200 || 11)) می باشد. در این عبارت یا منطقی دو عدد ۲۰۰ و ۱۱ محاسبه می شود. همانطور که پیشتر گفته شد مقدار منطقی اعداد همواره true می باشد پس مقدار این محاسبه true یا ۱ خواهد بود و در نتیجه مقدار ۱ در num ذخیره خواهد شد.

- **در لینوکس همواره عدد 0 به معنای success و عدد غیر 0 به معنای شماره خطا می باشد و در در دنیای برنامه نویسی 0 به معنای false و غیر صفر به معنای true می باشد**

حال مثال جالب زیر را در نظر بگیرید

```
#!/bin/nash
var=-2 && (( var+=2 ))
echo $?     # 1
var=-2 && (( var+=2 )) && echo $var
echo $var  # Guess the out put !!
```

خروجی اول مقدار ۱ خواهد بود. چرا؟

اما خروجی دوم چه خواهد بود؟ چرا؟

در مثال زیر می توان ساختار (( )) را بهتر درک نمود

```
#!/bin/bash
(( 0 && 1 ))              # Logical AND
echo $?     # 1     ***
# And so ...
let "num = (( 0 && 1 ))"
echo $num   # 0
# But ...
let "num = (( 0 && 1 ))"
echo $?     # 1     ***
(( 200 || 11 ))           # Logical OR
echo $?     # 0     ***
# ...
let "num = (( 200 || 11 ))"
echo $num   # 1
let "num = (( 200 || 11 ))"
echo $?     # 0     ***
(( 200 | 11 ))            # Bitwise OR
echo $?                # 0     ***
# ...
let "num = (( 200 | 11 ))"
echo $num                 # 203
let "num = (( 200 | 11 ))"
echo $?                # 0     ***
```

www.takbook.com

اما از این ساختار می توان در دستور if-else-fi نیز استفاده نمود

```
#!/bin/bash
var1=5
var2=4
if (( var1 > var2 ))
then #^     ^      Note: Not $var1, $var2. Why?
  echo "$var1 is greater than $var2"
fi    # 5 is greater than 4
```

با استفاده از این ساختار در دستور if-else-fi دیگر برای مقایسه اعداد نیز به استفاده از عملگرهای ,-eq, -lt ... نمی باشد و خوانایی برنامه نیز بهبود چشمگیری پیدا می کند

در مثال زیر تمامی محاسبات منطقی با استفاده از این ساختار بررسی شده‌اند

```
#!/bin/bash


# Arithmetic tests.
# The (( ... )) construct evaluates and tests numerical expressions.
(( 0 ))
echo "Result of \"(( 0 ))\" is $?."        # 1


(( 1 ))
echo "Result  of \"(( 1 ))\" is $?."        # 0


(( 5 > 4 ))                          # true
echo "Result  of \"(( 5 > 4 ))\" is $?."     # 0


(( 5 > 9 ))                          # false
echo "Result  of \"(( 5 > 9 ))\" is $?."     # 1


(( 5 == 5 ))                          # true
echo "Result  of \"(( 5 == 5 ))\" is $?."    # 0
# (( 5 = 5 ))  gives an error message.


(( 5 - 5 ))                          # 0
echo "Result  of \"(( 5 - 5 ))\" is $?."     # 1


(( 5 / 4 ))                          # Division o.k.
echo "Result  of \"(( 5 / 4 ))\" is $?."     # 0


(( 1 / 2 ))                          # Division result < 1.
echo "Result  of \"(( 1 / 2 ))\" is $?."     # Rounded off to 0.

let "t = ((1 == 1))"                 # t = true
let "f = ((1 == 0))"                 # f = false
```

www.takbook.com

```
let "v = (( t && f))"   # Logical and true and false is false or 0
echo "Result of \"( true && false)\" is $v." #  0

let "v = (( t || f))"   # Logical or true and false is true or 1
echo "Result of \"( true && false)\" is $v."   # 1
```

۱۱/۳ ساختار if-elif-fi

ساختار if-else-fi پیشتر معرفی شده در آن ساختار تنها یک شرط بررسی شده و بر اساس آن تصمیم گیری می گردد. اما در برخی مواقع نیاز است که در صورت عدم برقراری شرط اول شرط دوم و بعد سوم و ... نیز ارزیابی گردد.

برای این منظور می توان از ساختار if-else-fi به صورت تو در تو استفاده نمود

```
if [ exp1 ]
then
        commands
else if [ exp2 ]
        then
                commands
        else if  [ exp3 ]
                then
                        commands
                else
                        commands
                fi
        fi
fi
```

این ساختار تو در تو خوانایی مناسبی نداد و اگر تعداد شرایط نیز یاد شود ساختار پیچیده ای خواهد بود لذا

ساختار if-elif-fi معرفی می شود. این ساختار همان ساختار تو در تو می باشد که به نوعی خلصه نویسی شده و در نتیجه خوانایی بالاتریی خواهد داشت

```
if [ exp1 ]
then
        commands
elif [exp2 ]
then
        commands
elif [ exp3 ]
then
        commands
....
```

www.takbook.com

```
else
        commands
fi
```

پرواضح است که ساختار دوم بسیار واضح تر است. البته قابل ذکر است در صورتی که تعداد شرایط زیاد است باید از دستور case استفاده گردد که در بخش بعدی معرفی خواهد شد

```
ifelif.sh
/!#bin/bash
n=2

if [ $n -eq 1 ]; then
        echo value of n is 1
elif [ $n -eq 2 ]; then
        echo value of n is 2
else
        echo value of n is other than 1 and 2
fi

#> ./ifelif.sh
value of n is 2
```

۱۱/۴ دستور case-esac

ساختار دستور case مطابق شرح ذیل می باشد

```
case EXPRESSION in
CASE_1)
   COMMAND-LIST
;;
CASE_2)
   COMMAND-LIST
;;


CASE_N)
   COMMAND-LIST
;;
....
*)
;;
esac
```

در این ساختار ابتدا عبارت EXPRESSION ارزیابی می شود و حاصل آن یکی از مقادیر CASE_1, CASE_2, … خواهد بود. در این صورت دستورات مربوط به همان قسمت اجرا می شود چنانچه مقدار

www.takbook.com

ارزیابی شده با هیچکدام برابر نبود و حالت (* تعریف شده بود دستور این حالت، که معمولا به عنوان حالت default شناخته می شود، اجرا می گردد.

بنابراین این ساختار حالت خاصی از همان دستور if-elif-fi می باشد که خلاصه و ساده تر شده است. به مثال زیر دقت کنید

```
#!/bin/bash

time=12

# if condition is true
case $time in
9)
   echo Good Morning!
   ;;
12)
   echo Good Noon!
   ;;
17)
   echo Good Evening!
   ;;
21)
   echo Good Night!
   ;;
esac
```

در این مثال ساعت خوانده شده و براساس آن عبارت خروجی چاپ شده است. فرض کنید مقدار time برابر ۱۵ باشد آنگاه خروجی چه خواهد بود؟ چون حالتی برای آن در نظر گرفته نشده است چیزی در خروجی نخواهد بود در این حالت نیاز است که حالت default نیز تعریف شود بنابراین شکل کامل تر برنامه مثال زیر خواهد بود

```
#!/bin/bash

time=15

# if condition is true
case $time in
9)
   echo Good Morning!
   ;;
12)
   echo Good Noon!
   ;;
17)
```

www.takbook.com

```
   echo Good Evening!
   ;;
21)
   echo Good Night!
   ;;
*)
   echo Good Day!
   ;;
esac
```

در اینصورت برای ۱۵ خروجی !Good Day چاپ خواهد شد.

یکی از کاربردهای مناسب این دستور در ارزیابی پارمترهای خط فرمان می باشد که در مثال بعد دیده می شود

```
#!/bin/sh

option="${1}"

case ${option} in
  -f)
        FILE="${2}"
        echo "File name is $FILE"
    ;;
  -d)
        DIR="${2}"
        echo "Dir name is $DIR"
    ;;
  *)
        echo "usage: [-f file] | [-d directory]"
        exit 1 # Command to come out of the program with status 1
    ;;
esac
```

در این مثال در خط فرمان پارامترهای -f و -d ارسال می شود و با استفاده از ساختار case این پارامترها ارزیابی می گردند.

۱۱/۵ دستور test

دستور test از دستورات بسیار پرکاربرد در اسکریپت نویسی است و همانطور که از نام آن نیز بر می آید هدف آن ارزیابی مقادیر ورودی است. این دستور می تواند وجود یا عدم وجود فایل ، دیرکتوری و .. را چک کند، تهی بودن یک رشته را بررسی کند و …

مهمترین ارزیابی هایی که این دستور انجام می دهد به شرح ذیل می باشد

www.takbook.com

- e- : موجودیت یک فایل
- f- : فایل در صورت وجود فابل معمولی است. دایرکتوری نمی باشد
- s- : حجم فایل مخالف صفر است
- d- : فایل در صورت وجود دایرکتوری است. فایل معمولی نمی باشد
- b- : فایل از جنس بلاکی می باشد. فایل های بلاکی فایل هایی هستند که اطلاعات در آن به صورت بلاکی نوشته و خوانده می شود همانند فایل متنی که بر روی هارد ذخیره می شوند. در مقابل این فایل ها فایل های کاراکتری هستند که اطلاعات در آن ها به صورت جریانی از اطلاعات stream ذخیره می شوند مثلا فایل های مربوط به کیبورد، پرینتر و ...
- c- : فایل از جنس کاراکتری می باشد
- h- : فایل از جنس اشاره گر می symbolic link باشد. فایل های اشاره‌گر در واقع فاقد محتوی هستند و به آدرس فایل دیگری اشاره می کنند. مشابه میانبرها shourtcut در ویندوز
- S- : فایل سوکت شبکه می باشد
- r- : کاربر دسترسی خواندن به فایل دارد
- w- : کاربر دسترسی نوشتن به فایل دارد
- x- : کاربر دسترسی اجرا دارد
- u- : فایل با دسترسی کاربر root قابلیت اجرا دارد
- O- : آیا کاربر صاحب فایل است
- G- : آیا کاربر در گروه صاحب فایل است
- N- : آیا بعد از آخرین خواندن، فایل تغییر کرده است
- f1 -nt f2 : آیا فایل f1 از فایل f2 جدیدتر است
- f1 -ot f2 : آیا فایل f1 از فایل f2 قدیم‌تر است
- f1 -ef f2 : آیا فایل‌های f1 و f2 یکسان هستند
- ! : همانند عملگر نقیض در عبارت های منطقی نتیجه تست را معکوس می کند
- z- : آیا رشته ورودی تهی است یا طول آن صفر است
- n- : آیا رشته ورودی غیر تهی است. نقیض دستور z-
- a- : و منطقی دو پارامتر ورودی e1 && e2
- o- : یا منطقی دو پارامتر ورودی e1|| e2
- عملگرهای مقایسه : تمامی عملگرهای مقایسه نیز در این دستور معتبر هستند. عملگرهای -lt, -le, -gt, -ge, -ne -eq, <, <=, >, >=, ==, =, !=

رایج ترین تکنیک استفاده از دستور test به صورت زیر می باشد

www.takbook.com

- test exp1 || command : در این حالت اگر ارزیابی exp1 برابر true باشد command اجرا نخواهد شد در غیر اینصورت command اجرا می شود. مثلا ”test 1 -e1 1 || echo “Hello، در این حالت دیگر دستور echo اجرا نمی شود اما در دستور ”test 1 -eq 0 || echo “Hello چون ۱ برابر صفر نیست پس ارزیابی نادرست و دستور دوم یعنی echo اجرا می شود.
- test exp1 && command : دقیقا برعکس حالت قبل یعنی اگر exp1 برقرار باشد انگاه command اجرا می شود در غیر اینصورت اجرا نمی شود مثلا در ”test 1 -eq 0 && echo “Hello چون ۱ مخالف صفر است پس ارزیابی نادرست است و در نتیجه دستور echo اجرا نخواهد شد

۱۱/۶ ساختار [[ ]]

در واقع [[ ]] ساختار نیست و اگر با دستور type بررسی کنید خواهید دید که]] یک دستور است

```
#>type if
if is a shell keyword

#>type [
[ is a shell builtin

#>type [[
[[ is a shell keyword
```

همانطور که دیده می شود]] همانند if کلمه کلیدی شناخته شده است. به هر روی در اسکریپت نویسی در دستور if-else-if می توان به جای [ ] از [[ ]] استفاده نمود. دستور]] بسط دستور test می باشد یا به عبارت دیگر نام مستعار دستور test می باشد بنابراین تمامی سوییچ های دستور test در این دستور قابل استفاده است و خود باعث خوانایی بیشتر برنامه می شود.

مثلا در ساختار [ ] استفاده از نویسه‌های <، > و ... مجاز نیست و برنامه با خطا می شود در حالیکه در ساختار یا دستور]] مجاز است که خود باعث خوانایی برنامه می گردد به مثال زیر دقت کنید

```
1.  if [[ “$a” > “$b” ]]

2.  if [ “$a” \> “$b” ]
```

در خط اول از ساختار]] استفاده شده است و خوانایی به وضوح بالاست اما در خط دوم جهت جلوگیری از خطای مفسر عملگر به همراه کاراکتر فرار <\ استفاده شده است که باعث از بین رفتن خوانایی شده است

```
if [[ -z "$string" ]]; then
  echo "String is empty"
```

www.takbook.com

```
elif [[ -n "$string" ]]; then
  echo "String is not empty"
else
  echo "This never happens"
fi
```

در مثال بالا اعتبار سنجی یک رشته بررسی شده است در شرط اول تهی بودن رشته بررسی شده است و در رشته دوم عدم تهی بودن رشته در نتیجه هیچ گاه حالت else نباید رخ دهد چرا که بالاخره یک رشته یا تهی است یا نیست

به مثال زیر دقت کنید

test.sh

```
#!/bin/bash
i=1
while [[ $i -le 10 ]] ; do
   echo "$i"
  (( i += 1 ))
done

#>./test.sh
1
2
3
4
5
6
7
8
9
```

همانطور که دید می شود استفاد از ساختارهای [[ ]] و (( )) خوانایی برنامه را بسیار بالا برده است.

www.takbook.com

۱۲عبارات منظم Regular Expression

عبارت‌های منظم ترکیب‌هایی خاص از حروف و علامت‌ها هستند که برای جستجو و مقایسه‌ی رشته‌ها استفاده می‌شوند. این عبارت‌ها گاهی با نام «Regex» و گاهی با «Regexp» نیز شناخته می‌شوند. استفاده از این عبارت‌ها می‌تواند حجم کدنویسی را تا اندازه‌ی زیادی کاهش دهد. برای مثال ارزیابی ورودی کاربر برای شباهت به یک نشانی ایمیل یا جستجوی یک فایل برای یافتن شماره تلفن‌های موبایل با عبارت‌های منظم امکان‌پذیر است.

تقریبا تمام زبان‌های برنامه‌نویسی از Regex پشتیبانی می‌کنند یا این ویژگی با استفاده از کتابخانه‌های جانبی در آن‌ها امکان‌پذیر است. از آن جمله زبان‌های رایج برنامه‌نویسی وب مانند php، Javascript عبارت‌های منظم را پردازش کنند.

یک عبارت منظم خود نیز یک رشته می باشد که کاراکترهای آن معنای خاص دارند و ترتیب و ترکیب آن ها بیانگر توصیف عبارات یا رشته مورد جستجو می باشد. ترتیب و ترکیب این کاراکترها از قوانین خاصی پیروی می کنند که در ادامه بیان و بررسی می شوند

۱۲/۱ قوانین عبارات منظم

قوانین عبارات منظم همانند دستورات زبان برنامه نویسی می باشند با این تفاوت که هدف این دستورات مشابهت یابی یا تطابق است نه محاسبات

۱. تطابق کامل زیر شته : برای پیدا کردن یک زیر شته مثلا abcd خود زیر رشته بدون هیچ نویسهی استفاده می شود. یعنی abcd خود یک عبارات منظم است که دقیقا همین زیر رشته را جستجو می کند
۲. ابتدا و انتها : نویسه $ به معنای انتهای رشته و نویسه ^ به معنای ابتدای رشته می باشد.

عبارت منظم	تطابق	عدم تطابق
^abcd	**abcd**Hij	bcdfHij
abcd$	ij**Habcd**	Habdc

۳. نویسه ستاره * : به معنای تکرار به تعداد دلخواه . مثلا a* به معنی تکرار a به هر تعداد.

عبارت منظم	تطابق	عدم تطابق
bahman*	**bahmannnnn**	ali

www.takbook.com

۴. نویسه نقطه . : هر کاراکتر غیر از \n. مثال .Aa : زیر رشته Aa که پس از آن حداقل یک کاراکتر باشد مانند Aad, Aaddd

عبارت منظم	تطابق	عدم تطابق
Aa.	**Aab**cdef **AaP**old	ali

۵. [] : مجموعه ای از کاراکترها که که هر کدام معتبر می باشد. مثال [123ab]: هر کدام از کاراکترهای 1, 2, 3, a, b دیده شود این عبارت مطابقت دارد. به مثال‌های زیر دقت کنید

- [xyz] : هر کدام از کارکترهای x, y, z
- [c-h] : هر یک از کارکترهای بین c الی h. یعنی c,d,e,f,g,h در عمل معادل [cdefgh]
- [a-z0-9] : هر یک از کاراکترهای بین a الی z و بین 0 الی ۹

- داخل کروشه می توان از نویسه ^ نیز استفاده نمود. این نویسه به معنای نقیض یا " غیر از این " می باشد. مثلا [^a-z] یعنی هر نویسه‌ای غیر از حروف a تا z
- نویسه نقطه . در داخل کروشه فقط نویسه . را پیدا می‌کند و مانند بند ۹ عمل نمی کند.

۶. علامت لوله یا پایپ | : علامت پایپ «|» در عبارت به معنای «یا» کاربرد دارد. مثلا عبارت منظم «abc|def» یکی از رشته‌های "abc" یا "def" را جستجو می‌کند

۷. تطابق گروهی : استفاده از پرانتز در اطراف گروهی از نویسه‌ها، می‌تواند دامنه‌ی عملکرد پایپ «|» را محدود کند. مثال عبارت منظم «a(bc)|(de)» معادل "abc" یا "ade" خواهد بود.

۸. نویسه‌های ترکیبی : برخی نویسه‌های خلاصه یک عبارت منظم می باشند که جهت تسریع در نوشتن می توان از آن ها استفاده کرد

- \d : معادل [0-9]. مثلا \d\d یعنی یک عدد دو رقمی
- \s : معادل یک فاصله از هر نوع است. [\r\n\t\f\v]
- \w : معادل یک «حرف یا عدد انگلیسی» مثال عبارت \d\w\d معادل یک حرف در بین دو رقم می باشد مانند 1b2, 5G6

۹. نویسه نقطه . : این نویسه می تواند جایگزین هر حرف یا نویسه باشد. مثال عبارت «...» می‌تواند نماینده‌ی "abc" یا "۱۲۳" یا "_a8" باشد. یک مثال دیگر عبارت «a.\d» می‌تواند در یافتن عبارت‌های "ab9" یا "a99" یا "aa7" استفاده شود

www.takbook.com

۱۰. تکرار : برای تعیین تکرار حروف چند علامت وجود دارد. علامت سوال "?"، علامت ستاره "*" و علامت مثبت "+". این سه علامت پس از گروه‌های حرفی قرار می‌گیرند و تعداد تکرار حرف قبل از خود را مشخص می‌کنند.

- ؟ : علامت سوال به معنی «صفر یا یک». مثال عبارت «\d?abc» می‌تواند با رشته‌های "2abc"، "abc"، "0abc" برابر باشد
- * : علامت ستاره به معنی «صفر یا بیشتر» . مثال عبارت a*bc معادل رشته های bc, abc,aabc,aaabc,... می باشد
- + : علامت مثبت به معنی «یک یا بیشتر» هستند. مثال عبارت a+bc معادل رشته های abc,aabc,aaabc, ... می باشد

۱۱. علامت آکولاد {} : از این علامت جهت تعیین دقیق تعداد تکرار استفاده می شود. نویسه های * و + تعداد تکرار نامشخص داشتند اما گاهی اوقات نیاز به تعیین دقیق تعداد تکرار می باشد.تعیین تعداد دقیق تکرار سه حالت دارد

- حالت اول تکرار دقیق یا یک عدد مثلا {۳} یعنی دقیقا سه بار مثال عبارت \d{3} یعنی دقیقا سه رقم مانند 125, 159, 789
- حالت دوم حداقل و حداکثر تکرار با دو رقم مثلا {2,3} یعنی حداقل دو بار و حداکثر سه بار مثال عبارات \d{2,3} یعنی یک عدد دو رقمی یا سه رقمی مانند 12, 68, 685. 458
- حالت سوم تعیین حداقل با عدد و ویرگول مانند {2,} یعنی حداقل دو تکرار مثال \d{2,} یعنی یک عدد خداقل دو رقمی مانند 12, 32, 568, 2568, 125684

۱۲. نویسه فرار escap character : برای یافتن نویسه های خاص (مانند پرانتز، کروشه و ..) مورد استفاده در عبارات منظم در درون باید از نویسه ممیز وارو \ استفاده نمود. مثلا برای یافتن عبارت (abc) باید از عبارات منظم \(abc\) استفاده کرد چرا که پرانتز جزو نویسه‌های خاص می باشد

۱۳. گروه‌های نویسه‌ای Character classes : همانطور که پیشتر معرفی شد با استفاده از نویسه کروشه می‌توان گروهی از نویسه‌ها را مطابق داد. برخی از گروه‌های نویسه‌ای پرکاربرد هستند مانند ارقام 0-9. جهت تسریع و خوانایی در نوشتن عبارات منظم برخی از این گروه‌های پرکاربرد استاندارد شده اند و می توان از آن‌ها استفاده نمود. مثلا برای ارقام 0-9 می توان از [:digit:] استفاده نمود مه معادل [0-9] می باشد. در استانداردهای گروه‌های نویسه‌ای آمده است.

www.takbook.com

عنوان	استاندارد	معادل
حرف یا عدد	[:alnum:]	[A-Za-z0-9]
حرف	[:alpha:]	[A-Za-z]
فاصله و پرش	[:blank:]	[\t]
نویسه کنترلی	[:cntrl:]	[\x00-\x1F\x7F]
رقم	[:digit:]	[۰-۹]
نویسه قابل رویت	[:graph:]	[\x21-\x7E]
حروف کوچک	[:lower:]	[a-z]
نویسه قابل رویت و قابل چاپ	[:print:]	[\x20-\x7E]
نویسه‌های نشانه‌گذاری	[:punct:]	[][!"#$%&'()*+,./:;<=>?@\^_`{|}~-]
هر نوع فاصله	[:space:]	[\t\r\n\v\f]
حروف بزرگ	[:upper:]	[A-Z]
عدد مبنای ۱۶	[:xdigit:]	[A-Fa-f0-9]

جهت تمرین و افزایش تسلط به عبارات منظم پیشنهاد می گردد از سایت https://regex101.com استفاده کنید.

www.takbook.com

۱۳رشته‌ها و متغیرها

رشته ها و متغیرها در فصول قبلی معرفی شده اند اما در این فصل نگاه عمیق تری به این دو عنوان خواهیم داشت

۱۳/۱ رشته ها

رشته ها یکی از پرکاربردترین متغیرها و مقادیر در برنامه های مختلف می باشند که در اغلب مواقع هدف پردازش رشته به صورت پیدا کردن زیر رشته، جایگزینی و ... می باشد. لذا در اغلب زبان های برنامه نویسی امکانات متعددی برای پردازش رشته های در اختیار برنامه نویس قرار می گیرد.

رشته ها را می توان نوعی آرایه از کاراکتر در نظر گرفت مثلا رشته "abcde" آرایه ۵ عضوی از جنس کاراکتر می باشد بنابراین بسیاری از تکنیک ها و عملگرهای مورد استفاده برای آرایه ها، جهت رشته ها نیز قابل استفاده می باشد

در اسکریپت نویسی لینوکس تکنیک ها و دستورات متعددی برای پردازش رشته ها در نظر گرفته شده است که در این بخش به بررسی آن ها می پردازیم.

❖ در دستور expr ایندکس آرایه ها از ۱ شروع می شود اما در متغیرهای اسکریپت از صفر 0 شروع می شود. بنابراین در مثال ها هرگاه دستور expr دیده شود ایندکس اول ۱ خواهد بود اما اگر از فرم کامل متغیر {..:variable:arg1:arg2}$ استفاده شود ایندکس شروع 0 خواهد بود

۱۳/۱/۱ طول رشته

برای محاسبه طول رشته از عملگر **#** به صورت {string#}$ استفاده می شود.

❖ **دقت شود که در اغلب تکنیک‌های مورد استفاده برای پردازش رشته ها از فرم کامل متغیر استفاده می شود**

همچنین اگر از دستور expr استفاده شود طول متغیر با دستور '*.' : "string$" expr یا به صورت expr length $string نیز قابل محاسبه می باشد

```
stringZ=abcABC123ABCabc
echo ${#stringZ}                    # 15
echo `expr length $stringZ`         # 15
echo `expr "$stringZ" : '.*'`       # 15
```

www.takbook.com

۱۳/۱/۲ طول زیر رشته

در موارد قابل توجهی از پردازش رشته ها نیاز است که بررسی شود رشته با زیر رشته خاصی شروع شده است یا خیر. در اسکریپت نویسی می توان با استفاده از عبارات منظم این جستجو را انجام داد. برای این منظور با استفاده از دستور expr به دوشکل می توان عمل کرد

۱. دستور match : expr match “$string” “$substring”
۲. نویسه : : expr “$string” : “$substring”

توجه شود که متغیر $substring یک عبارات منظم است

```
stringZ=abcABC123ABCabc
#          |------|
#          12345678
echo `expr match "$stringZ" 'abc[A-Z]*.2'`    # 8
echo `expr "$stringZ" : 'abc[A-Z]*.2'`          # 8
```

۱۳/۱/۳ ایندکس زیر رشته

از دیگر توابع، تابع محاسبه اول ایندکس اولین تطابق یک نویسه است. مثلا اگر به دنبال زیر یکی از نویسه های رشته abc در شته Bahabc هستیم ایندکس ۲ جواب خواهد بود چرا که نویسه a دومین نویسه رشته می باشد. برای پیدا کردن این ایندکس از فرمان expr و دستور index استفاده می شود

```
stringZ=abcABC123ABCabc
#      123456 ...

echo `expr index "$stringZ" C12`          # 6 C position.

echo `expr index "$stringZ" 1c`          # 3  'c' (in #3 position) matches before '1'.
```

۱۳/۱/۴ استخراج زیر رشته

در برخی موارد نیاز است بخشی از یک رشته به عنوان زیر رشته استخراج گردد مثلا سه نویسه اول یا از نویسه دوم الی آخر و یا چهار نویسه از نویسه سوم. برای این استخراج از عبارت ${string:position:length} استفاده می شود. این عبارت از ایندکس position تعداد length نویسه را برمی گرداند و اگر قسمت length: ذکر نشود ${string:position} از ایندکس position تا انتهای رشته برگردانده می شود

```
stringZ=abcABC123ABCabc
#          0123456789.....
#          0-based indexing.
echo ${stringZ:0}                    # abcABC123ABCabc
echo ${stringZ:1}                    # bcABC123ABCabc
```

www.takbook.com

```
echo ${stringZ:7}                # 23ABCabc
echo ${stringZ:7:3}              # 23A  Three characters of substring.


echo ${stringZ:-4}               # abcABC123ABCabc equal to ${stringZ:0}
echo ${stringZ:(-4)}              # Cabc
echo ${stringZ: -4}              # Cabc
```

همانگونه که در مثال دید می شود می‌توان از ایندکس‌های منفی نیز استفاده کرد که در این صورت به معنای ایندکس از آخر رشته است. البته باید دقت شود در زمان استفاده از ایندکس منفی می بایست یا ایندکس در بین پرانتز ذکر شود یا بین : و ایندکس منفی یک فاصله قرار گیرد؛ در غیر اینصورت ایندکس صفر در نظر گرفته می شود.

۱۳/۱/۵ حذف زیر رشته

برای حذف زیر رشته از رشته از دستور # در نام کامل متغیر می توان استفاده نمود${string#substring} .

چنانچه از دستور **#** استفاده شود کوتاهترین تطابق زیر رشته حذف می شود و چنانچه از دستور ## استفاده شود بلندترین تطابق زیر رشته حذف خواهد شد

```
stringZ=abcABC123ABCabc
#         |----|          shortest
#         |----------|    longest

# Strip out shortest match between 'a' and 'C'.
echo ${stringZ#a*C}      # 123ABCabc

# Strip out longest match between 'a' and 'C'.
echo ${stringZ##a*C}     # abc


# امکان ارسال زیر رشته به صورت پارامتریک
X='a*C'
echo ${stringZ#$X}      # 123ABCabc
echo ${stringZ##$X}     # abc
```

همچنین می‌توان از دستور ٪ یا ٪٪، مشابه **#** و **##**، برای حذف زیر رشته از انتهای رشته استفاده نمود.

```
stringZ=abcABC123ABCabc
#                               ||     shortest
#            |------------------|     longest
```

www.takbook.com

```
# Strip out shortest match between 'b' and 'c', from back of $stringZ.
echo ${stringZ%b*c}     # abcABC123ABCa

# Strip out longest match between 'b' and 'c', from back of $stringZ.
echo ${stringZ%%b*c}    # a
```

۱۳,۱,٦ جایگذاری زیررشته

برای جایگذاری زیررشته یا زیررشته جدید می توان از دستور / در نام کامل متغیر استفاده نمود.

- ${string/substring/replacment} : اولین تطابق رشته substring را با رشته replacment جایگذاری می کند
- ${string//substring/replacment} : تمامی تطابق ها را جایگذاری می کند

```
stringZ=abcABC123ABCabc

# Replaces first match of 'abc' with 'xyz'.
echo ${stringZ/abc/xyz}       # xyzABC123ABCabc

# Replaces all matches of 'abc' with # 'xyz'.
echo ${stringZ//abc/xyz}      # xyzABC123ABCxyz
```

امکان فراخوانی به صورت پارامتریک با استفاده از متغیرها نیز وجود دارد

```
stringZ=abcABC123ABCabc

match=abc
repl=000

echo ${stringZ/$match/$repl}          # 000ABC123ABCabc
echo ${stringZ//$match/$repl}         # 000ABC123ABC000
```

چنانچه یکی از دو عبارت جستجو یا جایگذاری خالی باشد دستور مشابه حذف کردن عمل می کند مثلا دستور ${stringZ/abc} معادل حذف اولین تطابق abc می باشد

www.takbook.com

می توان دستور / را همراه با # یا ٪ استفاده کرد. استفاده از نویسه # به معنی تطابق در ابتدای رشته و نویسه ٪ تطابق در انتهای رشته می باشد

```
stringZ=abcABC123ABCabc

echo ${stringZ/#abc/XYZ}        # XYZABC123ABCabc
echo ${stringZ/%abc/XYZ}        # abcABC123ABCXYZ
echo ${stringZ/#abc/XYZ}        # abcABC123ABCabc  ! No match?
```

۱۳/۲ متغیرها

متغیرها پیشتر معرفی شده اند در اینجا قصد داریم گسترش متغیرها را در فرمت کامل مورد بررسی قرار دهیم.

زمانی که برای ارجاع به متغیر از فرمت کامل استفاده می شود می توان این فرمت را گسترش داد و رفتار متغیر را در زمان فراخوانی دستخوش تغییرات نمود

۱۳/۲/۱ فرمت کامل

همانطور که پیشتر نیز گفته شد می توان در زمان فراخوانی متغیر به جای فرمت ساده $var از فرمت کامل ${var} استفاده نمود. استفاده از فرمت کامل می تواند باعث ابهامزدایی کد نیز شود

```
your_id=${USER}-on-${HOSTNAME}
echo "$your_id"

echo "Old \$PATH = $PATH"
PATH=${PATH}:/opt/bin  # Add /opt/bin to $PATH for duration of script.
echo "New \$PATH = $PATH"
```

در خط اول از دو متغیر محیطی استفاده شد است USER که نام کاربر جاری است و HOSTNAME که نام سیستم جاری است.

www.takbook.com

۱۳/۲/۲ مقدار پیش فرض

ممکن است در زمان فراخوانی متغیر از مقداردهی آن مطمئن نباشید در این موقع می توانید مقدار پیش فرض برای آن در زمان فراخوانی تعریف نمایید. فرمت تعریف مقدار پیش فرض به یکی از دو شکل ${parameter-default} یا ${parameter:-default} می باشد مثلا ${var1:-10}. این دو فرمت عملکرد یکسانی دارند اما استفاده از نویسه : برای زمانی است که متغیر تعریف شده اما مقدار ان null است

```
#!/bin/bash
var1=1
var2=2


# var3 is unset.
echo ${var1-$var2}   # 1
echo ${var3-$var2}   # 2
echo ${var3-20}      # 20
echo ${var3:-20}     # 20

var3=
echo ${var3-20}     # output will be empty
echo ${var3:-20}    # 20  considering null value by using :

username0=
echo "username0 has been declared, but is set to null."
echo "username0 = ${username0-`whoami`}"
# Will not echo.

echo

echo username1 has not been declared.
echo "username1 = ${username1-`whoami`}"
# Will echo.

username2=
echo "username2 has been declared, but is set to null."
echo "username2 = ${username2:-`whoami`}"
# Will echo because of :- rather than just - in condition test.
```

در روش بالا فقط در زمان همان ارجاع چنانچه متغیر فاقد مقدار یا مقدار null باشد از مقدار default استفاده خواهد شد و این بدان معناست که در فراخوانی‌های بعدی دیگر مقدار default برای آن متغیر معتبر نمی باشد.

جهت مقداردهی به متغیر فاقد مقدار می توان از فرمت های ${parameter=default}, ${parameter:=default} در زمان فراخوانی استفاده نمود. تفاوت این فرمت با روش قبل در این است که مقدار default به متغیر منتسب خواهد شد و در فراخوانی های بعدی نیز معتبر می باشد

www.takbook.com

```
echo ${var=abc}   # abc
echo ${var=xyz}   # abc
# $var had already been set to abc, so it did not change.
```

۱۳/۲/۳ مقدار جایگزین

شاید عجیب به نظر بیاد که چرا وقتی که متغیر مقدار دارد بخواهید از آن صرفنظر کنید اما در اسکریپت نویسی چنین انتخابی وجود دارد که دقیقا مخالف بند ۱۳.۲.۲ عمل کنید یعنی مقدار جایگزین در صورتی که متغیر مقدار داشته باشد. در فرمت {parameter+alt_value}$ اگر متغیر مقداردهی شده باشد از مقدار آن صرفنظر شده و از مقدار alt_value استفاده خواهد شد و مشابه بند قبل نیز در فرمت {parameter:+alt_value}$ اگر متغیر مقدار دهی شده باشد یا مقدار آن null باشد از آن صرفنظر شده و مقدار alt_value استفاده می شود

```
#!/bin/bash

a=${param1+xyz}
echo "a = $a"      # a =

param2=
a=${param2+xyz}
echo "a = $a"      # a = xyz

param3=123
a=${param3+xyz}
echo "a = $a"      # a = xyz
echo

a=${param4:+xyz}
echo "a = $a"      # a =

param5=
a=${param5:+xyz}
# Different result from   a=${param5+xyz}
echo "a = $a"      # a =

param6=123
a=${param6:+xyz}
echo "a = $a"      # a = xyz
```

۱۳/۲/۴ پیام خطا

می توان متغیر را به فرمتی نوشت که اگر مقدار نداشت یا مقدار null داشت پیام خطا به خروجی فرستاده و برنامه را پایان دهد. برای این منظور از فرمت {parameter?err_msg}$ باید استفاده کرد. چنانچه متغیر مقداردهی نشده باشد پیام خطا به خروجی ارسال شده و برنامه با مقدار برگشتی ۱ پایان می یابد. همچنین

www.takbook.com

مشابه موارد قبل می توان از فرمت ${parameter:?err_msg} استفاده کرد کهدر صورتی که مقدار متغیر null باشد پیام خطا ارسال شده و برنامه با مقدار ۱ پایان می یابد.

در مثال زیر از این فرمت استفاده شده و پارامترهای ورودی برنامه چک شده است

```
#!/bin/bash
# usage-message.sh

: ${1?"Usage: $0 ARGUMENT"}

echo "These two lines echo only if command-line parameter given."
echo "command-line parameter = \"$1\""
exit 0  # Will exit here only if command-line parameter present.
```

۱۳/۲/۵ فراخوانی غیر مستقیم

در زبان های برنامه نویسی مفهومی به نام اشاره‌گر وجود دارد که در واقع اشاره‌گر مقداری را در خود ذخیره نمی کند بلکه آدرسی از حافظه، که عملا یک متغیر دیگر است، ذخیره می کند. به این ترتیب زمانی که مقدار اشاره‌گر به خروجی ارسال شود در خروجی یک آدرس دیده می شود. و برای دیدن مقدار ذخیره شده در آن آدرس باید از عملگر دیگری استفاده نمود مثلا در زبان C از عملگر *.

در اسکریپت نویسی نیز مفهومی به نام فراخوانی غیر مستقیم وجود دارد. می توان در زمان انتساب مقدار به متغیر از فرمت {*varprefix!}$ یا {@varprefix!}$ استفاده کرد در این فرمت تنها بخشی از ابتدای نام متغیر آن فراخوانی می گردد و برنامه تمامی متغیرهایی که قبل‌تر با این پیشوند تعریف شده اند به متغیر منتسب می کند، دقت کنید متغیر تعریف شده را منتسب می کند نه مقدار آن را. مثال اگر متغیر xyz123="This is variable" تعریف کنید و در خطی به صورت a=${!xyz*} آن را فراخوانی کنید متغیر xyz123 به متغیر a منتسب می شود، دقت کنید متغیر منتسب می شود نه مقدار، شبیه اشاره‌گرها در زبان C. اگر بخواهید مقدار متغیر را در خروجی ببینید باید متغیر را به صورت {a!}$ فراخوانی کنید.

```
# This is a variation on indirect reference, but with a * or @
xyz23="whatever"
xyz24=
a=${!xyz*}        #  Expands to *names* of declared variables

echo "a = $a"      #  a = xyz23 xyz24
a=${!xyz@}         #  Same as above.
echo "a = $a"      #  a = xyz23 xyz24

echo "---"

abc23="something else"
b=${!abc*}
echo "b = $b"     #  b = abc23
```

www.takbook.com

```
c=${!b}       #  Now, the more familiar type of indirect reference.
echo $c       #  something else
```