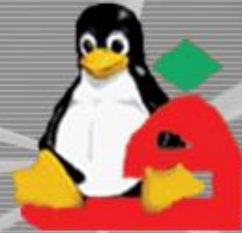




LAIT3C

Learn All Information Technology

Shell Programming



مرداد ۱۳۸۳

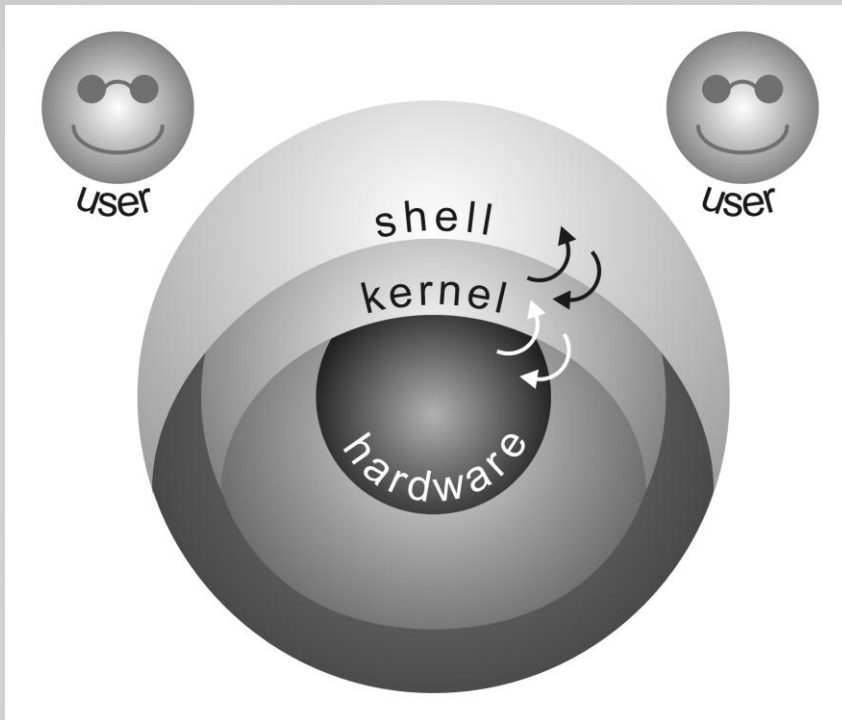
training@farsilinux.org

محمد عزیزی

azizikam@yahoo.com

پوسته

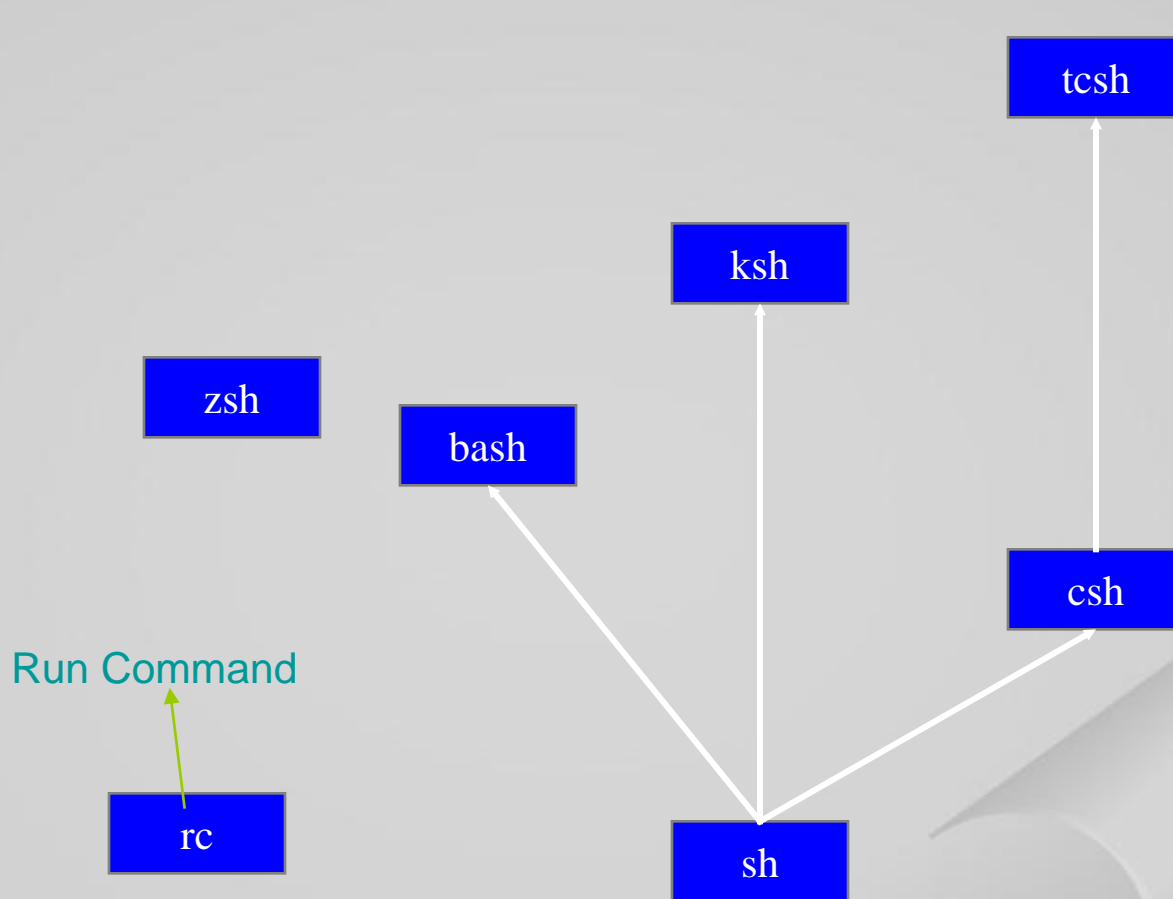
- کاربر با پوسته حرف می زند
- پوسته با هسته
- هسته با سخت افزار
- و سخت افزار انجام دهند کار است.



پوسته (ادامه)

• تاریخچه پوسته

Functionality

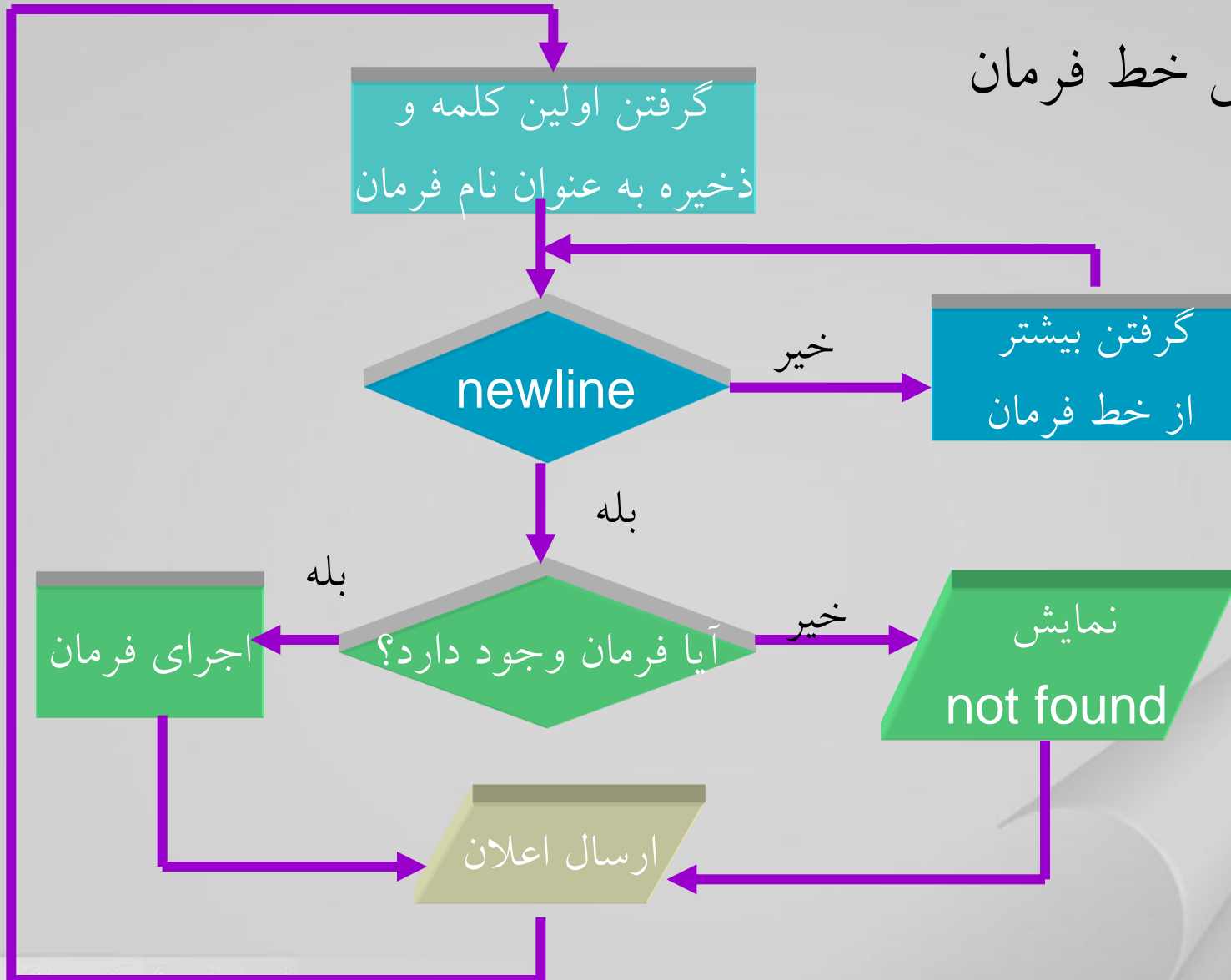


Run Command



پوسته (ادامه)

- پردازش خط فرمان



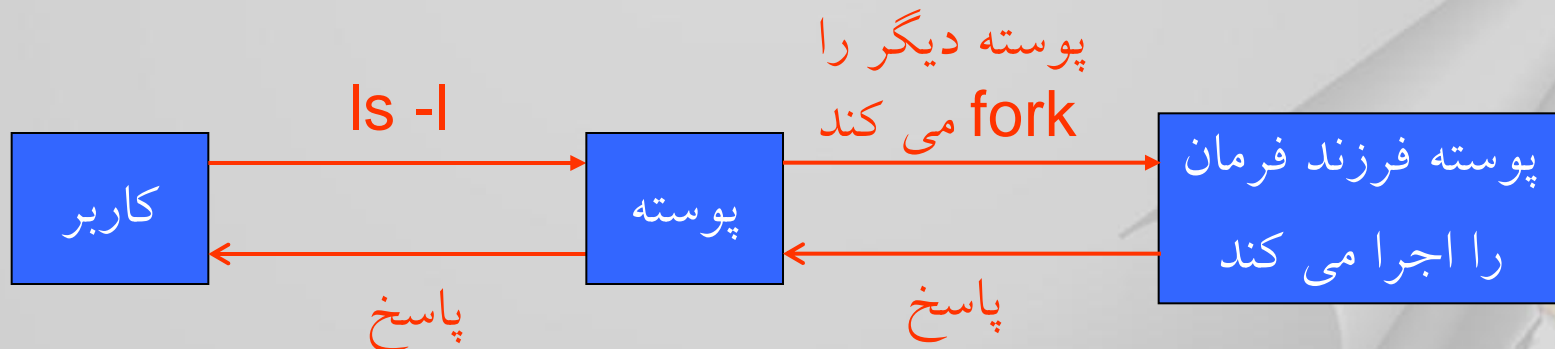
پوسته (ادامه)

پردازش خط فرمان

- پوسته یک فرمان را به سه طریق زیر اجرا می کند:
➤ خودش به درخواست کاربر پاسخ می دهد:



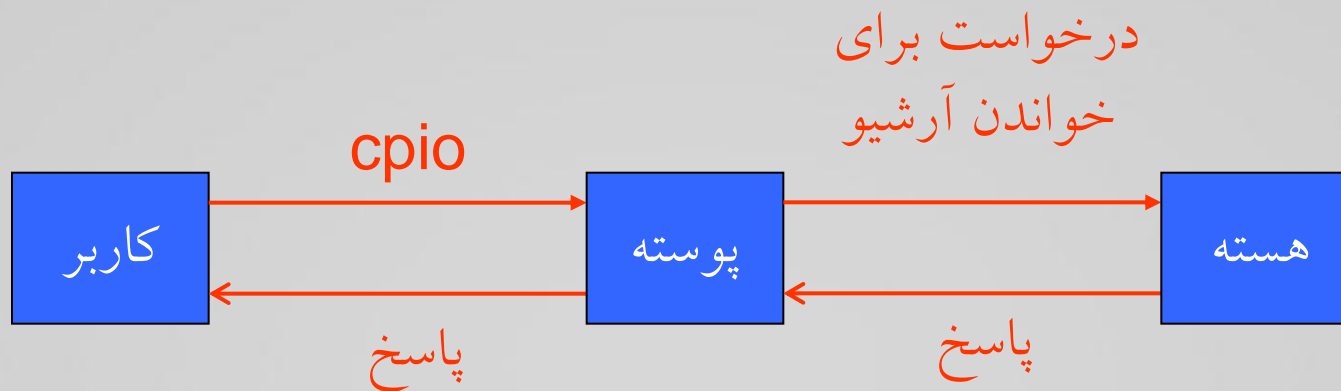
- پوسته فرزند را فراخوانی می کند:



پوسته (ادامه)

پردازش خط فرمان

➤ باید هسته را فراخوانی کند:

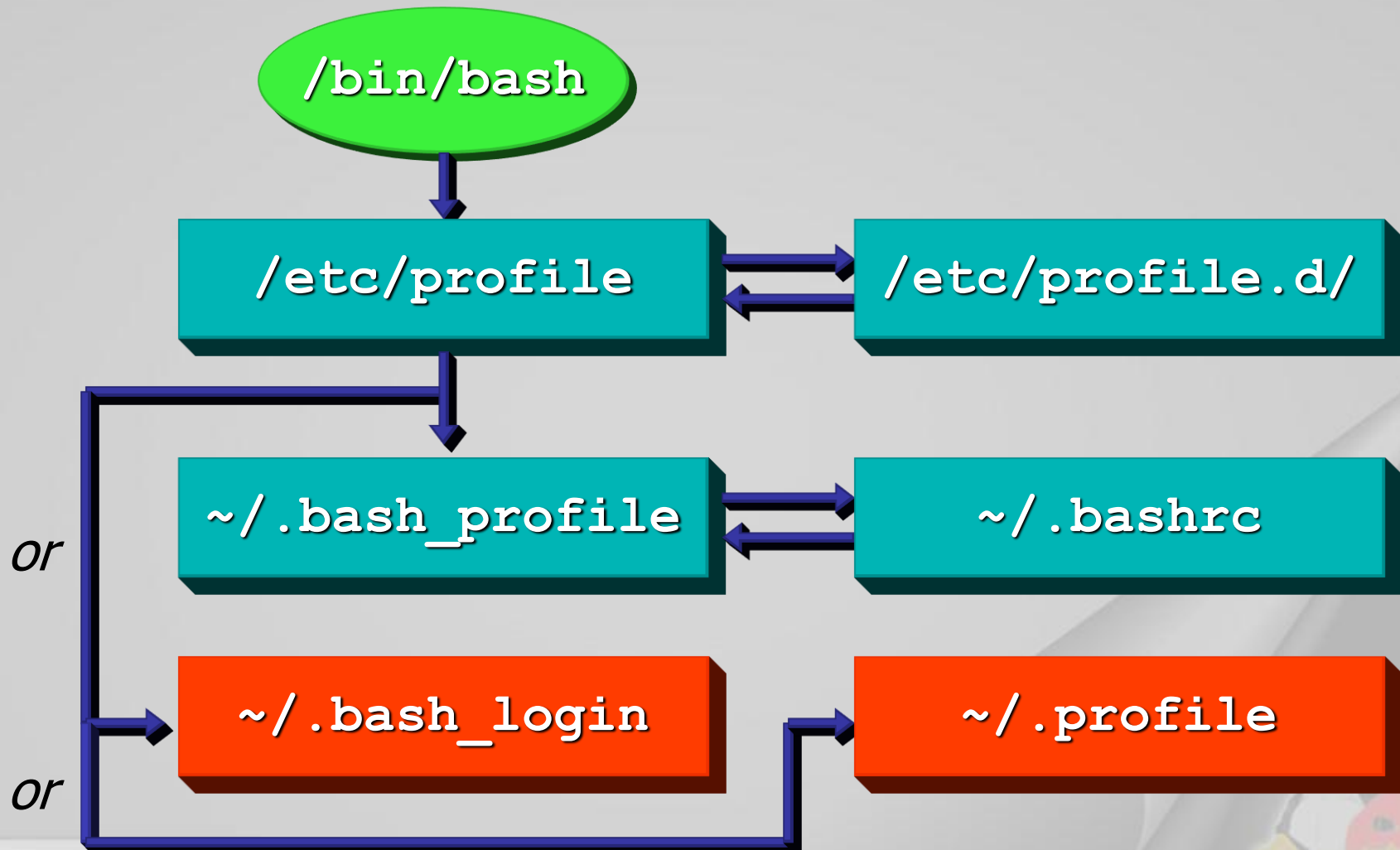


پوسته (ادامه)

- پوسته امکان نوشتن اسکریپت را به کاربر می دهد و می تواند آنرا اجرا کند.
- پوسته خصیصه های توکار زیادی برای کار با فایلها و فرمانها در اختیار قرار می دهد.
- اسکریپت: یک فایل متنی است که حاوی:
 - فرمانهای پوسته.
 - ساختارهای کنترل جریان (مانند if then else و ...).



bash Shell Logins



چرا نوشتن shell script مفید است؟

- فهمیدن (اصلاح کردن) اسکریپت‌های موجود در سیستم.
 - مثلاً اسکریپت‌های startup و shutdown
- جلوگیری از اعمال تکراری
 - اگر یک توالی از فرمانها را مجبورید همیشه اجرا کنید بهتر است آنها را در قالب یک اسکریپت قرار دهید.
- مکانیزه کردن کارهای مشکل
 - بیشتر فرمانهای لینوکس گزینه‌های مشکل زیادی دارند که دوست نداریم هر لحظه وقت زیادی را به یافتن گزینه مورد نظر خود اختصاص دهیم.



نوشتن یک اسکریپت

- ایجاد فایل با استفاده از یک ویراستار

مثلاً `emacs, vim, pico`

- اولین خط فایل معمولاً نام و مسیر پوسته مفسر را مشخص می کند. به عنوان مثال:

نام ← مسیر
#!/bin/bash

اگر `#!/bin/bash` در اسکریپت نیاید
پوسته جاری آن را اجرا می کند

```
$ echo $SHELL
/bin/bash
```

- اضافه کردن فرمانها به فایل.



نوشتن یک اسکریپت (ادامه)

- مثال: یک اسکریپت بنویسید که نام کاربر، زمان و تاریخ کنونی ورود به سیستم و تعداد کاربرانی که هم اکنون وارد سیستم هستند را نمایش دهد.

```
$ cat logininfo
```

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

```
echo -e "The current date and time:\c"
```

```
date
```

حذف کاراکتر **newline** در انتهای خط

گزینه **-e** باعث فعال شدن **\c** می شود.



نوشتن یک اسکریپت (ادامه)

```
echo -e "Number of users login: \c"
```

```
who | wc -l
```

```
echo -e "Personal information:\c"
```

```
whoami
```

```
exit 0
```



اجرای یک اسکریپت

- اجرای اسکریپت در زیرپوسته:

```
root@localhost:~  
File Edit View Terminal Go Help  
$pwd  
/root  
$loginfo  
bash: loginfo: command not found  
$echo $PATH  
/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/usr/X11R6/bin  
:/root/bin  
$./loginfo  
bash: ./loginfo: Permission denied  
$ls -l loginfo  
-rw-r--r--  1 root  root    152 Aug 24 11:35 loginfo  
$
```

شاخه جاری در مسیر نیست

اسکریپت در شاخه
جاری است

اما اسکریپت مجوز اجرایی ندارد

اجرای یک اسکریپت (ادامه)

```
root@localhost:~  
File Edit View Terminal Go Help  
$chmod u+x logininfo  
$ls -l logininfo  
-rwxr--r-- 1 root root 152 Aug 24 11:35 logininfo  
$./logininfo  
The current date and time:Tue Aug 24 12:07:54 IRST 2004  
Number of users login: 2  
Personal information:root  
$
```

حالا مجوز اجرایی
دارد

بالاخره اجرای این
اسکریپت

قرار دادن مجوز اجرایی اسکریپت



اجرای یک اسکریپت (ادامه)

- اجرای اسکریپت در پوسته جاری:

```
$ source loginfo
```

```
$ ↓  
$ loginfo
```

- اجرای صریح اسکریپت بوسیله پوسته مشخص شده:

```
$sh loginfo
```

- توجه: در دو حالت بالا می توان مجوز اجرایی فایل **loginfo** را قرار نداد.



متغیرها

- متغیر در پوسته بدون نوع است و پوسته چیزی که به یک متغیر نسبت داده می شود را رشته ای از کاراکترها فرض می کند.
- پوسته امکان ایجاد، مقداردهی یا حذف متغیر را فراهم می کند.
- نام یک متغیر
 - فقط با **a..z** یا **A..Z** یا **_** می تواند شروع شود.
 - می تواند حاوی **a..z** یا **A..Z** یا **0..9** باشد.
- تعریف متغیر

\$ name=value

در دو طرف = نباید فاصله باشد



ارجاع به متغیر

- برای دسترسی به مقدار یک متغیر باید قبل از نام آن، کاراکتر \$ را قرار داد.

➤ \$ mydir=/home/azizi/script

➤ \$ echo \$mydir

➤ /home/azizi/script

- اگر بخواهیم تغییری را از تغییرات تصادفی محفوظ کنیم باید آن را به صورت **readonly** تعریف کنیم.

➤ \$ count=4

➤ \$ readonly count



ارجاع به متغیر (ادامه)

➤ \$ count=2

➤ bash: count: readonly variable

• مثال: کدام یک از مقداردهی زیر معتبر است؟

• \$ fruit=apple+peach+kiwi ✓

• \$ fruit=apple orange kiwi

• **bash: orange not found** ✗

• برای رفع این مشکل باید بصورت زیر عمل کرد:

• \$ fruit="apple orange kiwi"

• \$ fruit='apple orange kiwi'



متغیرهای آرایه ای

- پوسته روشی را برای گروهی کردن مجموعه ای از متغیرها در اختیار قرار می دهد.
- `name=(value1 value2 ... valuen)`
 - ↑ نام آرایه
 - ↑ `name[0]`
 - ↑ `name[1]`
 - ↑ `name[n-1]`
- دسترسی به عناصر آرایه
- `${name[index]}`
- دسترسی به تمام عناصر آرایه
- `${name[*]}` یا `${name[@]}`



متغیرهای آرایه ای (ادامه)

مثال

```
$ fruit[0]=apple
```

```
$ fruit[1]=orange
```

```
$ fruit[2]=kiwi
```

```
$ echo ${fruit[*]}
```

```
apple orange kiwi
```

```
$ fruit2=( [0]=peach [3]=apple [2]=orange)
```

در مقداردهی یک آرایه به صورت بالا می توان ایندکس را قبل از مقدار نسبت داده شده آورد.

```
$ fruit2[1]=banana
```



حذف کردن یک متغیر

- فرمان توکار **unset** برای حذف یک متغیر بکار می رود.
 - **unset name**
 - یک متغیر **readonly** را نمی توان **unset** کرد.
- مثال:

```
$ unset fruit2[1]
```

```
$ echo ${fruit2[@]}
```

```
peach orange apple
```

```
$tmp=4;readonly tmp
```

```
$unset tmp
```

```
bash: unset: tmp: Cannot unset: readonly var..
```



متغیرهای پوسته

- متغیرهایی هستند که توسط پوسته مقدار دهی می شوند:

➤ متغیرهای محلی

- توسط کاربر تعریف می شوند.
- فقط در پوسته جاری در دسترس می باشند.
- در پرده های فرزند پوسته در دسترس نمی باشند.

➤ متغیرهای محیطی

- در پرده های فرزند پوسته نیز در دسترس می باشند.
- با استفاده از فرمان `printenv` می توان این متغیرها را دید.
- پوسته بیشتر آنها را در هنگام `login` مقداردهی می کند.
- با ویرایش فایل های `.bash_profile` و `.bashrc` دایرکتوری `home` می توان بعضی از آنها را مقداردهی کرد.



متغیرهای پوسته (ادامه)

➤ با استفاده از فرمان **export** می توان یک متغیر را محیطی نمود.

\$ export name

➤ با استفاده از فرمان **env** می توان لیست همه متغیرهای **export** تعریف شده را دید.

\$ env

➤ نام متغیری که جزئی از یک کلمه دیگر می باشد را باید درون { } قرار داد:

\$ directory=/etc/

\$ home=/home/azizi

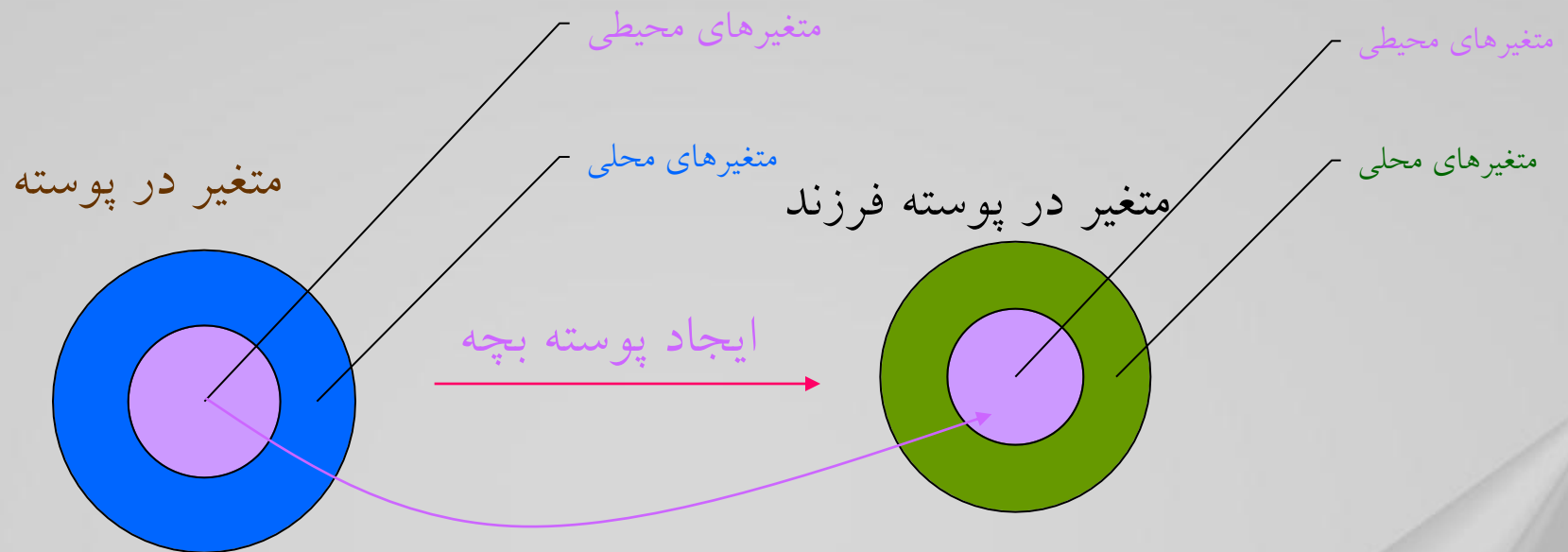
\$ cp \$directorypasswd \$home

غلط

\$ cp \${directory}passwd \$home



متغیرهای پوسته (ادامه)



یک کپی از متغیرهای محیطی در دسترس
پوسته فرزند قرار می گیرد.



متغیرهای پوسته (ادامه)

مثال

```
$ myname=mohammad
```

```
$ echo $myname
```

```
mohammad
```

```
$ bash
```

← ایجاد یک پوسته جدید

```
$ echo $myname
```

← متغیر myname در این پوسته در دسترس نیست.

```
$exit
```

← بازگشت به پوسته پدر



قرار دادن ویژگی و مقدار برای متغیر

- با استفاده از فرمان **declare** می‌توان خواص یک متغیر را محدود کرد.

\$ declare option variable

- **-a** متغیر را آرایه ای اعلان می‌کند.
- **-f** متغیر را نام یک تابع اعلان می‌کند.
- **-i** متغیر را صحیح اعلان می‌کند.
- **-r** متغیر را ثابت اعلان می‌کند.
- **-x** متغیر را جهانی اعلان می‌کند.
- **-p** ویژگیهای متغیر را نمایش می‌دهد.



قرار دادن ویژگی و مقدار برای متغیر (ادامه)

- فرمان **declare** به تنهایی با ویژگی خاصی و بدون نام متغیر به عنوان آرگومان، لیست تمام متغیرهایی را که آن ویژگی را دارند نمایش می دهد:

- \$ declare -rx**

لیست تمام متغیرهای خواندنی
یا جهانی را نمایش می دهد

- می توان ویژگی یک متغیر که **readonly** نیست را حذف نمود:

- \$ declare -x var** → تعریف متغیر به صورت جهانی

:

- \$ declare +x var** → حذف ویژگی جهانی بودن متغیر



مثال

```
$ cat demo_dec
```

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

```
declare -i var1
```

```
var1=12
```

```
var1=var1+1
```

```
var1="Hello"
```

```
echo $var1
```

```
declare -r var2=2
```

```
var2=3
```

متغیر **var1** را صحیح اعلان می کند

نیازی به استفاده از **let** نمی باشد.

نسبت یک رشته به یک متغیر صحیح

var1 مقدار صفر را خواهد داشت

متغیر **var2** را ثابت اعلان می کند

باعث ایجاد پیغام خطا خواهد شد



گرفتن ورودی کاربر

- فرمان **read** یک خط از ورودی استاندارد گرفته و آنرا به متغیرهایی که به عنوان آرگومان به دنبال **read** می آیند نسبت می دهد:

• **read var1 var2 var3 ...**

• عمل:

- یک خط از ورودی استاندارد می خواند.
- اولین کلمه را به **var1**، دومین به **var2**، و ... نسبت می دهد.
- آخرین متغیر حاوی کلمات اضافی خط فرمان خواهد بود.



گرفتن ورودی کاربر

```
$ cat read_2var
```

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

فاصله یا **tab** دو متغیر را از هم متمایز می کند

```
echo "Enter the value of var1 and var2"
```

```
read var1 var2
```

```
echo var1=$var1
```

```
echo var2=$var2
```

```
exit 0
```

```
$ ./read_2var This is a test
```

var1

var2



گرفتن ورودی کاربر (ادامه)

- با استفاده از گزینه های خاص `read` می توان تعدادی کاراکتر که به اینتر ختم نمی شوند را خواند:

- `read -s -nk -p "Message"`

یعنی قبل از خواندن ورودی،
Message را نمایش بده

تعداد کاراکترها را مشخص می کند

کاراکتر ورودی را نمایش نده

```
$ cat read_1char
```

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

```
read -s -n1 -p "Enter one character" ch  
echo "You enter character $ch"
```



گرفتن ورودی کاربر (ادامه)

- با استفاده از گزینه **read -a** می توان متغیر خوانده شده را آرایه ای قرار داد.

```
$ cat read_array_value
```

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

```
echo "Enter your favorite fruits"
```

```
read -a fruits
```

```
echo "You like ${#fruits[@]} fruits"
```

```
echo "such as ${fruits[0]}, ${fruits[1]} , ..."
```

طول آرایه را می دهد



گرفتن ورودی کاربر (ادامه)

- اگر در فرمان `read` نام متغیر به عنوان آرگومان نیاید، ورودی در متغیر `REPLY` ذخیره خواهد شد.

```
$ cat read_without_arg
```

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

```
echo "Enter a value of var1"
```

```
read
```

فرمان `read` آرگومان ندارد ←

```
var1=$REPLY
```

بنابراین ورودی در متغیر `REPLY` ذخیره می شود ←

```
echo "You enter $var"
```



معرفی چند متغیر محیطی پوسته

متغیر	معنی
BASH	مسیر پوسته bash
HOME	مسیر کامل شاخه home
PATH	پوسته جست و جوی برنامه ها را در مسیرهای نسبت داده شده به این متغیر شروع می کند. این مسیرها با ":" از هم جدا می شوند.
PS1	حاوی اعلانی است که بیانگر انتظار پوسته برای ورود یک فرمان می باشد.
PS2	حاوی اعلانی است که بیانگر انتظار پوسته جهت ادامه ورود توسط کاربر می باشد.

معرفی چند متغیر محیطی پوسته (ادامه)

متغیر	معنی
PS3	
PS4	
IFS	تمایز بین کلمات در یک رشته کاراکتر را برای پوسته مشخص می کند.
PWD	حاوی مسیری جاری است.
OLDPWD	حاوی مسیر قبلی است.



معرفی چند متغیر محیطی پوسته (ادامه)

معنی	متغیر
حاوی شناسه کاربر جاری است.	USER
حاوی شناسه عددی کاربر جاری است.	UID
هر بار که یک زیرپوسته دیگر ایجاد شود به آن یک واحد اضافه می شود.	SHLVL
نوع ترمینال کاربر جاری را مشخص می کند.	TERM
مسیر کامل mailbox کاربر جاری را ذخیره می کند.	MAILPATH

معرفی چند متغیر محیطی پوسته

مثال

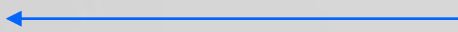
```
$ pwd
```

```
/home/azizi/script
```

```
$ echo $HOME
```

```
/home/azizi
```

```
$ cd
```



چون فرمان **cd** آرگومان ندارد بنابراین
مسیر مقصد، **\$HOME** خواهد بود

```
$ pwd
```

```
/home/azizi
```



معرفی چند متغیر محیطی پوسته

مثال (ادامه)

```
$ echo $PATH
```

```
/usr/local/bin:/usr/bin
```

به معنی شاخه جاری می باشد

```
$ export PATH="$PATH:$HOME/script:."
```

```
$ echo $PATH
```

```
/usr/local/bin:/usr/bin:/home/azizi/script:.
```

```
$ echo $PS1, $PS2
```

```
$ , >
```

```
$ PS2="Secondary prompt: "
```



معرفی چند متغیر محیطی پوسته

مثال (ادامه)

به جای بستن نقل
قول، ایتر زده ایم ←

```
$ echo "This demonstrates  
Secondary prompt: prompt string 2"
```

```
This demonstrates  
prompt string 2
```

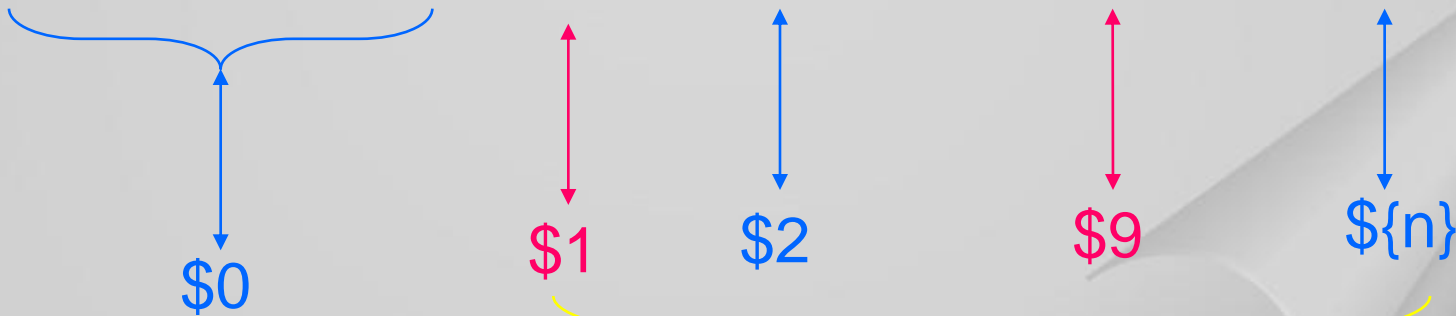
```
$ PS1="→" ; export PS1  
→echo "Prompt changed"  
Prompt changed  
→PS1="$" ; export PS1
```



پارامترهای مکانی

- وقتی که یک اسکریپت را صدا می زنیم نام اسکریپت و آرگومانهای آن پارامترهای مکانی می باشند.
- در داخل اسکریپت می توان به این آرگونها ارجاع کرد.
- روئیکرد ارجاع به موقعیت نسبی آرگونها در خط فرمان وابسته می باشد.

\$ script_name arg1 arg2 arg9 ...argn



پارامترهای مکانی (ادامه)

مثال

```
$ cat display_5args
```

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

```
echo The first five command line
```

```
echo arguments are
```

```
echo $1 $2 $3 $4 $5
```

```
echo Number of all arguments are $#
```

```
$ ./display_5args a b c d e f g h
```

```
The first five command line
```

```
arguments are a b c d e
```

```
Number of all arguments are 8
```



فرمان shift

- این فرمان آرگومانهای خط فرمان را دوباره مقداردهی می کند.

\$0 بدون تغییر و \$1 قبلی از بین می رود. ... , \$2 ← \$3 , \$1 ← \$2

```
$ cat demo_shift
```

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

```
echo "arg1=$1    arg2=$2    arg3=$3"
```

```
shift
```

```
echo "arg1=$1    arg2=$2    arg3=$3"
```

```
shift
```

```
echo "arg1=$1    arg2=$2    arg3=$3"
```

```
shift
```



فرمان shift (ادامه)

\$./demo_shift a b c d

arg1=a	arg2=b	arg3=c
arg1=b	arg2=c	arg3=d
arg1=c	arg2=d	arg3=

- بعد از هر بار **shift** آرگومانهای خط فرمان یک بار به سمت چپ جابه جا می شوند.



فرمان set

- راهی برای مقداردهی مستقیم پارامترهای مکانی وجود ندارد.
- فرمان **set** آرگومانهای خود را غیرمستقیم به پارامترهای مکانی نسبت می دهد.

```
$ cat demo_set
```

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

```
set A B C
```

بعد از اجرا



```
echo $1
```

```
echo $2
```

```
echo $3
```

```
exit
```

```
$/demo_set
```

A

B

C



فرمان set (ادامه)

مثال

```
$ date
```

```
Fri Aug 6 21:05:15 IRST 2004
```

```
$ cat dateset
```

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

```
set `date`
```

```
echo $*
```

```
echo "Argument 1: $1"
```

```
echo "Argument 2: $2"
```

```
echo "Argument 3: $3"
```

بعد از اجرا

Fri	↔	\$1
Aug	↔	\$2
5	↔	\$3
:		:



فرمان set (ادامه)

مثال

```
$ ./dateset
```

```
Fri Aug 6 21:05:15 IRST 2004
```

Argument 1: Fri

Argument 2: Aug

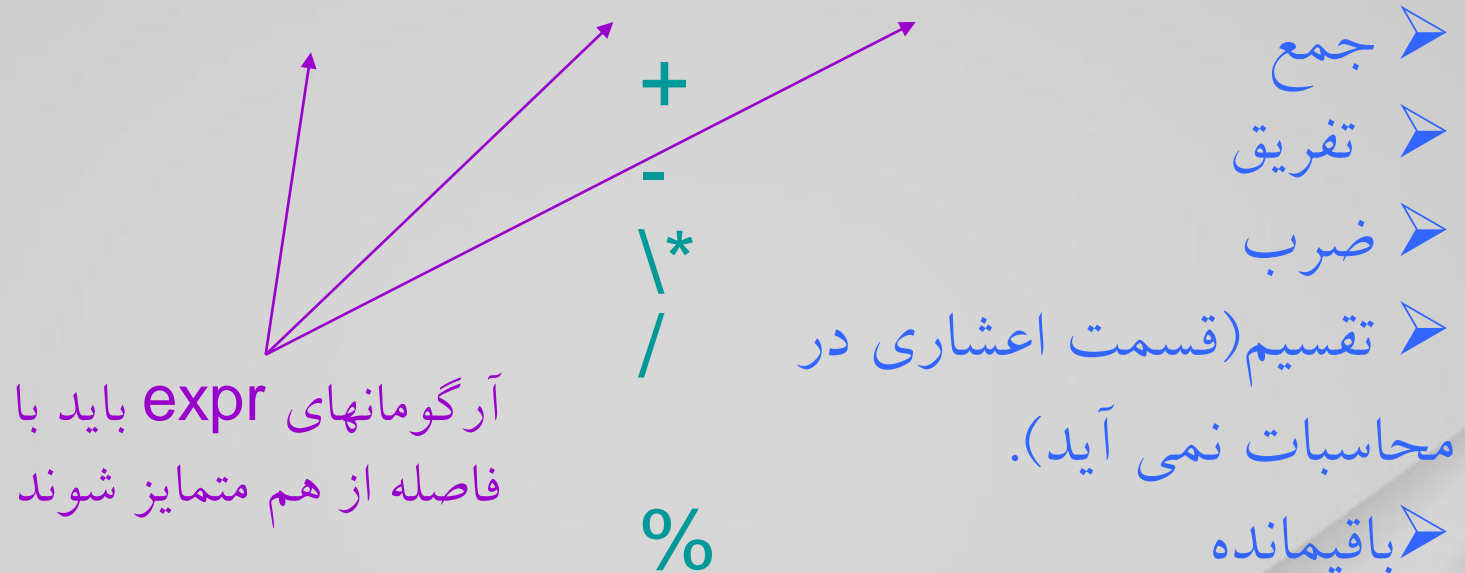
Argument 3: 6

Aug 6, 2004



expr

- برای انجام اعمال صحیح محاسباتی ساده استفاده می شود.
- $\$ \text{expr integer1 operator integer2}$



- **expr** آرگومانهایش را با توجه به **operator** محاسبه و نتیجه را در خروجی استاندارد می نویسد.



expr(cont)

- `expr` فقط محاسبات اعداد صحیح را انجام می دهد.

```
$ expr 10 / 3
```

3

```
$ expr 10 % 3
```

1

```
$ expr 5 * 6
```

`expr: syntax error`

```
$ expr 5 \* 6
```

30



متغیرهای توکار پوسته

• \$?

➤ حاوی وضعیت خروج یک فرمان، یک تابع و یا خود اسکریپت می باشد.

- صفر: در صورت موفقیت آمیز بودن اجرا
- غیر صفر: در صورت ناموفق بودن اجرا.

• \$!

➤ حاوی ID آخرین پردازش اجرا شده در پس زمینه می باشد.

• \$\$

➤ حاوی ID پردازش پوسته جاری می باشد



متغیرهای توکار پوسته (ادامه)

• \$* و @\$

➤ حاوی همه آرگومانهای خط فرمان می باشد:

\$1 \$2 \$3 ...

• "\$@"

➤ حاوی همه آرگومانهای خط فرمان می باشد اما با تفاوت زیر

"\$1" "\$2" "\$3" ...

• "\$*"

➤ یک آرگومان شامل همه آرگومانهای خط فرمان می باشد:

"\$1C\$2C\$3C ..."

C اولین کاراکتر \$IFS می باشد



متغیرهای توکار پوسته (ادامه)

• \$#

➤ حاوی تعداد پارامترهای مکانی خط فرمان می باشد.

• \$-

➤ حاوی ویژگیهای خاص پوسته که فعال هستند می باشد.



متغیرهای توکار پوسته

مثال (ادامه)

```
$ ls es
```

```
es
```

```
$ echo $?
```

```
0
```

← به معنی موفقیت آمیز بودن اجرا است

```
$ ls xxx
```

```
ls: xxx: No such file or directory
```

```
$ echo $?
```

```
1
```

← به معنی ناموفق بودن اجرا است



متغیرهای توکار پوسته

مثال (ادامه)

```
$ sleep 60 &
```

```
[1] 8376
```

```
$echo $!
```

```
8376
```

```
$ echo $$
```

```
3342
```

```
$ cp test $$ .test
```

```
$ ls
```

```
3342.test test
```



متغیرهای توکار پوسته

مثال (ادامه)

```
$ echo $-  
himBH
```

در مورد m

جانشین می‌دهد به متغیرهای تاریخچه در خط فرمان. پسوند را بسیار

\$set -f → غیر فعال کردن بسط مسیر، در پوسته

```
$ echo $-
```

```
fhimBH
```

\$set -f → فعال کردن بسط مسیر، در پوسته



متغیرهای توکار پوسته

مثال (ادامه)

- یک اسکریپت بنویسید که مجوز اجرایی به فایل‌هایی که به عنوان آرگومان به آن ارسال می‌شود بدهد.

```
$ cat set_file_x
```

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

```
chmod u+x "$@"
```

```
exit 0
```

“\$1” “\$2” “\$3”

a, b و c نام سه فایل معتبر

```
$ ./setfilex a b c
```



توالی فرمانها

- توالی ساده
- { توالی ساده }
- توالی شرطی
- توالی پس زمینه
- فرمانهای گروهی



توالی ساده

- با استفاده از **semicolon** می‌توان چند فرمان را در یک خط به صورت ترتیبی اجرا نمود (**x**، **y** و **z** سه فرمان می‌باشند).

`$ x`
`$ y`
`$ z`

`$ x ; y ; z`

فاصله باعث خوانایی بهتر می‌باشد
و بودن آن ضروری نیست

• مثال

`$ echo "List of existing partitions" ; fdisk -l`

↑
نمایش اطلاعات partition table



{ توالی ساده }

باید به semicolon ختم شود

• { command1 ; command2 ; ...; }

باید فاصله باشد

➤ توسط پردازنده پدرا اجرا می شود.

➤ مانند یک تابع بدون نام است.

➤ اگر متغیری درون { ... } تعریف شود، در خارج از آن نیز در دسترس است.

\$ cat test

#!/bin/bash

a=5

{ a=`expr \$a * 2` ; echo "a=\$a"; }

echo "a=\$a"

\$/test

10

10



توالی شرطی

- در توالی شرطی اجرای یک فرمان منوط به نتیجه اجرای فرمان قبلی است.

`$ command1 && command2 ...`

- `command1` اجرا می شود.
- اگر وضعیت خروج آن صفر بود آنگاه `command2` اجرا خواهد شد.

`$ command1 || command2`

- `command1` اجرا می شود.
- اگر وضعیت خروج آن غیر صفر بود آنگاه `command2` اجرا خواهد شد.



توای شرطی

مثال

مثال: یک اسکریپت بنویسید که پیغامی را به شخص خاصی ارسال و در صورت **logout** بودن به آن **mail** کند.

```
$ cat writemail
```

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

```
#usage: writemail user message
```

```
echo "$2" | { write "$1" || \  
                mail -s "warn" "$1"; }
```

```
exit 0
```



توالی پس زمینه

- به صورت پیش فرض پوسته منتظر اتمام اجرای یک فرمان می ماند و بعد اعلان پیش فرض را نمایش می دهد.
- این عمل پیش فرض پوسته را می توان با گذاشتن & در انتهای فرمان تغییر داد:

➤ فرمان مورد نظر در پس زمینه اجرا می شود.

➤ پوسته منتظر اتمام اجرای فرمان نمی ماند و بلافاصله اعلان پیش فرض خط فرمان را نمایش می دهد.

- در مواردی که اجرای یک فرمان وقت زیادی می خواهد بهتر است آنرا در پس زمینه اجرا کرد.



توالی پس زمینه (ادامه)

- می توان اجرای یک پردازش در حال اجرا در پس زمینه را بعد از **logout** شدن و یا قطع ارتباط از سیستم با استفاده از فرمان **nohup** ادامه داد.

\$ nohup ls &

[1] 2740

↑
Hang UP

- خروجی استاندارد این فرمان در فایل **nohup.out** در شاخه جاری ذخیره خواهد شد.

یعنی اولین پردازش اجرا شده در پس زمینه

ID پردازش اجرا شده در پس زمینه



توالی پس زمینه

مثال

- مثال زیر گویای تفاوت اجرای عادی یک فرمان و اجرای آن در پس زمینه می باشد:

\$ sleep 10

بعد از ۱۰ ثانیه اعلان \$ نمایش داده خواهد شد.

\$ sleep 10 &

بلافاصله اعلان \$ نمایش داده خواهد شد.

[2] 2741

\$



فرمانهای گروهی

- (command1 ; command2 ;....)

- فرمانهای بین دو پرانتز در یک زیرپوسته مجزا اجرا می شوند.

➤ جدا کننده فرمانها می تواند "|", "||", "&&", ";" باشد.

- هر تغییری که این پردازش در محیط خود انجام دهد در محیط پردازش پدر تأثیر نمی گذارد.

➤ اگر متغیری بین دو پرانتز تعریف شده باشد در خارج از آن در دسترس نیست.



فرمانهای گروهی

مثال

• مثال:

```
$ cat demo_subshell
```

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

```
outer_var="Outer "
```

```
( echo -e "In the subshell\n"
```

```
inner_var="Inner"
```

```
echo "\"inner_var\" =$inner_var"
```

```
echo "\"outer\" = $outer_var"; )
```

```
echo -e "In the main body of shell \n"
```

متغیر **outer_var** در زیرپوسته
جاری مقداردهی نشده است.



فرمانهای گروهی

مثال (ادامه)

```
echo "\"inner_var\" =$inner_var"
```

```
echo "\"outer\" = $outer_var"
```

```
exit 0
```

متغیر **inner_var** در زیرپوسته

جاری مقداردهی نشده است.



alias

- روشی برای جلوگیری از تکرار فرمانهای طولانی است.
- تعریف alias

می توان از " نیز استفاده کرد.

- `$ alias name='value'`

➤ وقتی یک خط فرمان وارد می شود، پوسته اولین کلمه آن را در لیست alias های خود جست و جو می کند و اگر تطبیقی رخ داد آن کلمه با معادل تعریف شده برای آن، جایگزین می شود.



alias(cont)

➤ اگر **name** نام یکی از فرمانهای لینوکس باشد که دوباره تعریف شده است می توان با گذاشتن \ قبل از آن، از بسط **alias** جلوگیری کرد.

➤ می توان با زدن فرمان **alias** تمام **alias**های تعریف شده را دید.

➤ **alias**ها معمولاً در فایل های زیر تعریف می کنند:

- `~/.bash_profile`
- `~/.bashrc`

➤ برای فعال شدن بسط **alias** توسط پوسته باید ویژگی زیر مربوط به آن را فعال نمود:

`$shopt -s expand_aliases`



alias

مثال

تعریف مجدد فرمان ls به شکل alias \$ alias ls='ls -l | more'

با گذاشتن \ قبل از ls پوسته از بسط alias صرفنظر می کند

\$ \ls

به معنی حذف به صورت بازگشتی

به معنی حذف بدون گرفتن
تأیید از کاربر

\$ alias deltree='rm -R -f '

• می توان در تعریف alias چند فرمان را با semicolon از هم جدا کرد:

\$ alias paper="cd script ; vi count.awk"



alias

مثال (ادامه)

```
$alias windir='mount -t vfat /dev/hda5  
/mnt/win'
```

\$alias دیدن همه alias های تعریف شده ←

```
alias ls='ls -l | more'
```

```
alias deltree='rm -R -f '
```

```
alias paper="cd myhist; vi paper.txt"
```

```
alias windir='mount -t vfat /dev/hda5  
/mnt/win'
```



حذف alias

- با استفاده از دستور `unalias` می توان یک `alias` را حذف کرد:

```
$ unalias alias_name
```

- مثال

```
$ unalias deltree dh
```

```
$ deltree
```

نمایش پیغام خطا

```
-bash: deltree: command not found
```



بسط در پوسته

- فرآیند تبدیل متاکاراکترها و سیمبلهای خاص به معانی دیگر
- انواع بسط با توجه به اولویت در بسط، توسط پوسته

- brace expansion
- tilde expansion
- parameter and variable expansion
- command substitution
- arithmetic expansion
- word splitting
- filename expansion



بسط در پوسته (ادامه)

- اگر پوسته، اولویتی در بسط قائل نشود نتایج مختلفی خواهیم داشت:

\$ ls

tmp1 tmp2 tmp3

\$ var=tmp*

\$ echo \$var $\xrightarrow{\text{ابتدا بسط متغیر}}$ \$ echo tmp* $\xrightarrow{\text{بعد بسط نام فایل}}$ \$ echo tmp*

tmp1 tmp2 tmp3

- اگر اولویت بسط در بالا عوض شود نتیجه چه خواهد بود؟



{ } Brace expansion

- معمولاً برای نمایش لیست نام فایلها استفاده می شود.
- می توان از این روش برای تولید رشته نیز استفاده کرد.

\$ls

در این شاخه هیچ فایل وجود ندارد

preamble
کاما بعنوان جداکننده
postamble

\$ echo chap_{one,two,three}.txt

chap_one.txt chap_two.txt chap_three.txt

➤ برای بسط براکت توسط پوسته حداقل یک کاما داخل براکت لازم است.



{ } Brace expansion(cont)

- در داخل براکت نمی تواند فاصله وجود داشته باشد مگر اینکه آن را quote کرد:

```
$ echo {file1,file2}\ :{\ A," B",' C'}
```

```
file1 : A   file1 : B   file1 : C   file2 : A  
file2 : B   file2 : C
```

- مثال: کپی چهار فایل `main.c`, `f1.c`, `f2.c`, `tmp.c` به شاخه جاری:

```
$ cp /usr/local/src/C/{main,f1,f2,tmp}.c .
```



{ } Brace expansion(cont)

- ایجاد سه دایرکتوری به نامهای `verA`, `verB`, `verC`:

```
$ mkdir ver{A,B,C}
```

- سؤال:

➤ فرمان زیر چه کاری انجام می دهد؟

`$ mkdir ver[A-C]` → اگر فایلی با نام `verA`, `verB`

یا `verC` وجود نداشت

بسط نام فایل توسط پوسته

➤ پوسته `ver[A-C]` را به عنوان آرگومان به `mkdir` ارسال کرده و دایرکتوری به نام `ver[A-C]` در شاخه جاری ایجاد می شود.



{ } Brace expansion(cont)

- بسط های زیر را انجام دهید.

```
$cp filename{ , -old}
```

```
$cp filename filename-old
```

```
$ ls /usr/{ ,local/ }{ ,s}bin/src
```

```
$ ls /usr/bin/src /usr/sbin/src
```

```
/usr/local/bin/src /usr/local/sbin/src
```



~: Tilde expansion

- اگر ~ در ابتدای یک token در خط فرمان بیاید پوسته بسط tilde را انجام می دهد:

پوسته ~ را در ابتدای
یک token می بیند

دیدن کاراکترهای بعدی تا

اولین /

آخر کلمه

کلمه نام login معتبر نباشد

پوسته \$HOME را
جایگزین ~ می کند.

کلمه نام login معتبر باشد

پوسته جانشینی انجام نمی دهد.

پوسته مسیر دایرکتوری home
را به همراه کلمه جایگزین آن
می کند.



~: Tilde expansion

مثال

```
$ echo $HOME
```

```
/home/azizi
```

```
$ echo ~
```

```
/home/azizi
```

```
$ echo ~/letter
```

```
/home/azizi/letter
```

```
$ echo ~root
```

```
/root
```

```
$ echo ~xx
```

```
~xx
```

کلمه xx نام login معتبر نیست
بنابراین پوسته جانشینی انجام نمی دهد.



parameter and variable expansion

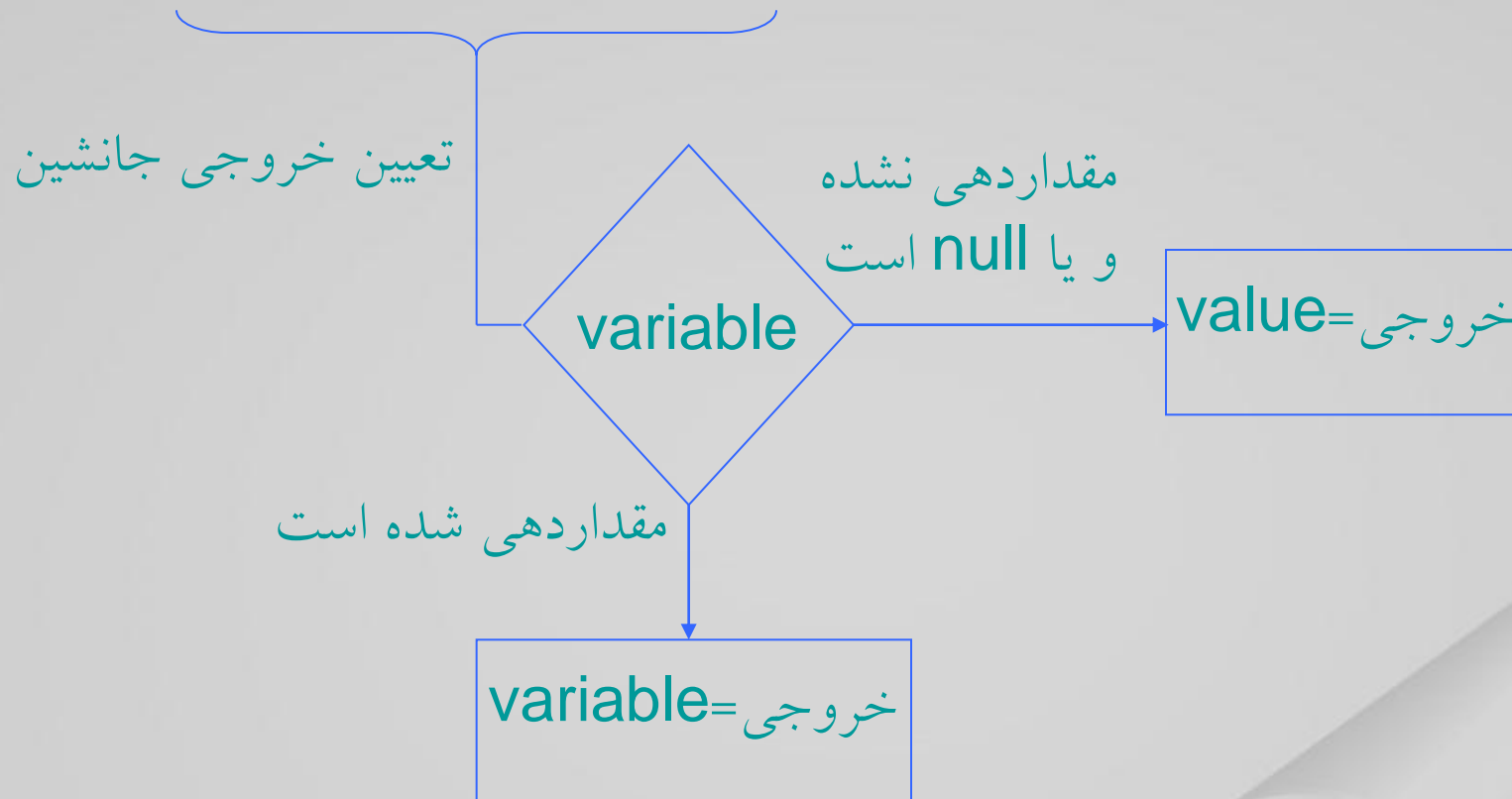
• با توجه به توضیح بسط متغیر در قبل، در اینجا چند ساختار پیچیده بسط متغیر را بررسی می کنیم.

- `${variable:-value}`
- `${variable:+value}`
- `${variable:=value}`
- `${variable:?message}`



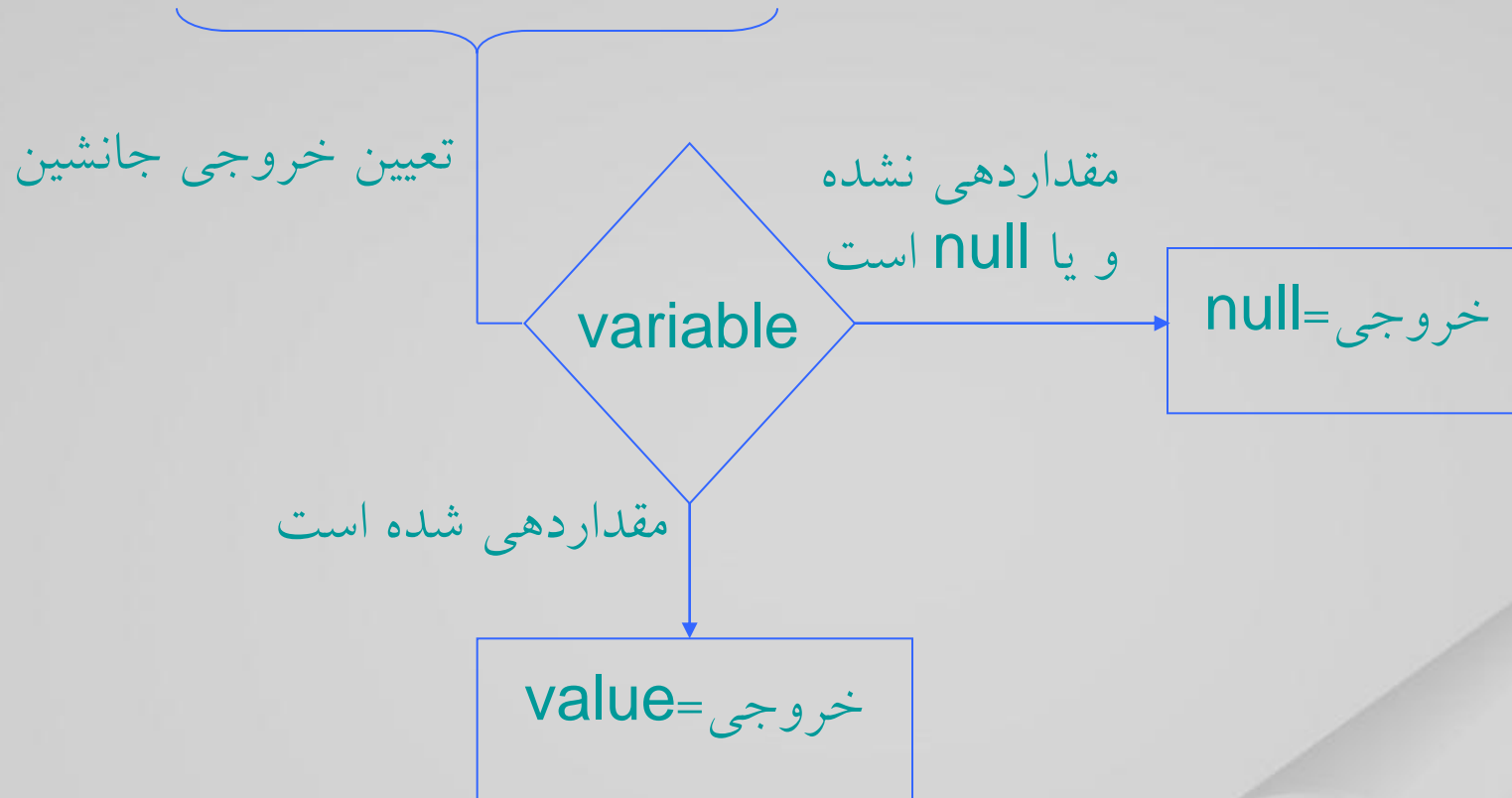
parameter and variable expansion

- `${variable:-value}`



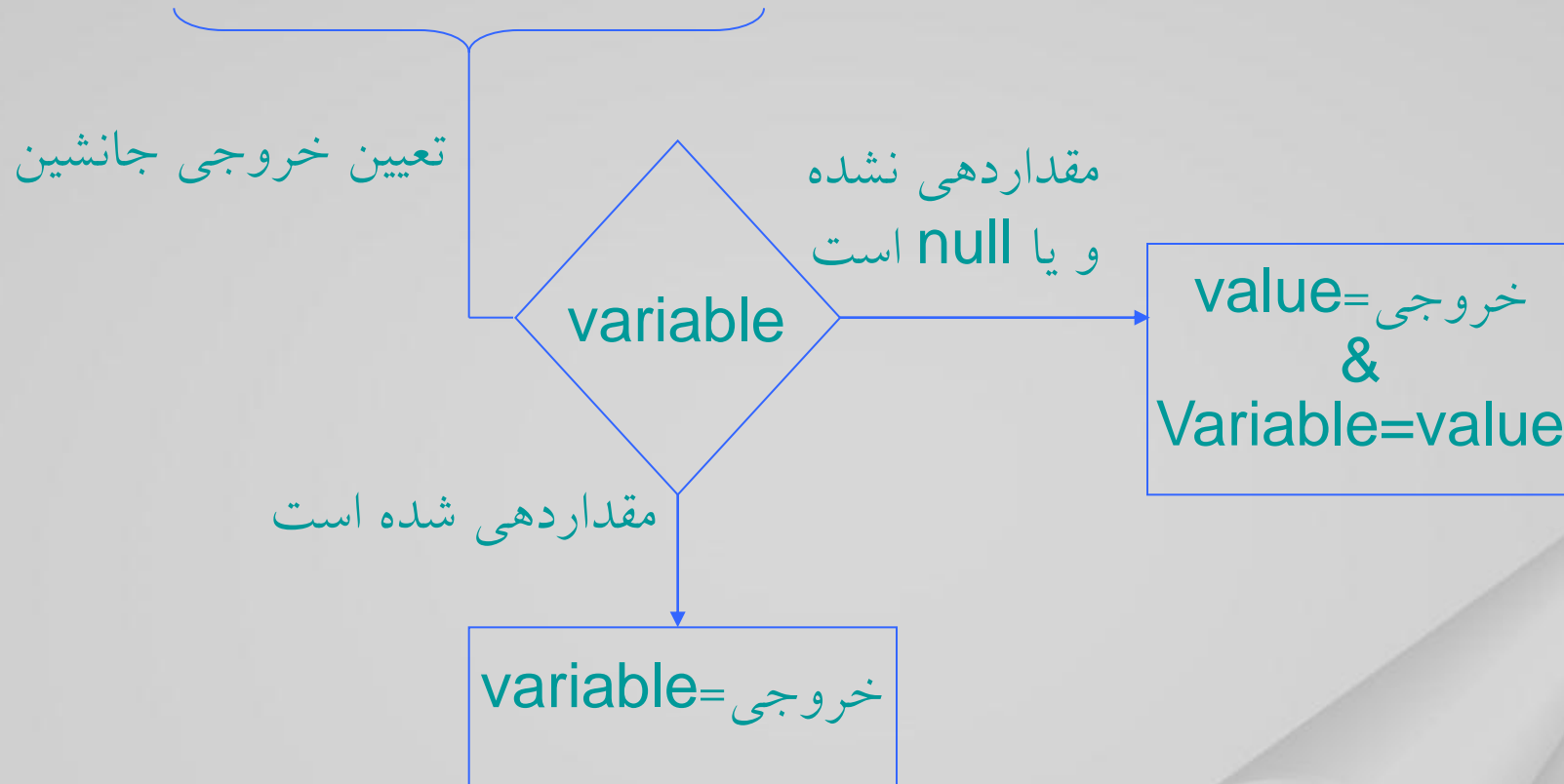
parameter and variable expansion

- `${variable:+value}`



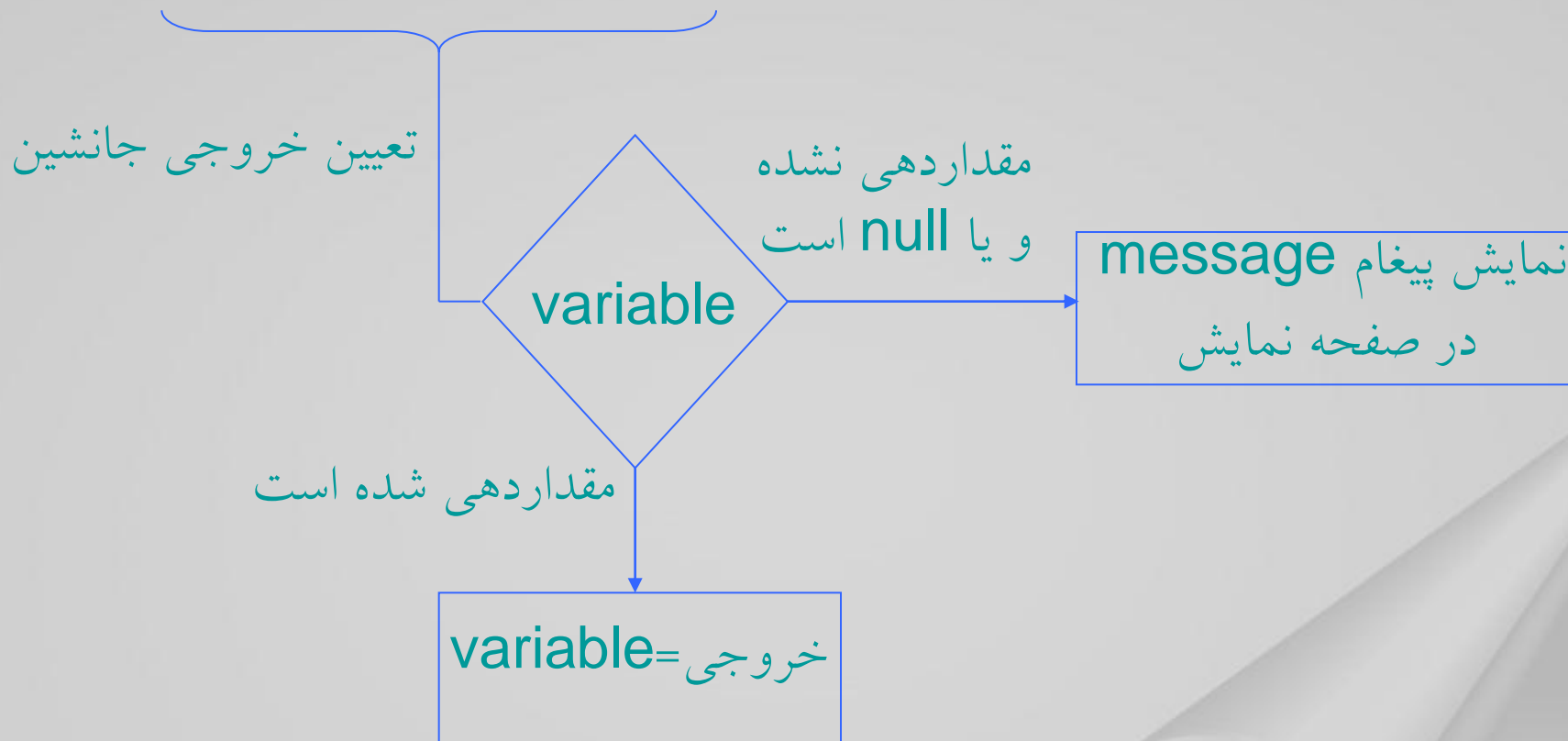
parameter and variable expansion

- `${variable:=value}`



parameter and variable expansion

- `${variable:?value}`



parameter and variable expansion

مثال

```
$ myName=
```

```
$ echo my name is $myName
```

```
my name is
```

```
$ echo my name is ${myName:-"No Name"}
```

```
my name is No Name
```

```
$ echo my name is ${myName:+ "No Name"}
```

```
my name is No Name
```



parameter and variable expansion

مثال (ادامه)

```
$ echo my name is $myName
```

```
my name is No Name
```

```
$ herName=
```

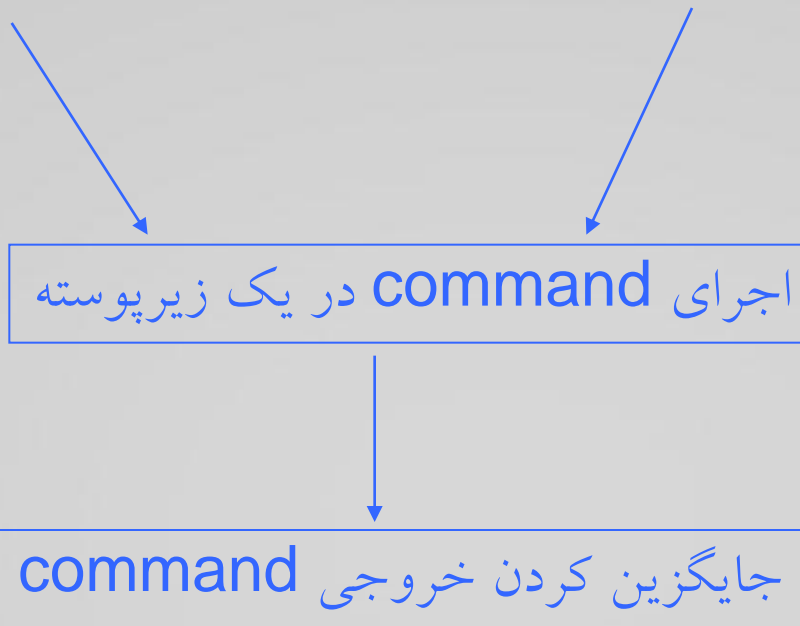
```
$ echo her name is "${herName:?}"She has  
not got a name"
```

```
She has not got a name
```



Command substitution

- ``command`` یا `$(command)`



- `command output`



Command substitution

مثال

```
$ curdir=`pwd`
```

اجرای فرمان `pwd` و نسبت دادن
خروجی آن به متغیر `curdir`

```
$echo There are `who | wc -l` users logged  
on
```

There are 5 users logged on

```
$ UP=`date ; uptime`
```



Arithmetic expansion

- پوسته ابزارهای زیر را برای انجام عملیات محاسباتی در داخل اسکریپت فراهم می کند.
- Using backtick
- Using double parentheses
- Using let



Using backtick

- backtick معمولاً با `expr` استفاده می شود.

```
$ z=4
```

```
$z=`expr $z + 5`
```

```
$ echo $z
```

```
9
```



Using double parentheses

➤ *= /= %=

➤ += -=

بین token ها می تواند

فاصله باشد یا نباشد.

```
$ a=19 ; echo $(( a *= 4 ))
```

76

```
$ echo $((a <= 2))
```

1

```
$ (( a += 1 ))
```

~~#((\$a += 1))~~

```
$ echo "a = $a"
```

a=77



Using double parentheses

مثال

```
$ RES= $(( ( (5+3*2) -4 ) /2 ))
```

```
$ echo $RES
```

3

```
$ var1=4 ; var2=7
```

```
$ var3=$var1+$var2 ; echo $var3
```

4+7

```
$ var3=$((var1+var2)) ; echo $var3
```

11



Using let

- `$ let variable=expression`

با quote کردن، استفاده از فاصله مشکلی نخواهد داشت

- `$ let "variable = expression"`

- متاکاراکترهای مورد استفاده در **expression** شبیه موارد ذکر شده در حالت قبل است.



Using let

مثال

```
$ a=17 ; let a+=4
```

```
$ echo $a
```

21

```
$ let "a >>= 2"
```

```
$ echo $a
```

5

بعد از دوبار شیفت

به سمت راست

عدد ۷ در مبنای ۲ = ۱۰۱۰۱ \longrightarrow a=5



word splitting

- متغیر IFS پوسته حاوی کاراکترهایی است که آرگومانهای خط فرمان را مشخص می کند.
- اگر متغیر IFS به مقداری غیر از پیش فرض خودش مقدار دهی شود باز هم می توان از **space** و **tab** به عنوان متمایز کننده های آرگومانهای خط فرمان استفاده کرد.
- وقتی IFS را به کاراکترهایی مقداردهی کنیم آنها در بسط متغیر متمایز کننده فیلدها خواهند بود.
➤ به این عمل **word splitting** گویند.



word splitting

مثال

\$ a=w:x:y:z

\$ cat \$a

→ در اینجا پوسته بسط متغیر انجام می دهد.

cat: w:x:y:z: No such file or directory

\$ IFS=":"

\$ cat \$a

→ در اینجا پوسته بسط متغیر انجام می دهد و چون IFS=:
است بنابراین به عنوان جدا کننده فیلد در بسط

cat: w: No such file or directory

استفاده می شود.

cat: x: No such file or directory

cat: y: No such file or directory

cat: z: No such file or directory



filename expansion

- متاکاراکتر

➤ کاراکترهایی که معنی خاصی برای پوسته دارند:

* ? [] ' " \ \$; & () | ^ < > newline

space tab

- فرآیند بسط یک رشته حاوی متاکاراکترهای زیر به نام فایل توسط پوسته.

- *

قابل تطبیق با صفر کاراکتر یا بیشتر

- ?

قابل تطبیق با یک کاراکتر

- [...]

قابل تطبیق با هر یک از کاراکترهای داخل کروشه

- [^...]

قابل تطبیق با هر کاراکتری که داخل کروشه نیست



filename expansion

مثال

* با نام تمام فایلها و دایرکتوریهای شاخه جاری جایگزین می شود و سپس **cat** محتوای آنها را نمایش می دهد .

\$ cat *

لیست نام تمام فایلهایی که با **a** شروع، به **bc** خاتمه و بین آنها هر چند کاراکتری می تواند باشد.

\$ ls a*bc

لیست نام تمام فایلهای چهار کاراکتری که با **a** شروع و به **bc** خاتمه می یابد.

\$ ls a?bc

لیست نام تمام فایلهای سه کاراکتری که با **i** یا **c** شروع می شوند.

\$ ls [ic]??

لیست نام تمام فایلهای سه کاراکتری که با **i** یا **c** شروع نمی شوند.

\$ ls [^ic]??



filename expansion

مثال (ادامه)

```
$ ls a\*bc
```

```
ls: a*bc: No such file or directory
```

- لیست نام فایل‌هایی که با نقطه شروع می‌شوند:

```
$ ls .*
```

- مشخص کردن محدوده ای از کاراکترها در نام فایل

```
$ ls [a-z]*
```

لیست نام فایل‌هایی که با حرف کوچک شروع می‌شوند

```
$ ls [a-zA-Z]*
```

لیست نام فایل‌هایی که با حرف شروع می‌شوند



History command

- پوسته فرمانهایی را که کاربر وارد می کند در فایلی به نام `bash_history` در مسیر `$HOME` ذخیره می کند.

↓
`$ echo $HISTFILE`

`/home/azizi/.bash_history`

- تعداد فرمانهایی که در این فایل ذخیره می شوند محدود است. ➤ تعداد این فرمانها در متغیر `HISTSIZE` ذخیره شده است.

↓
`$ echo $HISTSIZE`

`1000`

- با فرمان `history` می توان این تاریخچه را دید.



History command(cont)

فرمان	معنی
!!	اجرای فرمان قبلی
!n	اجرای فرمان شماره n ، history
!string	اجرای آخرین فرمانی که با string شروع می شود
!*	تکرار آرگومانهای فرمان قبل
^old^new	رشته new جایگزین رشته old در فرمان قبل می شود



History command

مثال

```
$ history
```

```
1  ls -l
```

```
2  pwd
```

```
3  jobs
```

```
4  kill %1
```

```
5  cp ~/home/azizi/*.sh ~/home/azizi/script
```

```
6  date +%r
```



History command

مثال (ادامه)

\$!!

اجرای فرمان قبل

date +%r

02:06:37 PM

\$!2

اجرای فرمان دوم history

pwd

/home/azizi/script

\$!cp

اجرای آخرین فرمان شروع شده با رشته cp

cp ~/home/azizi/*.sh ~/home/azizi/script



Quoting

- وقتی پوسته فرمانی را پوشش می کند ابتدا همه متاکاراکترهای موجود در آن را پردازش و بعد فرمان را اجرا می کند.
- **quoting** به معنی غیرفعال کردن معانی خاص متاکاراکترها از نظر پوسته می باشد:

➤ استفاده از backslash :

➤ استفاده از double quote : "..."

➤ استفاده از single quote : '...'



استفاده از \:backslash

• کاراکتر escape:

➤ کاراکتر \ که quote نشده است.

➤ اگر متاکاراکتری بلافاصله بعد از این کاراکتر بیاید معنی خاص خود را برای پوسته نخواهد داشت.

\$echo \

—————→ به معنی ادامه خط در خط بعد

> Hello

بنابراین نمایش اعلان PS2

Hello

\$echo \\\

—————→ برای نمایش کاراکتر \

\



استفاده از backslash \ (ادامه)

حذف تمام فایل‌هایی که در نام آنها کاراکتر فاصله وجود دارد

```
$ rm *\*
```

```
$ cat test01
```

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

```
$. /test01
```

```
ehco \
```

```
echo \
```

```
echo \
```

```
32239$
```

```
echo $$$
```

```
$32239
```

```
echo \$$$
```

```
\?
```

```
echo \?
```



استفاده از double quote: “...”

- تمام متاکاراکترها معنی خاص خود از نظر پوسته را از دست می دهند بجز در موارد زیر:

➤ \$ و ` و \ معنی خاص خود را خواهند داشت.

➤ \ معنی کاراکتر **escape** را خواهد داشت، اگر بلافاصله بعد از آن یکی از کاراکترهای زیر بیاید:

➤ ` \$ “ \ newline



استفاده از double quote :“...”

مثال

```
$echo “Time is `date +%r`”
```

```
Time is 03:47:20 PM
```

```
$echo “Time is `date +%r\`”
```

```
Time is `date +%r`
```

```
$echo “ali  
>reza”
```

newline را escape می کنیم

```
ali
```

```
reza
```

```
$
```

```
$ echo “ali\  
>reza”  
alireza  
$
```



استفاده از double quote : “...”

مثال (ادامه)

```
$ echo Here are some words
```

```
Here are some words
```

فاصله معنی

خاص خود

را خواهد

```
$ echo “Here are some words”
```

حفظ می کند

```
Here are some words
```

backslash معنی خاص \$

برای پوسته را غیر فعال می کند

```
$ echo “\ $PATH=$PATH”
```

```
$PATH=/usr/local/bin:/usr/bin/usr/sbin/..
```



استفاده از double quote : “...”

مثال (ادامه)

```
$var="c d"
```

```
$command a b $var
```

- اجرای command با چهار آرگومان “a” “b” “c” “d”

```
$command “a b $var”
```

- اجرای command با یک آرگومان “a b c d”



استفاده از single quote: '...'

- تمام متاکاراکترها معنی خاص خود از نظر پوسته را از دست می دهند.

باید کاراکتر ' را escape کرد



```
$ echo 'Hello I\'m in here'
```

```
Hello I'm in here
```

\ کاراکتر escape

```
$ echo 'ali\' ←
```

نخواهد بود.

```
> reza'
```

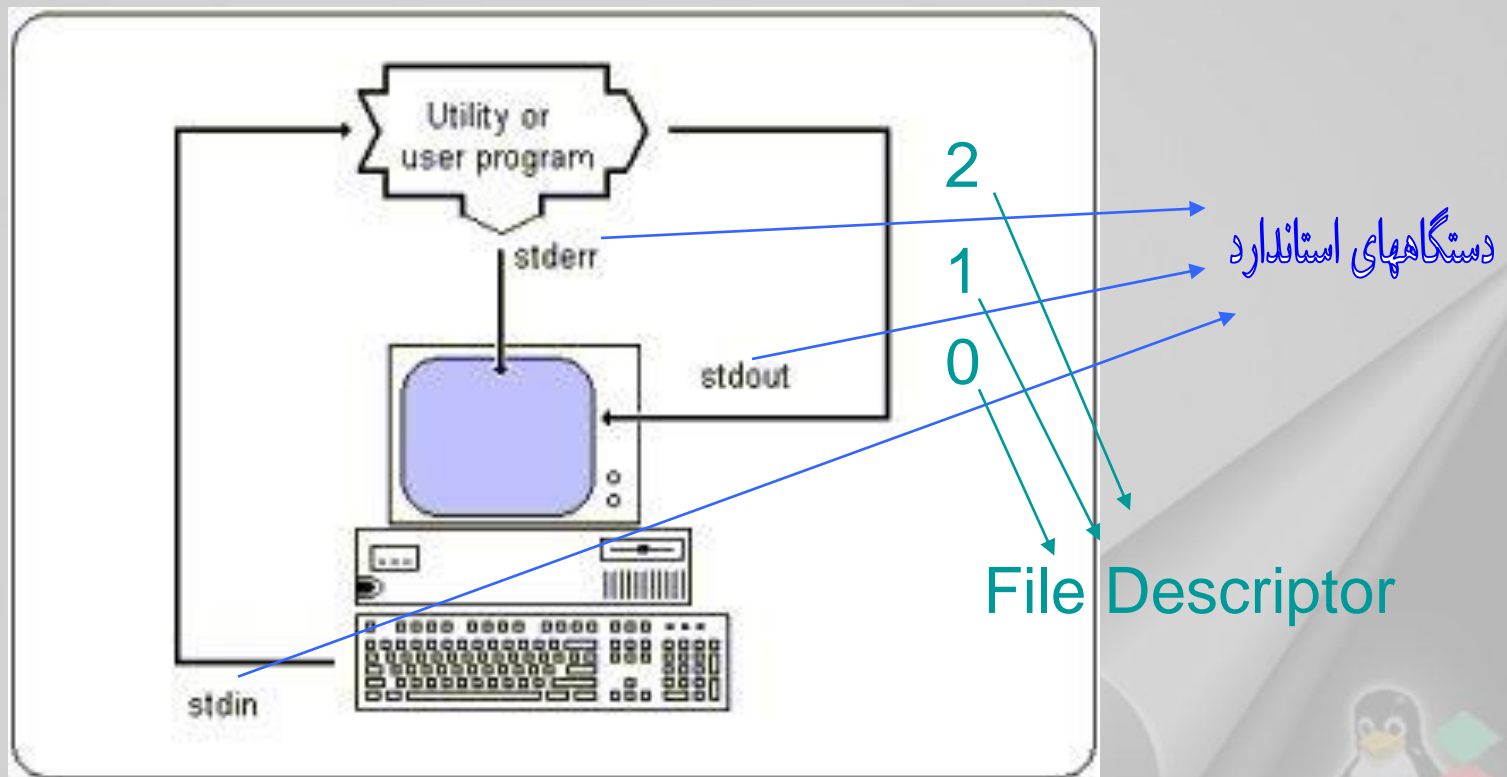
```
ali\'
```

```
reza
```



دستگاه‌های استاندارد

- سه دستگاه استاندارد پشتیبانی شده توسط پوسته که به هر برنامه یا فرمانی که اجرا شود اختصاص می‌یابند عبارتند از:



دستگاههای استاندارد (ادامه)

STandarDINput

- **stdin**: ورودی های مورد نیاز یک فرمان از این دستگاه که به صورت پیش فرض صفحه کلید است خوانده می شود.

STandarDOUtpuT

- **stdout**: خروجی های نرمال یک فرمان به این دستگاه که به صورت پیش فرض صفحه نمایش است ارسال می شود.

STandarDERRor

- **stderr**: خروجی غیرمعمول یک فرمان، مانند پیامهای خطا، به این دستگاه که به صورت پیش فرض صفحه نمایش است ارسال می شود.



redirection

- تغییر جایی که پوسته به صورت پیش فرض ورودی استاندارد (خروجی یا خطای استاندارد) خود را می خواند (می نویسد) **redirection** نام دارد.
- با استفاده از متاکاراکترهای **redirection** در خط فرمان که مقصد نیز بعد از آنها می آید ورودی (خروجی) را می توان **redirect** کرد. این متاکاراکترها عبارتند از:



redirection(cont)

- `>` خروجی استاندارد را `redirect` می‌کند.
- `>&` خروجی استاندارد و خطای استاندارد را `redirect` می‌کند.
- `<` ورودی استاندارد را `redirect` می‌کند.
- `|` خروجی استاندارد یک فرمان را به ورودی استاندارد فرمان دیگر `redirect` می‌کند.
- `>>` به خروجی استاندارد اضافه می‌شود.
- `>>&` به خروجی استاندارد و خطای استاندارد اضافه می‌شود.
- `<<` here documents



output redirection

- قالب عمومی:

\$ command > file



ورودی
استاندارد

command

خروجی استاندارد

file

➤ پوسته فرمان را اجرا کرده و خروجی
آنرا به فایل **file** می فرستد.
➤ اگر فایل موجود نباشد ایجاد شده
و در صورت وجود بازنویسی
خواهد شد



output redirection

- در چه مواقعی بهتر است خروجی را **redirect** کنیم؟
 - ممکن است بخواهیم خروجی را ذخیره کرده و آنرا در فرصتی مناسب بررسی کنیم:
 - اگر برنامه خروجی زیادی تولید کند.
 - اگر برنامه زمان زیادی برای اجرا نیاز داشته باشد.
 - ممکن است بخواهیم خروجی را به شخص دیگری ارائه دهیم.



output redirection example

• مثال:

```
$ cat > sample.txt
```

This text is being entered at the keyboard.

Cat is copying to a file.

Press control-D to indicate the end of file.

\$

Control-D



appending standard output

- قالب عمومی:

`$ command >> file`

➤ اگر فایل `file` موجود نباشد، ایجاد و خروجی `command` به آن اضافه می شود.

➤ اگر فایل `file` موجود باشد، خروجی `command` به آخر آن اضافه می شود.

`$date>myLog`

`$uptime>>myLog`

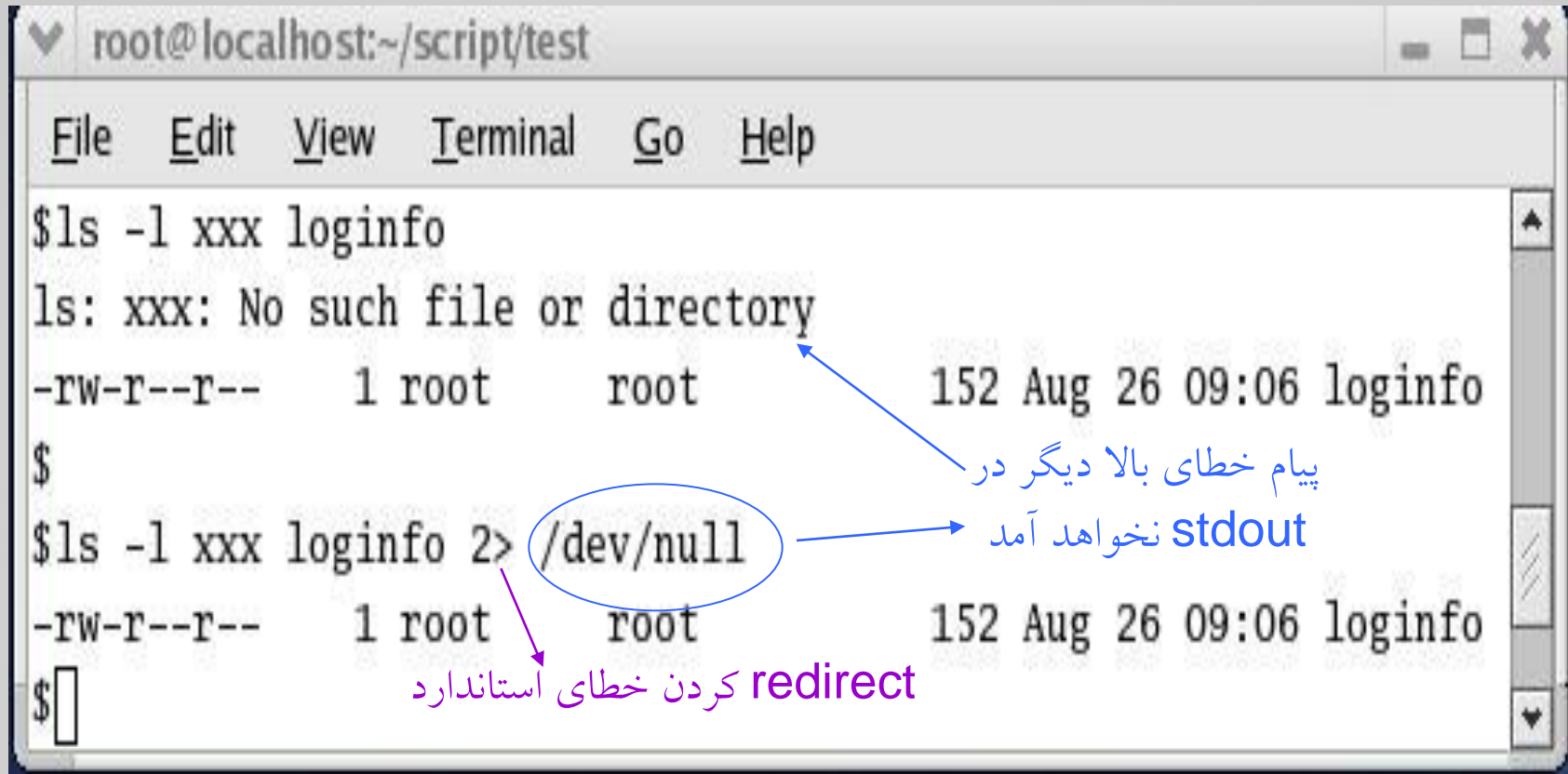
`$who>>myLog`

`{ date ; uptime ; who ; } > myLog`



error redirection

- می توان خروجی های نامعمول یک فرمان را نادیده گرفت:



The image shows a terminal window titled 'root@localhost:~/script/test'. It contains two commands and their outputs. The first command is `$ls -l xxx loginfo`, which results in an error: `ls: xxx: No such file or directory`. The second command is `$ls -l xxx loginfo 2> /dev/null`, which results in the same file listing as the first command. Annotations in Persian explain the redirection: a blue arrow points from the text 'پیام خطای بالا دیگر در stdout نخواهد آمد' (The error message above will not appear in stdout) to the error message of the first command; another blue arrow points from the text 'stdouth نخواهد آمد' (stdout will not appear) to the `2>` redirection operator; and a purple arrow points from the text 'redirect کردن خطای استاندارد' (redirect standard error) to the `/dev/null` destination.

```
root@localhost:~/script/test
File Edit View Terminal Go Help

$ls -l xxx loginfo
ls: xxx: No such file or directory
-rw-r--r--  1 root  root    152 Aug 26 09:06 loginfo
$
$ls -l xxx loginfo 2> /dev/null
-rw-r--r--  1 root  root    152 Aug 26 09:06 loginfo
$
```

پیام خطای بالا دیگر در stdout نخواهد آمد

stdouth نخواهد آمد

redirect کردن خطای استاندارد



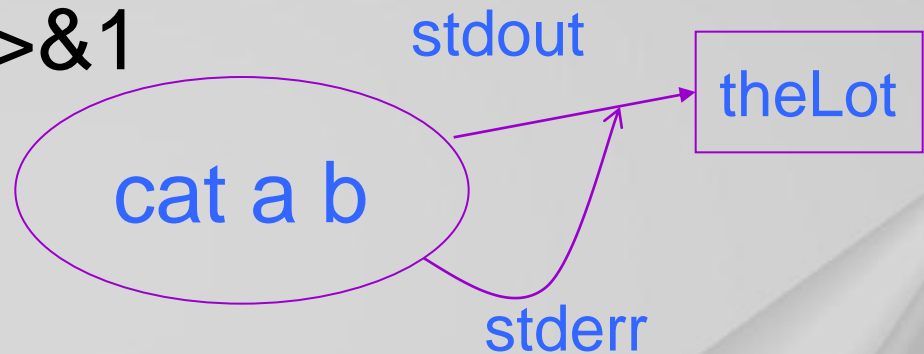
redirect one stream to another

• مثال:

```
$ cat a b 1> theLot 2>&1
```

```
$ cat a b > theLot 2>&1
```

```
$ cat a b >& theLot
```



• نتیجه اجرای هر سه فرمان بالا مشابه هم است.



input redirection

- قالب عمومی:

\$ command < file

➤ محتوای فایل **file** به عنوان ورودی **command** در نظر گرفته می شود.



input redirection(cont)

- در چه مواقعی بهتر است ورودی را **redirect** کرد؟
➤ ممکن است بخواهیم یک برنامه را به صورت مدام و با ورودی های یکسان اجرا کنیم.
- مثال:

```
$ mail -s "ExamAnser" azizi@yahoo.com <\
```

```
> Final_Exam_ans
```

اعلان PS2



input redirection(cont)

- می توان از طریق حلقه `while` خطوط یک فایل را با استفاده از `redirection` ورودی و فرمان `read` خواند:

```
while read LINE
```

```
do
```

```
:
```

```
done < file
```

- مشکل اصلی این حلقه اجرا شدن آن در زیرپوسته می باشد، بنابراین تغییراتی که بر روی محیط اسکریپت مانند متغیرها و ... انجام شود در خارج از حلقه بی تأثیر است.



input redirection example 1

- نمایش خطوطی از فایل `/etc/passwd` با استفاده از `redirection` ورودی، که شامل `root` هستند:

```
while read LINE
```

```
do
```

```
    case "$LINE" in
```

```
        "*root*" ) echo "$LINE"
```

```
        ;;
```

```
    esac
```

```
done < /etc/passwd
```



input redirection example 2

- این اسکریپت سعی در شمردن تعداد خطوط فایل که به عنوان آرگومان به آن ارسال می شود دارد:

```
$ cat count_line
#!/bin/bash
if [ -f $1 ] ; then
    i=0
    while read LINE ; do
        i=`echo "$i + 1" | bc`
    done < "$1"
    echo $i
fi
```



input redirection example 2(cont)

حال اگر فایل **dirs.txt** را به عنوان پارامتر به آن ارسال کنیم:

```
$cat dirs.txt
```

```
/tmp
```

```
/usr/local
```

```
/opt/bin
```

```
/var
```

خروجی زیر را خواهد داشت:

```
0
```

چرا؟



input redirection example 2(cont)

- راه حل: `redirect` کردن ورودی استاندارد قبل از ورود به حلقه و بعد از خاتمه حلقه برگشت به حالت قبل:

```
exec n<&0 <file
```

```
while read LINE
```

```
do
```

```
:      #manipulate file here
```

```
done
```

```
exec 0<&n n<&-
```

- n عددی صحیح و بزرگتر از دو است.



Pipes

- فرض کنید بخواهیم لیستی مرتب از افرادی که هم اکنون وارد سیستم هستند را چاپ کنیم.
- برای این کار باید فرمانهای زیر را صادر کنیم:

```
$ who > temp_file
```

```
$ sort < temp_file > sorted_file
```

```
$ lpr sorted_file
```

```
$rm -f temp_file sorted_file
```

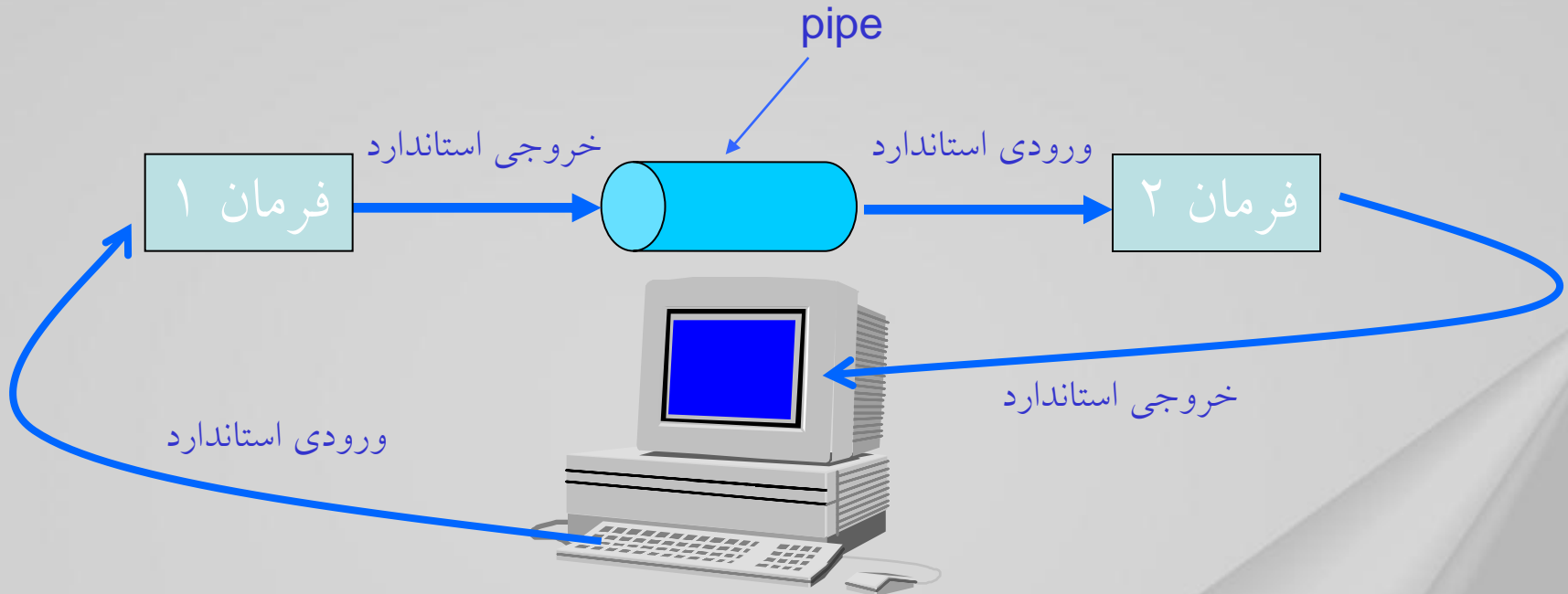
- می توان با استفاده از **pipe** و بدون ساختن فایل های موقتی عمل فوق را انجام داد:

```
$ who | sort | lpr
```



Pipes(cont)

```
$ command1 | command2
```



- پیوسته با استفاده از **pipe** خروجی یک فرمان (برنامه) را نگهداری کرده و آنرا به ورودی یک فرمان (برنامه) دیگر ارسال می کند.



Pipes(cont)

- چون پردازنده های موجود در pipe به صورت همزمان اجرا می شوند و نیازی به نوشتن در فایل های واسط را ندارند، بنابراین افزایش کارایی و سرعت را خواهیم داشت.
- pipe ها یک جهت هستند.
➤ آنها می توانند داده را فقط در یک جهت ارسال کنند.



Pipes example

- `$ phone.txt > tr '[A-Z]' '[a-z]' > testfile`
 - هدف فرمان فوق استفاده از محتوای فایل `phone.txt` به عنوان ورودی استاندارد `tr` است.
 - آیا فرمان فوق درست است؟
 - خیر. چون اولین کلمه خط فرمان باید نام فرمان باشد.
- تصحیح:
- `$cat phone.txt | tr '[A-Z]' '[a-z]' > testfile`



here documents

- قالب عمومی here document:

command << delimiter

document

:

delimiter

- پوسته با دیدن عملگر << آنرا فرمانی فرض می کند که باید از ورودی تا وقتی به delimiter نرسیده است بخواند.
- پوسته وقتی به delimiter رسید، تمام خطوط خوانده شده را به عنوان ورودی به command اعمال می کند.
- delimiter به پوسته می گوید که here documents کامل شده است.



here document example

- اسکریپتی بنویسید که پیغامی را به چندین کاربر که به عنوان آرگومان آن می باشند ارسال کند.

```
$ cat send_mail
```

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

```
for name in "$@"
```

```
do
```

```
    mail -s 'hi there' name << EOF
```

```
    Hi $name, meet me at the water fountain
```

```
EOF
```

```
done
```

```
exit 0
```



here document example

- با استفاده از here document لیستی از شماره تلفن‌ها را به چاپگر ارسال نمایید.

```
$ lpr << MYPHONE
```

```
6006399
```

```
6006370
```

```
6504560
```

```
MYPHONE
```

➤ بدون استفاده از here document عمل فوق چگونه انجام پذیر است؟



redirection example

- معنی هر یک از فرمانهای زیر چیست؟

```
$ cat << finished > input
```

از کاربر تا زمانی که **finished** را وارد نکرده ورودی گرفته و آن را در فایل به نام **input** ذخیره می کند.

```
$ cd /bin > output.file
```

چون فرمان **cd** خروجی تولید نمی کند، فایل تهی به نام **output.file** ایجاد می شود.

```
$ ls | cd
```

چون فرمان **cd** ورودی قبول نمی کند بنابراین نتیجه نمایش پیغام خطا خواهد بود.



redirection example(cont)

```
$ ls chap1.c xx
```

```
chap1.c
```

```
/bin/ls: xx: No such file or directory
```

- با توجه با اطلاعات فوق تفاوت دو فرمان زیر را توضیح دهید؟

```
$ ls chap1.c xx 2>&1 > out.and.err
```

```
ls: xx: No such file or directory
```

```
$ cat out.and.err
```

```
chap1.c
```



redirection example(cont)

```
2) $ ls chap1.c xx > out.and.err 2>&1
```

```
$ cat out.and.err
```

```
ls: xx: No such file or directory
```

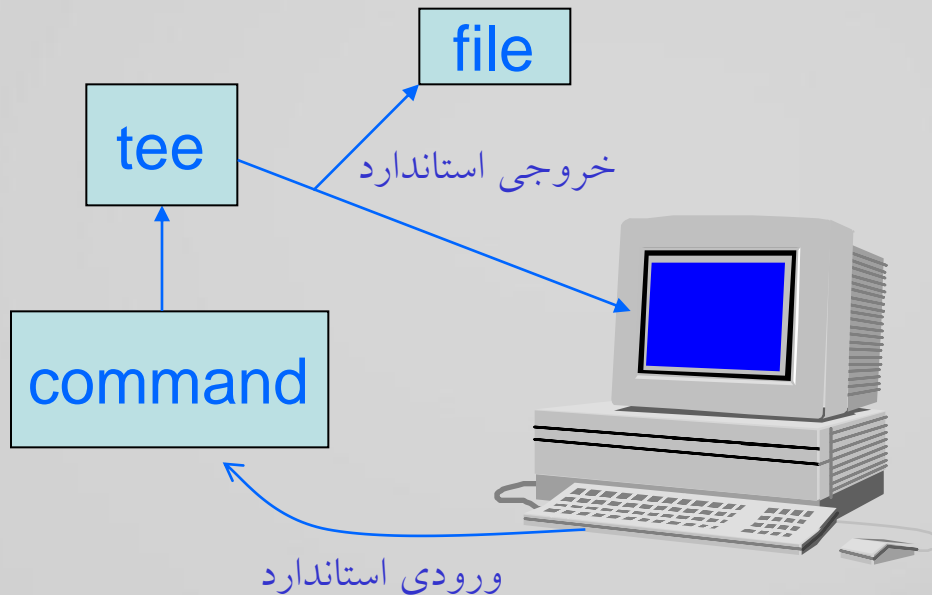
```
chap1.c
```

- توجه: در پاسخ به این سؤال باید توجه داشت که پوسته آرگومانها را از چپ به راست محاسبه می کند.



فرمان tee

- `$ command | tee file`
- ارسال ورودی خود به خروجی استاندارد و فایل .file



فرمان test

- این فرمان یک عبارت را محاسبه کرده و مقدار بازگشتی را

➤ در صورت صحیح بودن، صفر قرار می دهد.

➤ در صورت صحیح نبودن، غیر صفر قرار می دهد.

- Syntax: test expression
or

[expression]

باید فاصله وجود داشته باشد

- در ساختارهای کنترل جریان if, while, until, ... استفاده می شود.



فرمان test (ادامه)

- سه نوع expression زیر در test استفاده می شود:

➤ File tests

- test option file
- [option file]

➤ String comparisons

- test option string
- [option string]

➤ Numerical comparisons

- test int1 operator int2
- [int1 operator int2]



File tests

option description

- -d file True if file exists and is a directory
- -f file True if file exists and is regular file
- -r file True if file exists and is readable
- -s file True if file exists and has nonezero length
- -e file True if file exists.
- -w file True if file exists and is writable
- -x file True if file exists and is executable
- - others are available



File tests

```
$ touch myfile
```

```
$ test -f myfile
```

```
$ echo $?
```

```
0
```

```
$ test -d myfile
```

```
$ echo $?
```

```
1
```



String comparisons

option	description
•-z string	True if string has zero length
•-n string	True if file string has non zero length
•string	True if file string has non zero length
•string1 = string2	True if the strings are equal
•string1 != string2	True if the strings are not equal



String comparisons

```
$ X=abc
```

```
$ [ "$X" = "abc" ]
```

```
$ echo $?
```

```
0
```

```
$ [ "$X" != "abc" ]
```

```
$ echo $?
```

```
1
```



Numerical comparisons

option	description
• <code>int1 -eq int2</code>	True if int1 equals int2
• <code>int1 -neq int2</code>	True if int1 not equals int2
• <code>int1 -lt int2</code>	True if int1 is less than int2
• <code>int1 -le int2</code>	True if int1 is less than or equal int2
• <code>int1 -gt int2</code>	True if int1 is greater than int2
• <code>int1 -ge int2</code>	True if int1 is greater than or equal int2



Other operators

option	description
• <code>! expr</code>	True if <code>expr</code> is false
• <code>expr1 -a expr2</code>	True if both <code>expr1</code> and <code>expr2</code> are true
• <code>expr1 -o expr2</code>	True if either <code>expr1</code> or <code>expr2</code> is true

`expr` یک فرمان `test` معتبر است



Test examples

```
$ x=3
```

```
$ [ x -lt 2 ]
```

```
$ echo $?
```

```
1
```

- مراحل اجرای دستور زیر، به ترتیب، را مشخص کنید؟

```
$ test ! -d $HOME/bin && mkdir $HOME/bin
```



Test examples

```
$ var1=12 ; var2=14
```

```
$[ "$var1" -ne "$var2" ] && echo "$var1 is  
not equal to $var2"
```

12 is not equal to 14

```
$[ "`hello`" ]
```

bash: hello: Command not found

```
$echo $?
```

1



دستورهای کنترلی

- بدون دستورهای کنترلی توالی اجرای فرمانهای یک اسکریپت از یک فرمان به فرمان بعدی می باشد.
- دستورهای کنترلی نحوه جریان اجرای یک اسکریپت را کنترل می کنند.

➤ دستور if

➤ دستور case

➤ دستور while

➤ دستور until

➤ دستور for

➤ دستور select



دستور if

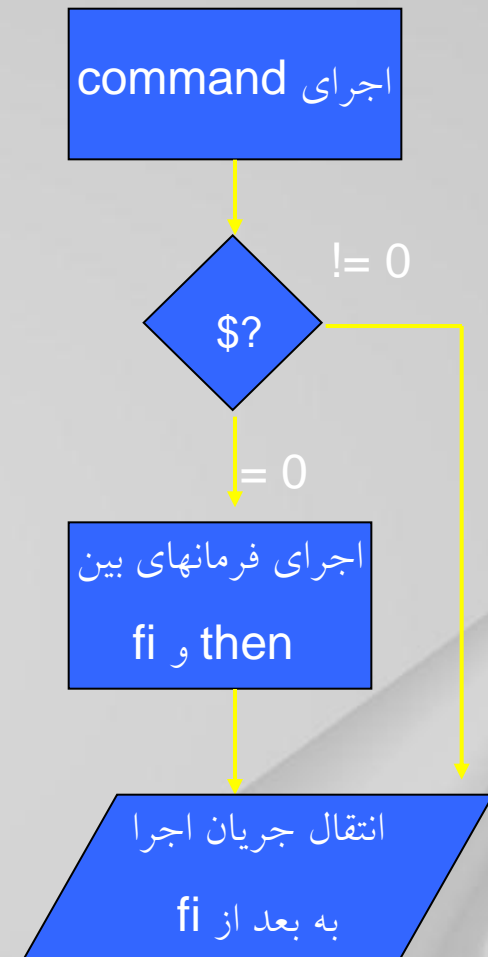
• Syntax

if command
then

command1
command2
...

fi

command می تواند یک توالی از فرمانها که با semicolon از هم متمایز می شوند باشد که در این صورت الگوریتم بالا برای آخرین فرمان این مجموعه صادق است.



مثال

```
$ cat permit2shut  
#!/bin/bash  
if test `who | wc -l` -ge 1  
then  
    echo "Users still logged in. Do not  
shutdown."  
else  
    echo "All clear to shutdown."  
fi  
exit 0
```



مثال

- اسکرپتی بنویسید که عمل آن مشابه `cp -i file1 file2` باشد.

```
$ cat cpi
#!/bin/bash
if [ -f $2 ]
then
    echo "$2 exists. Do you want to overwrite\
it? (y/n)"
    read yn
```



مثال (ادامه)

```
if [ $yn = "N" -o $yn = "n" ]
```

```
then
```

```
    exit 0
```

```
fi
```

```
fi
```

```
cp $1 $2
```

```
exit 0
```



مثال

- یک اسکریپت بنویسید که با استفاده از `/dev/zero` یک `swapfile` بسازد.

```
$ cat create-swap-file  
#!/bin/bash  
ROOT_UID=0  
E_WRONG_USER=65  
FILE=/swap  
BLOCKSIZE=1024  
MINBLOCKS=40
```



دستور if (ادامه)

```
if [ "$UID" -ne "$ROOT_UID" ] ; then
    echo -e "\nYou must be root to run this \
    script.\n"
    exit $E_WRONG_USER
fi

blocks=${1:-$MINBLOCKS }
if [ "$blocks" -lt $MINBLOCKS ] ; then
    blocks=$MINBLOCKS
fi
```



دستور if (ادامه)

```
echo "Creating swap file of size $blocks\  
blocks (KB)."
```

```
dd if=/dev/zero of=$FILE \  
bs=$BLOCKSIZE count=$blocks
```

```
mkswap $FILE $blocks
```

```
swapon $FILE
```

```
echo "Swap file created and activated."
```

```
exit 0
```



صورت دیگر دستور if

• Syntax

if command
then

command1
command2

...

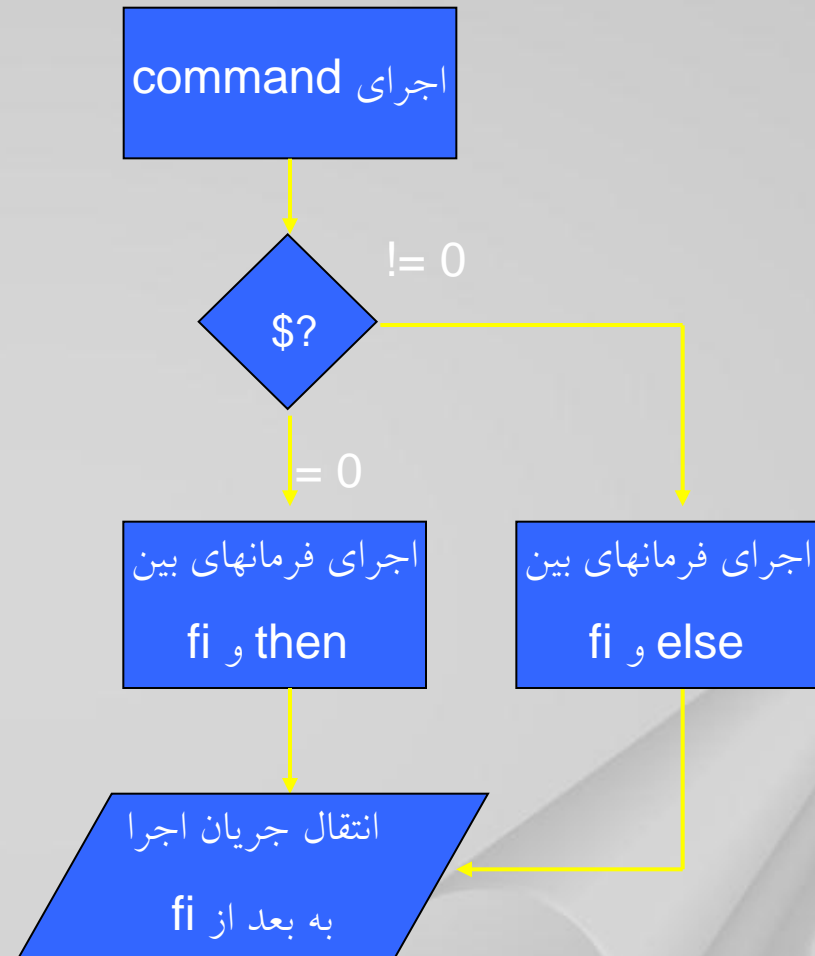
else

command3
command4

....

fi

اختیاری



مثال

- یک اسکریپت بنویسید که از فایل‌های دایرکتوری `home` نسخه پشتیبان تهیه و در مسیر `$HOME/backup_mon` ذخیره کند.

```
$ cat home_back
```

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

```
if [ -e $HOME/backup_mon ]
```

```
then
```

```
    echo "The backup directory exists."
```

```
else
```



مثال (ادامه)

```
echo "A backup directory does not exist,\  
and will be created."
```

```
mkdir $HOME/backup_mon
```

```
echo "A backup directory has been  
created"
```

```
fi
```

```
FILES=$HOME
```

```
DEST= $HOME/backup_mon
```

```
ARCHIVENAME=backup_mon.tgz
```



مثال (ادامه)

```
tar czf $ARCHIVENAME $FILES
cp -ap $ARCHIVENAME $DEST
if [ -e $ARCHIVENAME ]
then
    echo "The backup worked!"
else
    echo "The backup script failed. Time to debug."
fi
exit 0
```



صورت دیگر دستور if

• Syntax

if command1
then

command
command
...

elif command2

command
command
...

elif command3

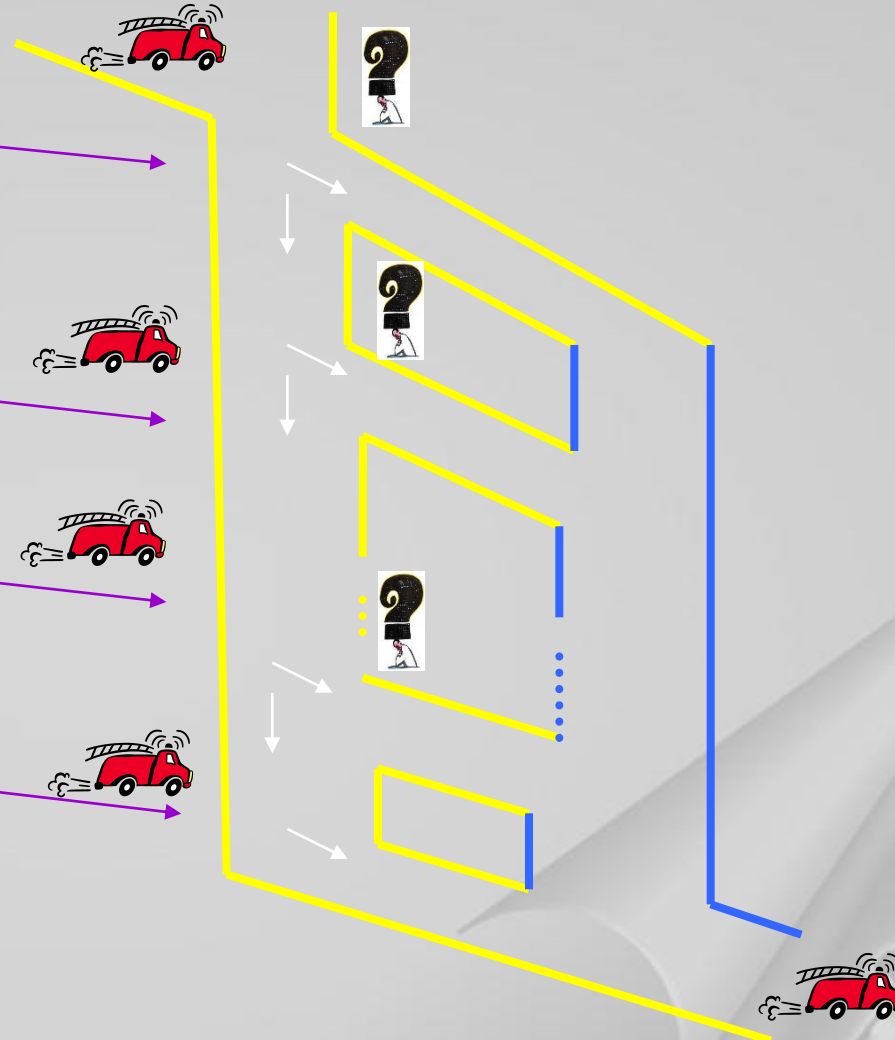
command
command
...

else

command
command
....

fi

اختیاری



مثال

- \$ cat demo_elif

```
#!/bin/bash  
echo -n "word1: "  
read word1  
echo -n "word2: "  
read word2  
echo -n "word3"  
read word3
```



مثال (ادامه)

```
if [ "$word1" = "$word2" -a \  
    "$word2" = "$word3" ]  
then  
    echo "Match: words 1, 2, and 3"  
elif [ "$word1" = "$word2" ]  
then  
    echo "Match: words 1 and 2"  
elif [ "$word1" = "$word3" ]  
    echo "Match: words 1 and 3"
```



مثال (ادامه)

```
elif [ "$word2" = "$word3" ]  
    echo "Match: words 2 and 3"  
else  
    echo "No match"  
fi  
exit 0
```



دستور case

- با استفاده از این دستور می توان یک مقدار را با چند مقدار مقایسه کرد و اگر تطبیقی رخ داد یک یا چند فرمان را اجرا نمود.

- Syntax

```
case value in
  pat1) command
```

```
...
```

```
;;
```

```
pat2) command
```

```
...
```

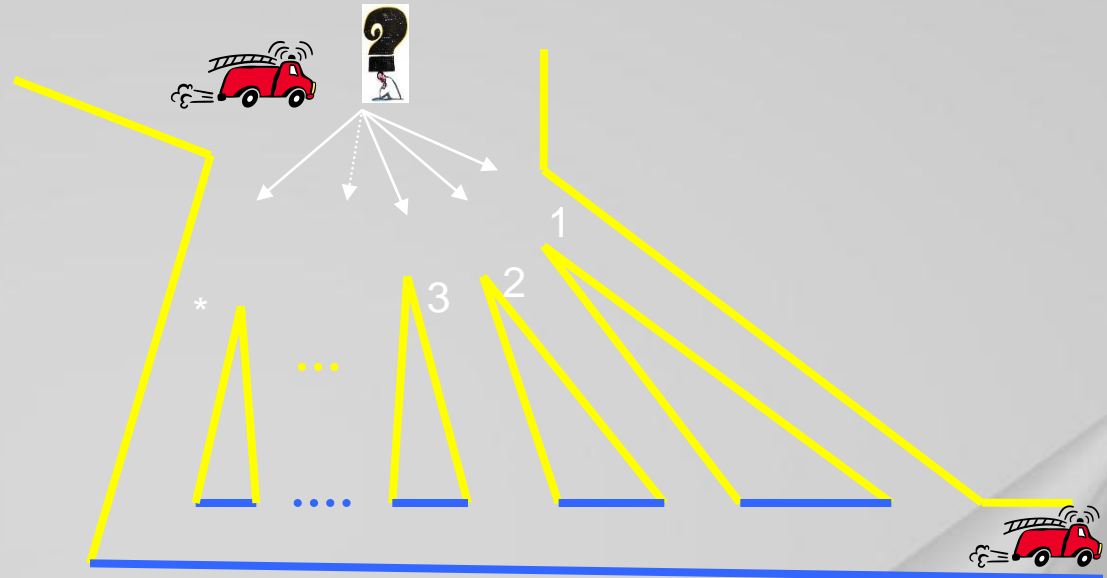
```
;;
```

```
:
```

```
*) command
```

```
;;
```

```
esac
```



مثال

```
$ cat command_menu
```

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

```
cat << end
```

```
COMMAN MENU
```

- a. Current date and time
- b. Users currently logged in
- c. Name of working directory
- d. Contents of working directory

```
Enter a, b, c, or d:
```

```
end
```

به جای استفاده از چند فرمان
echo، از here document استفاده کرده ایم.



مثال (ادامه)

read answer

case "\$answer" in

a) date ;;

b) who ;;

c) pwd ;;

d) ls -C ;;

*) echo "\$answer not legal" ;;

esac



مثال (ادامه)

```
$ ./command_menu
```

COMMAN MENU

- a. Current date and time
- b. Users currently logged in
- c. Name of working directory
- d. Contents of working directory

Enter a, b, c, or d:

a

```
Tue Aug 24 12:35:17 IRST 2004
```

```
$
```



Other case Patterns

- `?` matches a single character
- `[...]` any character in brackets.
Use `–` to specify a range(e.g. `a-z`)
- `|` logical or(e.g. `a|A`)
- `*` it can be used to match:
“any number of characters”



مثال

```
$ cat timeDay
#!/bin/bash
echo "Is it morning? answer yes or no"
read timeofday
case "$timeofday" in
    "yes" | "y" | "Yes" | "YES" )
        echo Good Morning ;;
    [nN]*) echo Good Afternoon ;;
    *)     echo "Wrong answer" ;;
esac
```



دستور while

- Syntax

while command

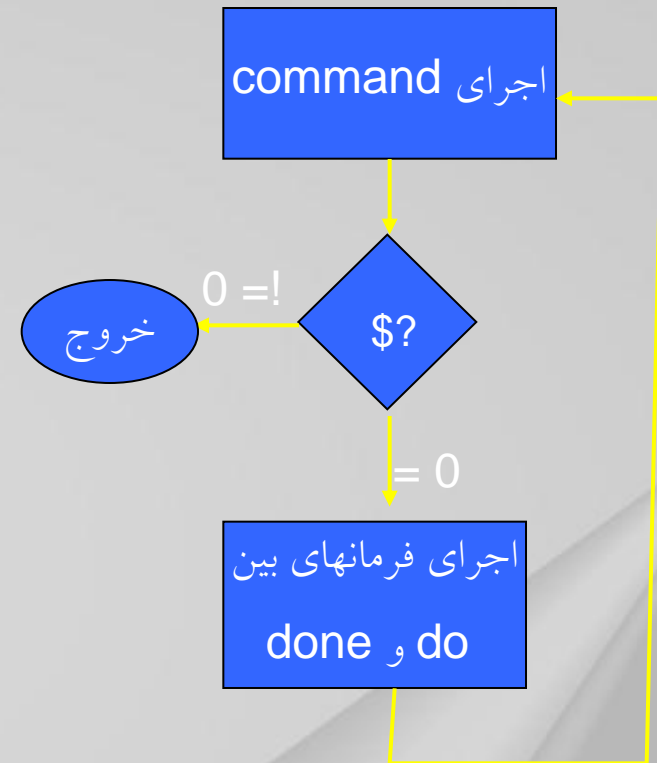
do

command1

command2

...

done



مثال

- یک اسکریپت بنویسید که بصورت متناوب، هر ده ثانیه یک بار، اگر کاری اجرا شد آنرا نمایش دهد.

```
$ cat show-job
```

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

```
result=$(atq)
```

```
# repeat this loop while the result is not a null
```

```
#string
```

```
# atq returns nothing if there are no jobs
```



مثال (ادامه)

```
while [ -n "$result" ]  
do  
    echo -e "\a" # beep  
    echo "Job is running: $result"  
    sleep 10  
    result=$(atq)  
done  
exit 0
```



مثال

- با استفاده از حلقه **while** یک اسکریپت بنویسید که نام یک دایرکتوری دلخواه و معتبر را از ورودی بخواند.

```
$ cat read-dir
```

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

```
RESPONSE=
```

```
while [ -z "$RESPONSE" ]
```

```
do
```

```
    echo "Enter the name of a directory where \  
        your files are located"
```

```
    read RESPONSE
```



مثال (ادامه)

```
if [ ! -d "$RESPONSE" ]  
then  
    echo "Error: Invalid directory name"  
    RESPONSE=  
fi  
done  
exit 0
```



مثال

- اسکرپتی بنویسید که هر نیم ساعت یک بار آخرین Log های سیستم را به email administrator بزند.

```
root@localhost:~/script
File Edit View Terminal Go Help
$cat send-log
#!/bin/bash
PRIOD=1800
currentLine=`cat /var/log/messages | wc -l`
while true
do
    echo "Press <CTRL+C> to end"
    sleep $PRIOD
    newLine=`cat /var/log/messages | wc -l`
    difLine=$((currentLine-newLine))
    dif=`cat /var/log/messages | tail ${difLine}`
    echo "$dif" | mail -s "UpdateLog" root
    currentLine="$newLine"
done
$
```

چرا لازم است؟



دستور until

- Syntax

until command

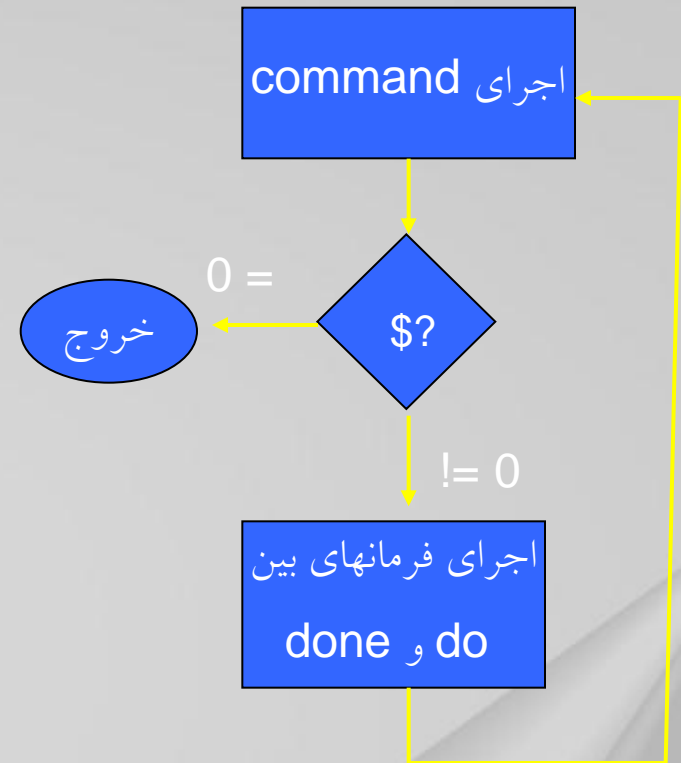
do

command1

command2

....

done



مثال

- با استفاده از حلقه **until** یک اسکریپت بنویسید که نام یک دایرکتوری دلخواه و معتبر را از ورودی بخواند.

```
$ cat read-dir
```

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

```
RESPONSE=
```

```
until [ ! -z "$RESPONSE" ]
```

```
do
```

```
    echo "Enter the name of a directory where \  
        your files are located "
```

```
    read RESPONSE
```



دستور until (ادامه)

```
if [ ! -d "$RESPONSE" ]  
then  
    echo "Error: Invalid directory name"  
    RESPONSE=  
fi  
done  
exit 0
```



دستور for

- Syntax

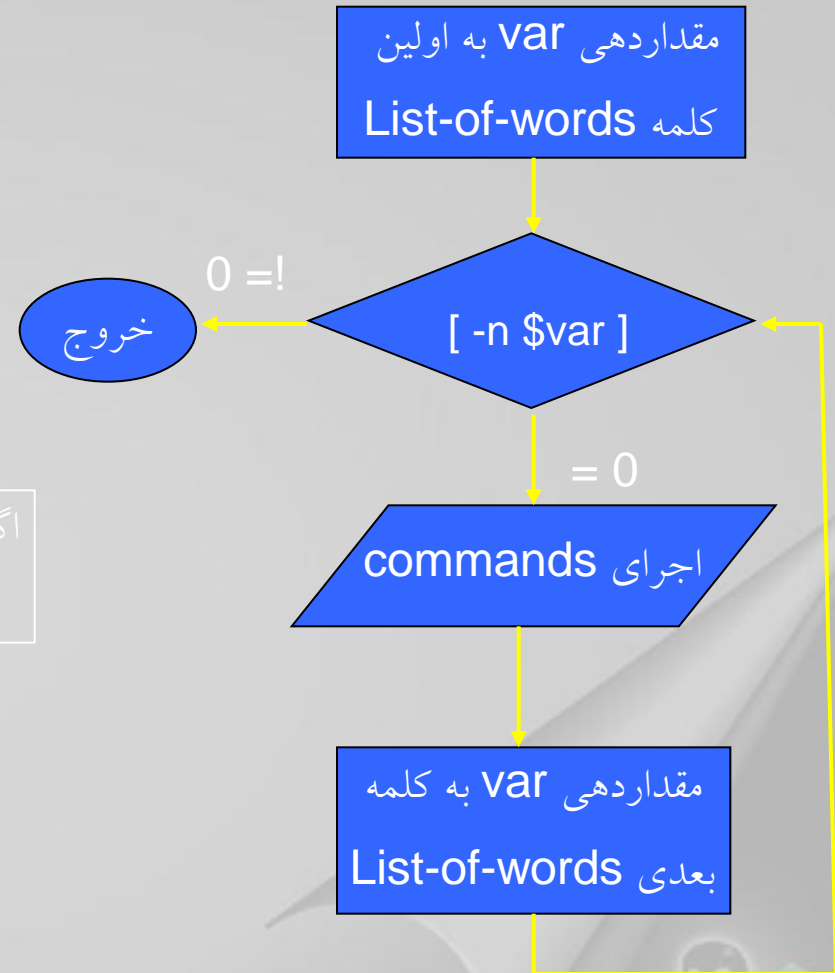
for var in List-of-words

do

Commands

done

اگر نباشد، var به "\$@"
مقداردهی خواهد شد.



مثال ۱

```
for X in *  
do  
    if [ -d $X ]  
    then  
        ls $X  
    else  
        cat $X  
    fi  
done
```

فایلها و دایرکتوریهای شاخه
جاری جانشین آن می شوند



مثال ۲

- مثال: یک اسکریپت بنویسید که فاصله موجود در نام کلیه فایلها و دایرکتوریهای شاخه جاری را، در صورت وجود، به _ تبدیل کند.

```
$ cat blank-rename
```

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

```
for filename in *
```

```
do
```

```
echo "$filename" | grep -q " "
```

```
if [ $? -eq 0 ]
```

چرا لازم است؟

چیزی در خروجی نمایش نده



دستور for (ادامه)

then

```
newname=`echo "$filename" | sed -e 's/ /_/g'`
```

```
if [ -e $newname ]
```

```
then
```

```
mv "$filename" $newname.$$
```

ID پوسته جاری

```
else
```

```
mv "$filename" $newname
```

```
fi
```

```
fi
```

```
done
```

```
exit 0
```



کنترل کننده های حلقه

- برای توقف حلقه و یا پرش به تکرار بعدی.

break ➤

- باعث خروج از حلقه جاری می شود.

continue ➤

- باعث شروع تکرار بعدی می شود.
- معمولاً در حالتی که بنا به هر دلیلی دوست ندارید مقدار کنونی متغیر حلقه را پردازش کنید اما دوست دارید تکرارهای بعدی را امتحان کنید استفاده می شود.



مثال ۳

- تغییر فایل‌های با پسوند `tex` به `latex`

```
$ cat chprefix
```

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

```
for f in *.tex
```

```
do
```

```
    newname=$(basename $f tex)latex
```

```
    if [ -e $newname ] ; then
```

```
        mv $f $newname.$$
```

```
    else
```

```
        mv $f $newname
```

```
    fi
```

```
done
```

باعث حذف پسوند از نام فایل می شود



کنترل کننده های حلقه

مثال

- یک اسکریپت بنویسید که از کاربر فرمانها را گرفته و اجرا کند و با ورود **finish** اسکریپت خاتمه پذیرد.

```
$ cat exec-cmnd
```

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

```
while echo "Please enter your command \  
and at the end enter finish"
```

```
read response
```

```
do
```



کنترل کننده های حلقه

مثال

```
case $response
```

```
    " " ) continue
```

خواندن ورودی بعدی

```
    ;;
```

```
    "finish" ) break
```

```
    ;;
```

```
    * ) eval $response
```

```
esac
```

```
done
```

```
exit 0
```



کنترل کننده های حلقه (ادامه)

- یک اسکریپت بنویسید که تمام profile های کاربران را نمایش دهد.

```
$ cat allprofs
```

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

```
FILE=.bash_profile
```

```
for home in `awk -F: '{print $6}' /etc/passwd`  
do
```

```
    [ -d "$home" ] || continue
```

```
    [ -r "$home" ] || continue
```



کنترل کننده های حلقه (ادامه)

```
cd $home
```

```
[ -e $FILE ] && less $FILE
```

```
done
```

```
exit 0
```



دستور select

- این فرمان برای ایجاد منو به کار می رود:

- Syntax

select name in word1 word2 ... wordN

do

List

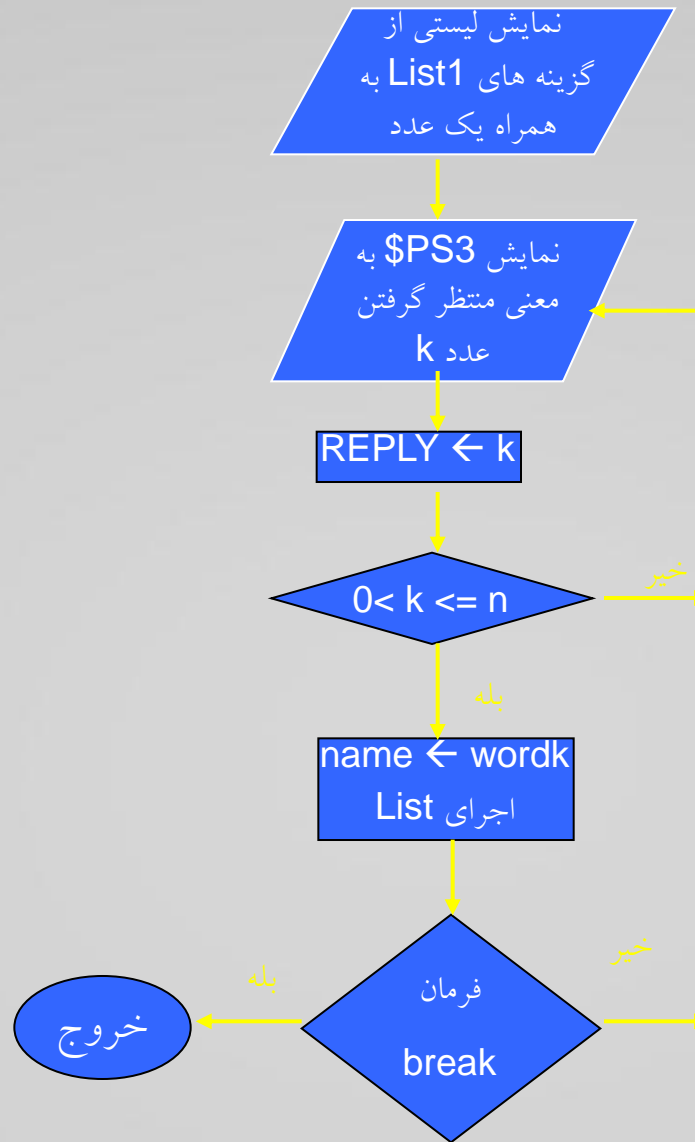
done

List 1

اگر نباشد، name به “\$@”

مقداردهی خواهد شد.





مثال ۱

```
$ cat interactive_del
#!/bin/bash
PS3="Remove file: "
select file in * quit
do
    [ "$file" = "quit" ] && break
    [ -e "$file" ] || rm -f "$file"
done
exit 0
```



مثال ۱ (ادامه)

```
root@localhost:/home/azizi
File Edit View Terminal Go Help
$./interactive_del
1) dead.letter          5) man-pages-1.55.tar  9) pushpop1
2) demo_select          6) mbox                10) script
3) exam01               7) progs.tar           11) quit
4) interactive_del      8) public_html
Remove File: 7
Remove File: 9
Remove File: 12 → عدد نامعتبر
Remove File: 11
$
```



مثال (ادامه)

```
root@localhost:/home/azizi
File Edit View Terminal Go Help
$cat demo_select
#!/bin/bash
PS3="Select an option and press enter:"
select i in Date Host Users Quit
do
    case $i in
        Date) date;;
        Host) hostname;;
        Users) who;;
        Quit) break;;
    esac
done
$
```



مثال



```
root@localhost:/home/azizi
File Edit View Terminal Go Help
$ ./demo_select
1) Date
2) Host
3) Users
4) Quit
Select an option and press enter:1
Sat Sep 4 16:03:44 IRST 2004
Select an option and press enter:4
$
```



عبارت‌های منظم

- عبارت منظم توصیف ساده از یک الگو است که مجموعه ای از کاراکترها را در یک رشته ورودی مشخص می کند.
- عبارت‌های منظم در بسیاری از **utility**های لینوکس استفاده می شوند.
➤ مانند **grep, sed, awk, vi, ...**
- شکل یک عبارت منظم
➤ می تواند یک متن ساده باشد.
▪ **grep linux file**: قابل تطبیق با هر وقوع **linux** در فایل **file**
➤ می تواند یک متن خاص باشد.
▪ **grep '[L]inux' file**: قابل تطبیق با هر وقوع **linux** یا **Linux** در فایل **file**
- تفسیر عبارت منظم از یک **utility** به **utility** دیگر متفاوت است.
➤ عبارت منظم در **sed** کاملاً مشابه با **awk** یا **grep** نمی باشد.



عبارتهای منظم (ادامه)

- عبارتهای منظم با **wildchar**های نام فایل متفاوت می باشند:
 - عبارتهای منظم را **utility**های خاصی تفسیر و تطبیق می کند (مانند **grep**).
 - **wildchar**های نام فایل را، پوسته تفسیر و تطبیق می کند.
 - سیستم **wildchar** در عبارت منظم متفاوت با فایل می باشد.
 - بسط **wildchar** در فایل زودتر از عبارت منظم است.
- بیشتر کاراکترهای خاص عبارت منظم، معانی دیگری برای پوسته دارند بنابراین بهتر است عبارت منظم را داخل **single quote** قرار دهیم.
 - این عمل باعث محافظت کاراکترهای خاص آن از دسترسی پوسته می شود.



مناقصاتی عبارات منظم

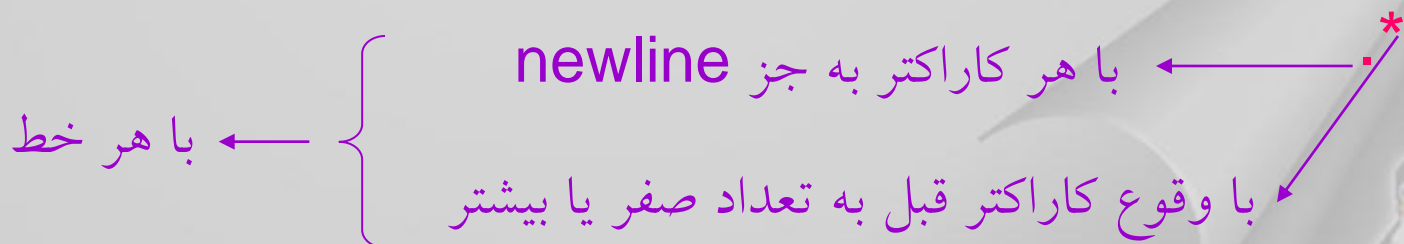
• نقطه .

- قابل تطبیق با یک کاراکتر به جز `newline`
- `a.b` قابل تطبیق با `a.b`, `axb`, `acb`, `a$b`
- غیر قابل تطبیق با `a$bbb`, `axxb`, `ab`

• ستاره *

- قابل تطبیق با وقوع کاراکتر قبل به تعداد صفر بار یا بیشتر.
- `a*b` قابل تطبیق با `b`, `ab`, `aab`, `aaab`, `aaaab`, ...

• سؤال: عبارت زیر با چه چیزی قابل تطبیق است؟



متماکاراکترهای عبارت منظم (ادامه)

• ^

➤ قابل تطبیق با ابتدای خط

'The^' قابل تطبیق با هر خط که با The شروع شود.

• \$

➤ قابل تطبیق با انتهای خط

'\$will' قابل تطبیق با هر خط که به will ختم شود.

• سؤال: عبارت زیر با چه چیزی قابل تطبیق است؟

\$^ ← یعنی ابتدای خط تهی

{ ← با هر خط تهی

↘ انتهای خط نیز تهی



متاکاراکترهای عبارت منظم

character class

• [...]

➤ قابل تطبیق با مجموعه کاراکتر مشخص شده در [...]

➤ '[aeiou]' قابل تطبیق با هر حرف صدا دار.

'[u-x]' قابل تطبیق با هر یک از کاراکترهای 'u', 'v', 'w', 'x'

محدوده از u تا x را مشخص می کند.

'[aA]wk' قابل تطبیق با 'awk' یا 'Awk'

'[-123]' قابل تطبیق با '1', '2', '-' یا '3'

اگر کاراکتر - داخل یک محدوده باشد معنی خاص خود را می دهد.



متاکاراکترهای عبارت منظم

character class(cont)

• [^...]

➤ قابل تطبیق با هر کاراکتر غیر از کاراکترهای مشخص شده در [...]

➤ '[^aeiou]' قابل تطبیق با هر حرف بدون صدا.

'a', 'b', '^', '\$' قابل تطبیق با هر یک از کاراکترهای '[ab^\$]'



کاراکتر ^ داخل یک محدوده معنی خاص خود را نمی دهد.

• [:alpha:]

➤ قابل تطبیق با هریک از کاراکترهای اسکی a-z یا A-Z

• [:alpha:]

➤ قابل تطبیق با هریک از کاراکترهای اسکی a-z یا A-Z



متمم کاراکترهای عبارت منظم

character class(cont)

- `[digit:]`

➤ قابل تطبیق با هر یک از ارقام 0-9

- `[alnum:]`

➤ قابل تطبیق با `[alpha:]` یا `[digit:]`

- `[lower:]`

➤ قابل تطبیق با هر کاراکتر با حرف کوچک.

- `[upper:]`

➤ قابل تطبیق با هر کاراکتر با حرف بزرگ.



مناقصات کارهای عبارت منظم

character class(cont)

• [:space:]

➤ قابل تطبیق با هر یک از کاراکترهای

tab, newline, vertical tab, form feed,
carriage return, space

• [:xdigit:]

➤ قابل تطبیق با هر یک از کاراکترهای مبنای ۱۶.

• [:punct:]

➤ قابل تطبیق با هر یک از کاراکترهای

! " # % & ' () ; < = > ? [\] * + , - . / : ^ _ { | }

• [:graph:]

➤ قابل تطبیق با هر کاراکتر غیر از [:alnum:] و [:punct:]



ماتا کاراکترهای عبارت منظم

معنی خاص کاراکتر \

• $\backslash +$

➤ قابل تطبیق با وقوع کاراکتر قبل به تعداد یک یا بیشتر.

$a\backslash + b$ قابل تطبیق با ab , aab , $aaab$

• $\backslash ?$

➤ قابل تطبیق با وقوع کاراکتر قبل به صفر یا یک.

$a\backslash ? b$ قابل تطبیق با ab , aab , $aaab$

• $a\backslash | b$

➤ قابل تطبیق با وقوع کاراکتر a یا کاراکتر b



متاکاراکترهای عبارت منظم

معنی خاص کاراکتر \

• $\{N\}$

➤ قابل تطبیق با کاراکتر قبلی خود به تعداد N بار.

• $\{N,M\}$

➤ قابل تطبیق با کاراکتر قبل از خودش به تعداد حداقل N و حداکثر M بار.

• $\{2,10\}[a-z]$ قابل تطبیق با تمام رشته های با حرف کوچک به طول ۲ تا ۱۰

• $\{N,\}$

➤ قابل تطبیق با کاراکتر قبل از خودش به تعداد حداقل N .



متماکاراکترهای عبارت منظم

معنی خاص کاراکتر \

الگوی که فقط شامل حرف،
رقم و _ است.

• <

قابل تطبیق با ابتدای یک کلمه.

'<LIN' قابل تطبیق با ابتدای کلمه ای که با 'LIN' شروع شده است.

• >

قابل تطبیق با انتهای کلمه.

'UX>' قابل تطبیق با انتهای کلمه ای که به 'UX' ختم شده است.

• (chars\)

برای به خاطر سپردن قسمتی از یک عبارت منظم.

یادآوری قسمت به ذهن سپرده شده به تعداد n بار.



متاکاراکترهای عبارت منظم

مثال

چون اولین کاراکتر داخل [...] می باشد معنی خاص [...] از بین می رود.

• '[)]}'

➤ قابل تطبیق با هر یک از کاراکترهای '['] ') ' { ' }

• '^\[a-z]\1'

➤ قابل تطبیق با تمام خطوطی که با دو حرف یکسان شروع شده اند.



مناقصاتی‌های عبارت منظم

مثال (ادامه)

• '\{1,\}'

➤ قابل تطبیق با بیشتر از صفر کاراکتر

• '^[aeiou]\{2,\}'

➤ قابل تطبیق با حداقل دو حرف بی صدا در ابتدای خط

• '#\{23\}'

➤ قابل تطبیق با ۲۳ کاراکتر #

• 'At\ (ten\|nine\)tion'

➤ قابل تطبیق با 'Attention' یا 'Atninetion'



GREP

- Get Regular ExPressions یا
- Global Regular Expression Print
- توسط Ken Thompson نوشته شد.
- جست و جو به دنبال یک کلمه یا رشته در فایل(ها).
- اگر تطبیقی پیدا شد خروجی را در خروجی استاندارد نمایش و یا با توجه به گزینه خاص عمل متناسب را انجام می دهد.
- چون **newline** برای جدا کردن خطوط در فایل استفاده می شود، راهی برای جست و جو به دنبال **newline** وجود ندارد.



GREP(cont)

\$ grep [options] regexp file(s)

- C تعداد خطوط تطبیق شده را نمایش می دهد
- i از حالت حساس به نوع حرف چشم پوشی می کند
- l نمایش لیستی از نام فایلها به همراه تطبیق
- v نمایش خطوطی که تطبیقی در آنها پیدا نشده است
- n شماره خط تطبیق شده را نیز چاپ می کند.
- s فقط در صورت وجود خطا، پیام خطا را چاپ می کند



GREP Examples

```
root@localhost:~  
File Edit View Terminal Go Help  
$cat data-file  
northwest      NW      3.0      .98      3      34  
western        WE      5.3      .97      5      23  
southwest      SW      2.7      .8       2      18  
southern       SO      5.1      .95      4      15  
southeast      SE      4.0      .7       4      17  
eastern        EA      4.4      .84      5      20  
northeast      NE      5.1      .94      3      13  
north          NO      4.5      .89      5      9  
central        CT      5.7      .94      5      13
```



GREP Examples(cont)

```
root@localhost:~  
File Edit View Terminal Go Help  
$grep '4$' data-file  
northwest      NW      3.0      .98      3      34  
$  
$grep '5\..' data-file  
western        WE      5.3      .97      5      23  
southern       SO      5.1      .95      4      15  
northeast      NE      5.1      .94      3      13  
central        CT      5.7      .94      5      13  
$  
$grep -n '^south' data-file  
3:southwest    SW      2.7      .8       2      18  
4:southern     SO      5.1      .95      4      15  
5:southeast    SE      4.0      .7       4      17
```



GREP Examples(cont)

```
root@localhost:~  
File Edit View Terminal Go Help  
$grep '[a-z]\{9\}' data-file  
northwest      NW      3.0      .98      3      34  
southwest      SW      2.7      .8        2      18  
southeast      SE      4.0      .7        4      17  
northeast      NE      5.1      .94       3      13  
$
```



GREP Examples(cont)

- یک اسکریپت بنویسید که مشخص کند آیا شخص خاصی در سیستم وجود دارد یا خیر؟

```
$ cat user-logon
```

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

```
If who | grep -s "$1" > /dev/null ; then  
    echo "$1 is logged in"
```

```
else
```

```
    echo "$1 is not logged in"
```

```
fi
```

```
exit 0
```

فقط نمایش پیغام خطا



GREP Examples(cont)

- یک اسکریپت بنویسید که منتظر ورود شخص خاصی به سیستم باشد.

```
$ cat wath-for-in
```

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

```
case $# in
```

```
1) ;;
```

```
2) echo "Usage: watch-for-in username"
```

```
exit 1
```

```
;;
```

```
esac
```



GREP Examples(cont)

```
until who | grep -s "$1" > /dev/null
```

```
do
```

```
    sleep 60
```

```
done
```

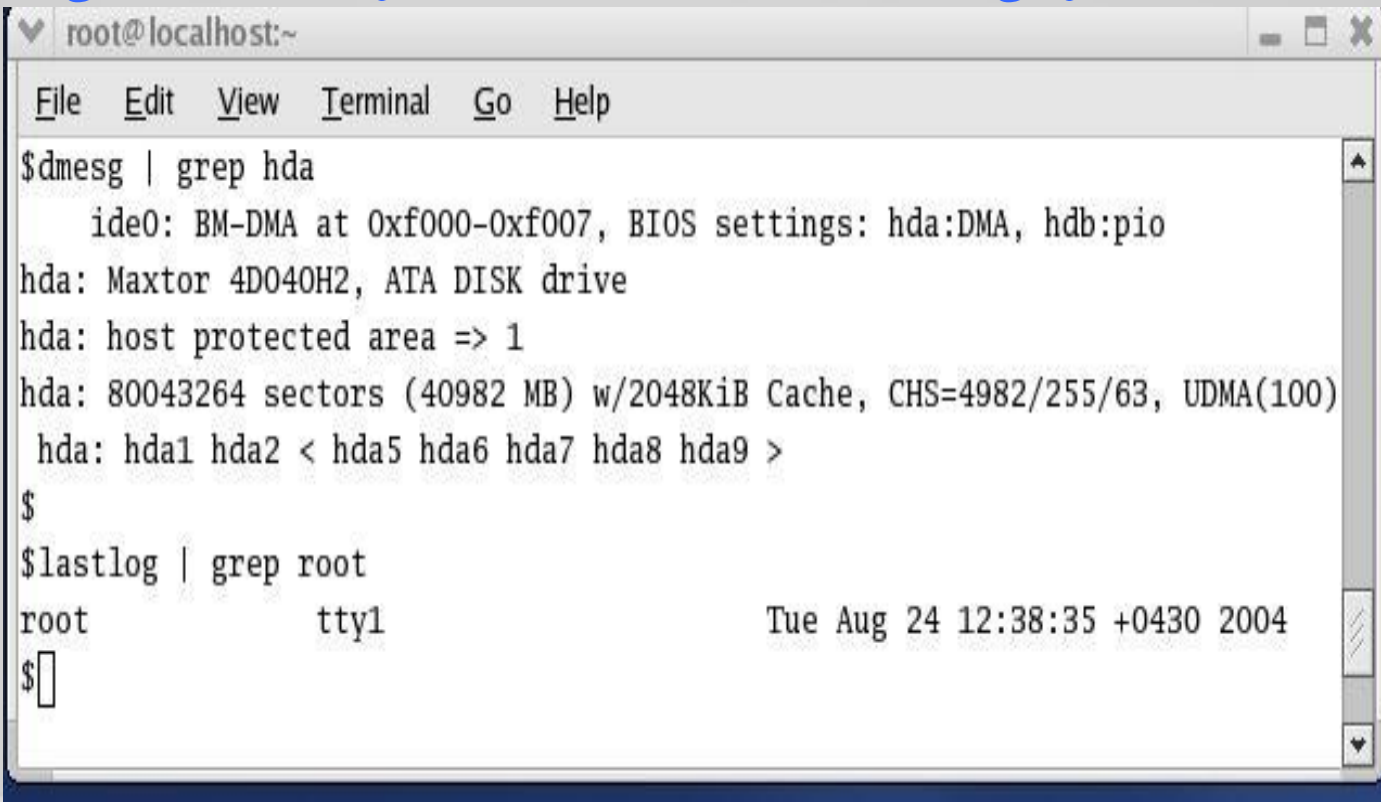
```
echo "$1 has logged on"
```

```
exit 0
```



GREP Examples(cont)

- **dmesg** تمام پیامهای سیستم در هنگام بوت را در خروجی استاندارد نمایش می دهد.
- **lastlog** آخرین زمان **login** همه کاربران را نمایش می دهد.



```
root@localhost:~  
File Edit View Terminal Go Help  
$dmesg | grep hda  
    ide0: BM-DMA at 0xf000-0xf007, BIOS settings: hda:DMA, hdb:pio  
hda: Maxtor 4D040H2, ATA DISK drive  
hda: host protected area => 1  
hda: 80043264 sectors (40982 MB) w/2048KiB Cache, CHS=4982/255/63, UDMA(100)  
    hda: hda1 hda2 < hda5 hda6 hda7 hda8 hda9 >  
$  
$lastlog | grep root  
root          tty1          Tue Aug 24 12:38:35 +0430 2004  
$
```

GREP Examples(cont)

- فرمان `free` حافظه و `cache` مصرفی را به صورت جدول نمایش می دهد.

```
root@localhost:~  
File Edit View Terminal Go Help  
$free  
              total      used      free      shared    buffers     cached  
Mem:        513804    167528    346276           0       11020      82836  
-/+ buffers/cache:    73672    440132  
Swap:      1044184           0    1044184  
$  
$free | grep Mem | awk '{print $4}'  
346260  
$
```



GREP Examples(cont)

- با توجه به شکل زیر، اسکریپتی بنویسید که نام یک پردازش و email شخصی را گرفته و در صورت اجرا شدن آن پردازش به شخص خبر دهد، ضمناً مانع اجرا شدن آن پردازش شود.

```
root@localhost:~/script
File Edit View Terminal Go Help
$ps axg |tail -15
7121 ?      S      0:01 gnome-panel --sm-client-id default2
7123 ?      S      0:01 nautilus --no-default-window --sm-client-id default3
7125 ?      S      0:00 magicdev --sm-client-id default4
7127 ?      S      0:00 egg cups --sm-client-id default6
7129 ?      S      0:00 pam-panel-icon --sm-client-id default0
7131 ?      S      0:00 /usr/bin/python /usr/bin/rhn-applet-gui --sm-client-i
d default5
7132 ?      S      0:00 /sbin/pam_timestamp_check -d root
7136 ?      S      0:00 /usr/libexec/notification-area-applet --oaf-activate-
iid=OAFIID:GNOME_NotificationAreaApplet_Factory --oaf-ior-fd=20
7143 ?      S      0:03 gnome-terminal
7144 ?      S      0:00 [gnome-pty-helpe]
7145 pts/0   S      0:00 bash
8371 ?      S      0:00 /usr/libexec/nautilus-throbber --oaf-activate-iid=OAF
IID:Nautilus_Throbber_Factory --oaf-ior-fd=24
8373 ?      S      0:00 xvidcap
8375 pts/0   R      0:00 ps axg
8376 pts/0   S      0:00 tail -15
$
```

GREP Examples(cont)

```
root@localhost:~/script
File Edit View Terminal Go Help
$cat watcher
#!/bin/bash
echo -n "Enter a process name to watch for: "
read processname
echo -n "Enter a email address to warn when the process runs: "
read email
echo "Press CTRL C to end"
while true
do
    until ps axg | grep "$processname" | grep -s -v "grep" |
        grep -s -v "watcher" > /dev/null
    do
        sleep 5
    done
    dayandtime=`date`
    echo "$processname was running on $dayandtime" >> \
        /var/log/watchprocess
    ps axgu | grep "$processname" | grep -v "grep" | /bin/mail -s "run!" "$email"
    killall -9 $processname
done
$
```

AWK

• در سال ۱۹۹۷ توسط

Alfred V. Aho

Peter J. Weinberger and

Brian W. Kernigan

طراحی و پیاده سازی شد.

- یک زبان برنامه نویسی پردازش داده و تولید گزارش می باشد.
- معمولاً برای انجام عملیات ساده از آن در خط فرمان استفاده می شود.
- همچنین می توان آن را برای کاربردهای بزرگتر در قالب یک برنامه یا اسکریپت نوشت.



AWK(cont)

- با **awk** آدرس دهی در سطح فیلد خواهیم داشت.
- ورودی **awk** می تواند
 - از فایل (ها) بیاید.

- از **pipe** یا **redirection** بیاید.
- مستقیماً از ورودی استاندارد بیاید.

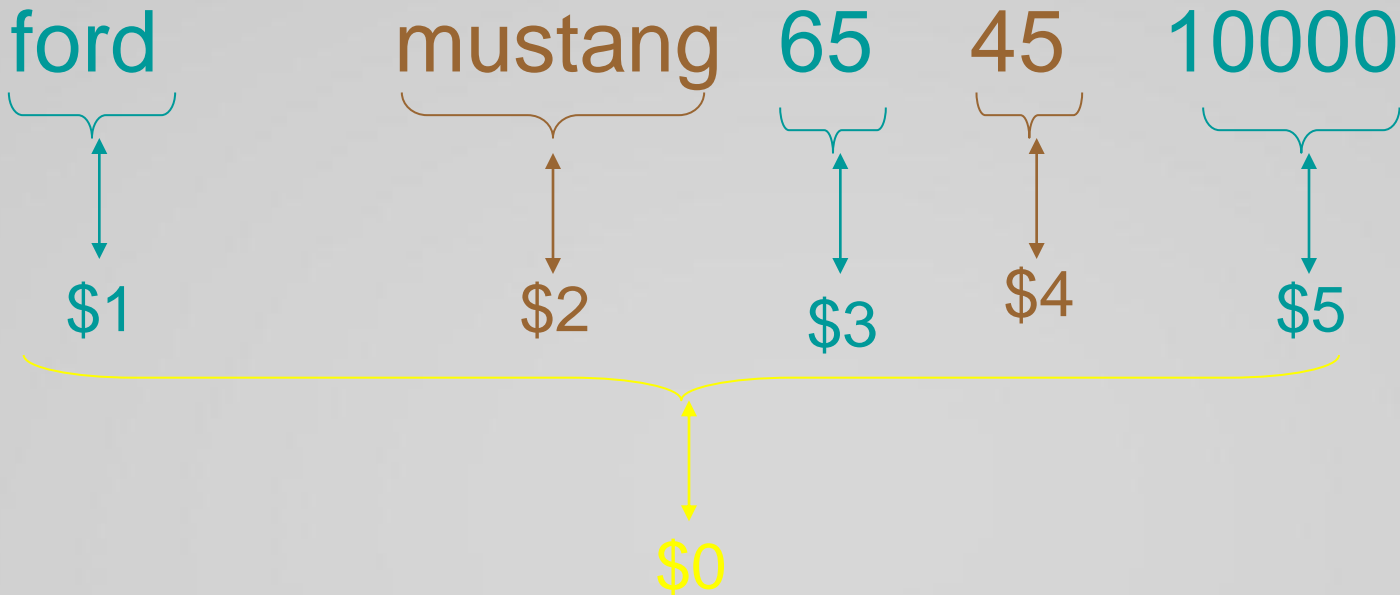
- **awk** ورودی خود را خط به خط برای یافتن یک **pattern** جست و جو کرده و در صورت مشاهده تطبیق **action** متناظر را بر آن خط اعمال می کند.

- اگر **pattern** مشخص نشده باشد، **action** متناظر بر روی تمام خطوط اعمال خواهد شد.



Fields

\$ cat cars | head -1



- هر خط یک رکورد است.
- هر رکورد از یک فیلد یا بیشتر تشکیل شده است.



Fields(cont)

- جدا کننده فیلدها را می توان هر چیزی تعیین کرد حتی یک عبارت منظم.
- جدا کننده پیش فرض فیلدها، `<tab>` می باشد.
- به فیلدها می توان به ترتیب با `$1`، `$2` و ... رجوع نمود.
- `$0` به کل رکورد رجوع می کند.



ساختار یک برنامه awk

BEGIN {action }

- اختیاری

➤ قبل از خواندن داده های ورودی اجرا می شود.

{ action }

- اجرای **action** روی تمام خطوط ورودی.

pattern { action }
:
pattern { action }

- خط ورودی با هر **pattern** تطبیق و اگر تطبیقی صورت گرفت **action** متناظر بر روی آن اجرا می شود.

END { action }

- اختیاری

➤ بعد از خاتمه پردازش داده های ورودی اجرا می شود.



اجرای برنامه awk

- چندین روش برای اجرای یک برنامه awk وجود دارد:

➤ `awk 'program' input_file(s)`

➤ `program` و `input_file(s)` هر دو به عنوان آرگومان خط فرمان می باشند.

➤ `awk 'program'`

➤ `program` به عنوان آرگومان خط فرمان

➤ ورودی از ورودی استاندارد — ← بله، `awk` فیلتر می باشد.

▪ بنابراین می تواند در `pipe` به همراه فیلترهای دیگر استفاده شود

➤ `awk -f program_file_name input_file(s)`

➤ برنامه از یک فایل خوانده می شود.



Patterns

- **awk** مجموعه ای از خطوط ورودی را به ترتیب پوشش کرده و اگر تطبیقی با **pattern** پیدا شد **action** متناظر را روی آن اجرا می کند.

- **pattern** می تواند:

عبارت منظم باید داخل

/RE/ باشد

- /regular expressions/
- relational expressions
- pattern, pattern



Patterns

relational expressions

- **awk** می تواند هر فیلد را با یک عدد یا عبارت منظم مقایسه کند.

➤ Variable operator pattern

رشته ای	~	→	دو رشته مساوی
	!~	→	دو رشته نامساوی
منطقی	&&	→	Logical AND
		→	Logical OR
	!	→	Logical NOT



Patterns

relational expressions

➤ Variable operator pattern

محاسباتی

<	→	کوچکتر
<=	→	کوچکتر مساوی
>	→	بزرگتر
>=	→	بزرگتر مساوی
=	→	تساوی عددی
!=	→	نامساوی عددی



Patterns

pattern1, pattern2

- محدوده ای از خطوط را انتخاب می کند که:
 - اولین خط آن با pattern1 تطبیق شود.
 - آخرین خط آن با pattern2 تطبیق شود.
- ممکن است چندین گروه از خطوط را برگرداند.



Actions

- **action** شامل یک یا چند فرمان، تابع یا مقداردهی متغیر احاطه شده در **{ }** است که با **newline** یا **semicolon** از هم جدا شده اند.

- فرمانها به چهار دسته تقسیم می شوند:

- مقدار دهی متغیر
- فرمانهای نمایش
- توابع توکار
- دستورهای کنترل جریان



بعضی از متغیرهای توکار awk

معنی	متغیر
شماره خط جاری	NR
جدا کننده فیلدها (پیش فرض newline و tab)	FS
جدا کننده فیلدهای خروجی (پیش فرض blank)	OFS
تعداد فیلدها	NF
جدا کننده رکورد (پیش فرض newline)	RS
نام فایل ورودی	FILENAME

فرمانهای نمایش

print

- فرمان **print** ابزار خروجی ساده **awk** است.
- می توانید خروجی این فرمان را با استفاده از متاکاراکترهای **>** و **>>** به یک فایل هدایت کرد.

```
$ls -l | awk '{print $2}'
```



فرمانهای نمایش

printf

- این فرمان نمایش یک خروجی قالب بندی شده را ممکن می سازد.
- Syntax
- `printf("format string", var1, var2, ...)`
- **Foramt specifiers**
 - `%c` - single character
 - `%d` - number
 - `%f` - floating point number
 - `%s` - string
 - `\n` - Newline
 - `\t` - Tab



فرمانهای نمایش

printf

- Format modifiers
 - - left justify in column
 - n column width
 - .n number of decimal place to print



فرمانهای نمایش printf

- `$ echo "Linux" | awk '{printf("%-15s\n",$1)}'`

|Linux|
۱۰ کاراکتر

باعث می شود اعلان پیش فرض
پوسته در خط بعد نمایش داده شود.

- `$ echo "Linux" | awk '{printf("%15s\n",$1)}'`

|Linux|
۱۰ کاراکتر

- `$ echo 27.876 | awk '{printf("%15.2f\n",$1)}'`

|27.87|
۱۰ کاراکتر



مثال

- با توجه به شکل به سؤالات زیر پاسخ دهید.

```
root@localhost:~/script
File Edit View Terminal Go Help
$ls -l
total 40
-rwxr--r-- 1 root root 255 Sep 2 11:23 blank-rename
-rwxr--r-- 1 root root 246 Sep 5 15:13 countfldr.awk
-rwxr--r-- 1 root root 78 Jul 6 18:04 fat-mnt
-rwxr--r-- 1 root root 80 Sep 5 08:29 nlog
-rwxr-xr-x 1 root root 1153 Sep 5 08:54 t1.txt
-rwxr--r-- 1 root root 71 Aug 8 15:29 te
-rwxr-xr-x 1 root root 682 Sep 5 08:54 timedkill
-rwxr-xr-x 1 root root 1153 Sep 5 08:54 t.txt
-rwxr--r-- 1 root root 530 Sep 5 14:51 watcher
drwxr-xr-x 2 root root 4096 Sep 5 15:42 xvidcap
$
```



مثال (ادامه)

- `$ ls -l | awk '/^d/' { print "rm -r "$9 }' | bash`

• حذف فقط دایرکتوریهای شاخه جاری

- `$ ls -l | grep -v '^d' |`
`> awk '{ print "rm -f"$9}' | bash`

• حذف فقط فایل‌های شاخه جاری



مثال

- اسکرپتی بنویسید که تعداد فایلها و دایرکتوریهای شاخه جاری را نمایش دهد:

```
root@localhost:~/script
File Edit View Terminal Go Help
$cat countfldr.awk
#!/bin/awk -f
BEGIN{filecount = 0 ; dircount = 0 }
/^-/ {filecount = filecount +1 }
/^d/ {dircount = dircount + 1 }
END { print "\n"
      print "Total number of Files      :" filecount
      print "Total number of Directories  :" dircount }
$
$ls -l | awk -f countfldr.awk ← اجرای اسکرپت

Total number of Files      :12
Total number of Directories :1
$
```



توابع توکار

length([argument])

- این تابع طول رشته argument را می دهد.
- اگر argument نیاید، این تابع طول 0\$ را می دهد.



توابع توکار

index() , substr()

- تابع `index(string,target)` موقعیت اولین رخداد `target` در `string` را می‌دهد.
- تابع `substr(string,start[,length])` قسمتی از `string` که با `start` شروع می‌شود به طول `length` کاراکتر را بر می‌گرداند.



دستورهای کنترل جریان

if - else

- Syntax
 - if (condition is true or non zero){
 statement1(s)
}
else{
 statement2(s)
}
- اگر statement یکی باشد
{ } لازم نیست.



دستورهای کنترل جریان

عملگرهای if - else

- $A < B$ A is less than B
- $A \leq B$ A is less than or equal B
- $A == B$ A equals B
- $A > B$ A is greater than B
- $A \geq B$ A is greater than or equal B
- $A != B$ A is not equal B
- $A \sim /RE/$ A contains the regular expression RE



دستورهای کنترل جریان

while

- Syntax
- while (condition is true or non zero){
 statement(s)
}

اگر statement یکی باشد
{ } لازم نیست.

• تا وقتی که



دستورهای کنترل جریان

for

- `for (expression1;expression2;expression3){
statement(s)
}`

- عملکرد حلقه بالا شبیه حلقه **while** باشد:

```
expression1  
while (expression2){  
statement(s)  
expression3  
}
```



مثال

- دستورات زیر باعث می شوند هر کاراکتر فیلد جداگانه ای محسوب شود:

```
root@localhost:~  
File Edit View Terminal Go Help  
$echo a b |awk 'BEGIN { FS="" }  
> {  
>     for (i = 1; i <= NF; i = i + 1)  
>         print "Field", i, "is", $i  
>     }'  
Field 1 is a  
Field 2 is  
Field 3 is b  
$
```



دستورهای کنترل جریان

do - while

- `do{`
`statement(s)`
`} while(condition is false or non zero)`
 - در حلقه `do while` ابتدا `condition` تست می شود و بعد بنده حلقه اجرا می گردد.
 - در حلقه `do while` ابتدا بنده حلقه اجرا و بعد `condition` تست می شود.
- بنابراین در این حلقه حداقل یک بار اجرای بنده آن را خواهیم داشت.



دستورهای کنترل جریان

Action break, continue, next

• break

➤ باعث خروج از بدنه حلقه می شود.

▪ و باعث خروج از اسکریپت **awk** نمی شود.

➤ کنترل به خط بعد از بدنه حلقه منتقل می شود.

• continue

➤ باعث می شود که **awk** به ازاء مقدار کنونی از اجرای باقیمانده بدنه حلقه خودداری کرده و تکرار بعدی آغاز شود.

• next

➤ باعث می شود اسکریپت **awk** از اول اجرا شود.



awk example

\$ cat cars

plym	fury	77	73	2500
chevy	nova	79	60	3000
ford	mustang	65	45	10000
volvo	gl	78	102	9850
ford	ltd	83	15	10500
chevy	nova	80	50	3500
fiat	600	65	115	450
honda	accord	81	30	6000
ford	thundbd	84	10	17000
toyota	tercel	82	180	750
ford	bronco	83	25	9500



awk example(cont)

\$ awk '/chevy/ { print }' cars

chevy	nova	79	60	3000
chevy	nova	80	50	3500

کل رکورد حاوی chevy را نمایش می دهد.

\$ awk '/chevy/' cars

chevy	nova	79	60	3000
chevy	nova	80	50	3500

action پیش فرض نمایش کل رکورد حاوی chevy است.

\$ awk '/^[th]/' cars

toyota	tercel	82	180	750
honda	accord	81	30	6000

نمایش کلیه خطوطی که با t یا h شروع می شوند.



awk example(cont)

```
$ awk '{print $3 , $1}' cars
```

77 plym

79 chevy

65 ford

78 volvo

83 ford

80 chevy

:

کاما نشان می دهد که فاصله
جدا کننده خروجی است.



awk example(cont)

```
$ awk '{print $3 $1}' cars
```

77plym

79chevy

65ford

78volvo

83ford

80chevy

: :

چون کاما وجود ندارد خروجی های
به هم متصل می باشند



awk example(cont)

```
$ awk '{print $3, $1} { price += $NF }  
      END { print price}' cars
```

فیلد آخر هر خط

```
77    plym  
79    chevy  
65    ford  
78    volvo  
83    ford  
80    chevy  
:  
:  
73050
```



awk example(cont)

```
$ awk ' $1 ~! / ^[chf] / ' cars
```

volvo	gl	78	102	9850
-------	----	----	-----	------

toyota	tercel	82	180	750
--------	--------	----	-----	-----

```
$ awk '$5 >= 2000 && $5 < 9000' cars
```

plym	fury	77	73	2500
------	------	----	----	------

chevy	nova	79	60	3000
-------	------	----	----	------

chevy	nova	80	50	3500
-------	------	----	----	------

honda	accord	81	30	6000
-------	--------	----	----	------



awk example(cont)

\$ awk ' \$1 ~ /^h/ ' cars

honda	accord	81	30	6000
-------	--------	----	----	------

\$ awk '/chevy/ , /ford/' cars

chevy	nova	79	60	3000
ford	mustang	65	45	10000
chevy	nova	80	50	3500
fiat	600	65	115	450
honda	accord	81	30	6000
ford	thundbd	84	10	17000

گروه ۱

گروه ۲



awk example(cont)

\$4

```
root@localhost:~  
File Edit View Terminal Go Help  
$df  
Filesystem            1K-blocks      Used Available Use% Mounted on  
/dev/hda9              8949884    1675712    6819536   20% /  
/dev/hda8              101057       9325     86515   10% /boot  
none                   256900         0    256900    0% /dev/shm  
/dev/hda6             15351128   1134104   14217024    8% /mnt/e  
$df | awk '$4 > 300000'  
Filesystem            1K-blocks      Used Available Use% Mounted on  
/dev/hda9              8949884    1675644    6819604   20% /  
/dev/hda6             15351128   1134104   14217024    8% /mnt/e  
$
```



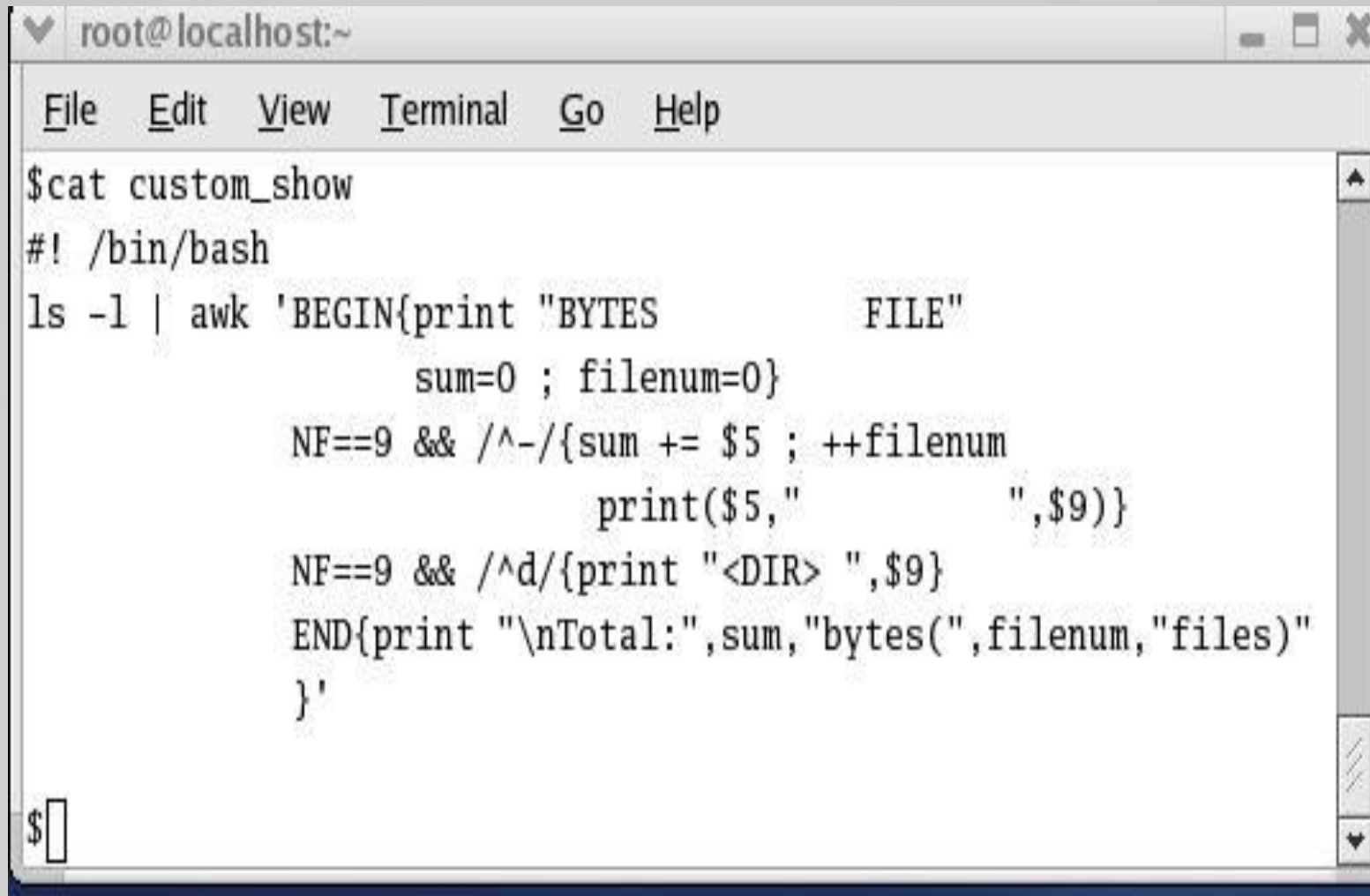
awk example(cont)

- یک برنامه **awk** بنویسید که محتوای دایرکتوری جاری را به صورت زیر نمایش دهد.

```
root@localhost:~  
File Edit View Terminal Go Help  
BYTES      FILE  
1161      anaconda-ks.cfg  
315       custom_show  
15910     install.log  
3019     install.log.syslog  
<DIR>  script  
<DIR>  xvidcap  
  
Total: 20405 bytes( 4 files)  
$
```



awk example(cont)



```
root@localhost:~  
File Edit View Terminal Go Help  
$cat custom_show  
#!/bin/bash  
ls -l | awk 'BEGIN{print "BYTES          FILE"  
                sum=0 ; filenum=0}  
NF==9 && /^-/{sum += $5 ; ++filenum  
                print($5,"          ",$9)}  
NF==9 && /^d/{print "<DIR> ",$9}  
END{print "\nTotal:",sum,"bytes(",filenum,"files)"  
}'  
$
```

awk examples(cont)

- یک اسکریپت بنویسید که مسیری را از کاربر بعنوان پارامتر گرفته و بزرگترین فایل در آن مسیر را نمایش دهد.

```
$ cat find-big-file
```

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

```
#usage: find-big-file path
```

```
echo -e "Please enter your path:\c"
```

```
read path
```

```
if [ ! -d "$path" ]
```

```
then
```



awk Examples(cont)

```
echo "Your path is not valid."
```

```
exit 1
```

```
fi
```

مرتب کردن بر حسب عددی

```
echo "The biggest file in $path is"
```

```
big=`ls -l | awk '/^-/ {print $5}' | \
```

پیدا کردن بزرگترین فایل

```
sort -nr | head -1`
```

```
ls -l | grep $big | awk '/^-/ { print $9 }'
```

```
exit 0
```

نمایش به صورت معکوس

نمایش نام بزرگترین فایل



awk Examples(cont)

- با توجه به فایل `/etc/passwd` که ده خط اول آن در زیر آمده است، به سؤالات زیر پاسخ دهید:

نام واقعی کاربر

مسیر پوسته مربوط به کاربر

```
root@localhost:~  
File Edit View Terminal Go Help  
$cat /etc/passwd | head -10  
root:x:0:0:root:/root:/bin/bash  
bin:x:1:1:bin:/bin:/sbin/nologin  
daemon:x:2:2:daemon:/sbin:/sbin/nologin  
adm:x:3:4:adm:/var/adm:/sbin/nologin  
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin  
sync:x:5:0:sync:/sbin:/bin/sync  
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown  
halt:x:7:0:halt:/sbin:/sbin/halt  
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin  
news:x:9:13:news:/etc/news:  
$
```



awk Examples(cont)

- نام واقعی تمام کاربرانی که به A شروع می شوند:

جدا کننده فیلد

```
$ awk -F : '$5 ~ /^A/ { print }' /etc/passwd
```

- نمایش تعداد کاربران از هر نوع پوسته sh, bash, csh و ksh:

```
$cat count_shell_users.awk  
BEGIN {FS=":"}
```



awk Examples(cont)

```
/bash$/ {bash++}
```

→ انتهای خط را مشخص می کند

```
/sh$/ { sh++}
```

```
/csh$/ { csh++}
```

```
/ksh$/ { ksh++}
```

```
END { print bash,"Users of bash."
```

```
      print sh,"Users of sh."
```

```
      print csh,"Users of csh."
```

```
      print ksh,"Users of ksh."}
```



awk Examples(cont)

```
$awk -f count_shell_users.awk /etc/passwd
```

۲۹۹ users of bash

۱ users of sh

۴ users of csh

۶ users of ksh

نمایش هر قسمت از متغیر **PATH** در خطی جداگانه:

```
$ echo $PATH | awk 'BEGIN { RS=:; }  
{ print $0 }'
```



awk Examples(cont)

- اسکرپتی بنویسید که اگر کاربری بیشتر از سه login داشت، نام آن کاربر را به مسئول سایت email بزند.

```
root@localhost:~  
File Edit View Terminal Go Help  
$w  
21:56:42 up 3:32, 5 users, load average: 0.06, 0.03, 0.01  
USER      TTY      FROM            LOGIN@  IDLE   JCPU   PCPU   WHAT  
root      tty1     -               6:29pm 19:07   0.82s  0.01s  /bin/s  
azizi     tty2     -               9:37pm 19:21   0.00s  0.00s  -bash  
azizi     tty3     -               9:20pm 20:57   0.01s  0.01s  -bash  
azizi     tty4     -               9:37pm 19:13   0.00s  0.00s  -bash  
root      pts/0    :0.0            9:38pm 0.00s   0.05s  0.01s  w  
$
```



awk Examples(cont)

```
root@localhost:~/script
File Edit View Terminal Go Help
$cat more3log
#!/bin/bash
w | awk '{print $1}' | tee totaluser.$$ | sort -u > sortuser.$$
while read line
do
    count=`grep -c $line totaluser.$$`
    if [ $count -gt 2 ]
    then
        echo "$line , $count" >> max-user-login.$$
    fi
done < sortuser.$$
mail -s "Users with more than three login" azizi \
< max-user-login.$$
rm -f totaluser.$$ sortuser.$$ max-user-login.$$
exit 0
$
```

sed, The Stream Editor

- ویراستاری خاص منظوره است که فرمانهای خود را فقط از خط فرمان و یا از یک اسکریپت می تواند بگیرد و به صورت محاوره ای نمی تواند استفاده شود.
- ورودی ها همگی از ورودی استاندارد و خروجی نیز در خروجی استاندارد نوشته می شود.
- بنابراین فایل ویرایش شده هیچ تغییری نمی کند، اما فایل ورودی به همراه تغییرات در خروجی استاندارد نوشته می شود.
- اگر لازم است تغییرات حاصل از sed را ذخیره کنید باید خروجی استاندارد را به یک فایل redirect نمایید:

```
$ sed -f scriptfile editfile > outputfile
```



sed Syntax

- `$ sed [-n] [-e] ['command'] [file...]`
- `$ sed [-n] [-f scriptfile] [file ...]`

➤ `-n` فقط خطوطی که فرمان `p` یا `s` بر روی آنها اجرا می شود را نمایش می دهد

➤ `-e command` آرگومان بعدی یک فرمان ویرایش است و نام فایل نمی باشد.

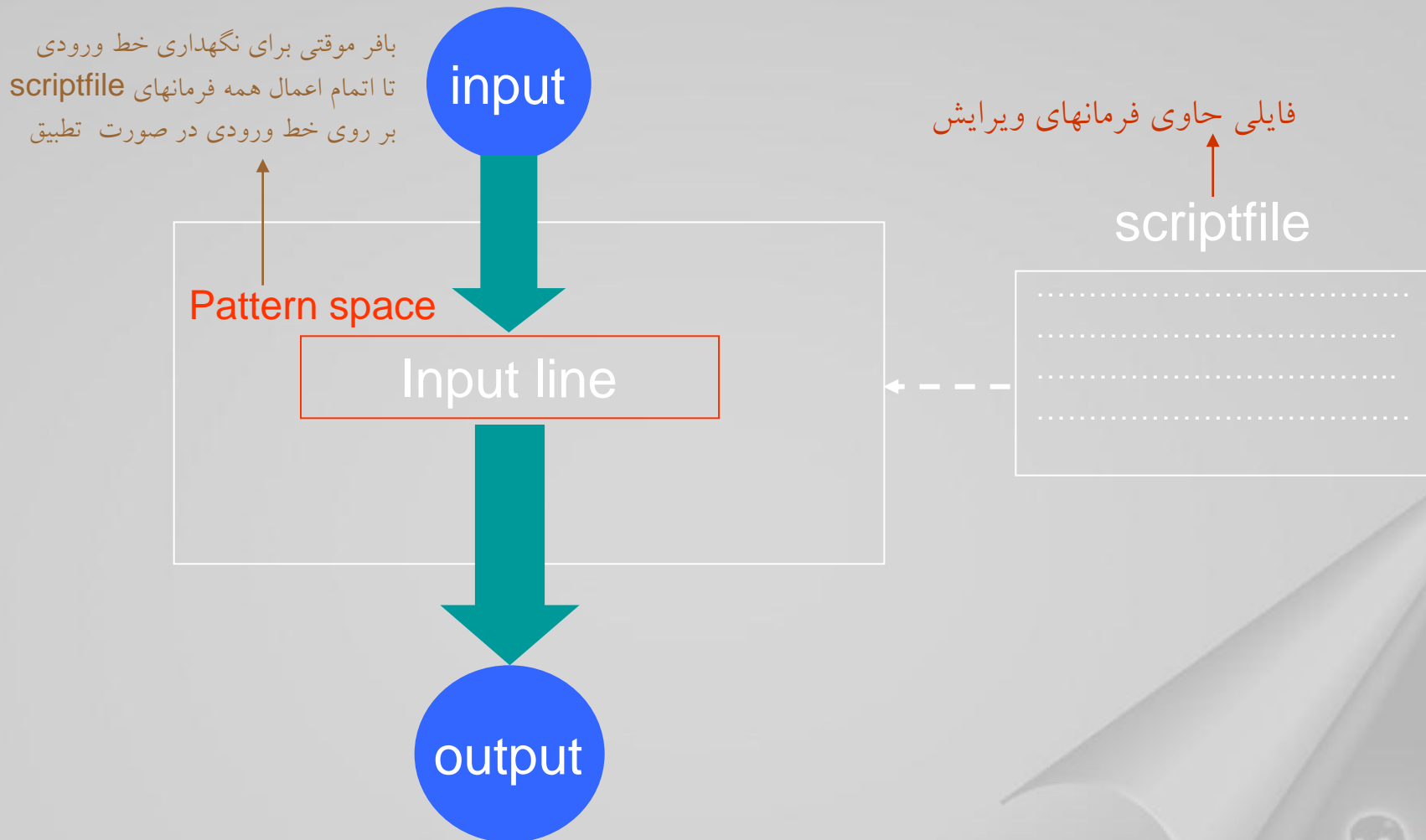
➤ `-f scriptfile` آرگومان بعدی نام فایلی است که حاوی فرمانهای ویرایش است.

➤ شکل‌های مختلف فراخوانی `sed` همگی مشابه هستند:

`$sed [options] script file_argument(s)`



نحوه برخورد sed با فایلها



Sed script

- اسکریپت یک فایل حاوی فرمانهای ویرایش است.
- هر فرمان شامل یک آدرس و یک **action** می باشد که آدرس **pattern**(عبارت منظم) نیز می تواند باشد.
- هر وقت یک خط از فایل ورودی خوانده شد **sed** اولین فرمان از اسکریپت را خوانده و آدرس یا **pattern** را با آن مقایسه می کند:
 - اگر تطبیقی رخ داد فرمان اجرا می شود.
 - اگر تطبیقی رخ نداد از فرمان صرفنظر می شود.
- **sed** سپس همین عمل را برای فرمانهای بعدی در اسکریپت تکرار می کند.



Sed script(cont)

- وقتی به انتهای اسکریپت رسید آنگاه **sed** خط جاری را در خروجی نمایش می دهد، مگر اینکه گزینه **-n** فعال باشد.
- **sed** سپس خط ورودی بعدی را می خواند و دوباره از ابتدای فایل اسکریپت شروع به خواندن فرمانها می کند.
- فایل ورودی اولیه بدون تغییر می ماند و هر بار **sed** پردازش خود را به ترتیب بر روی یک کپی از خطوط آن انجام و نتیجه را در خروجی استاندارد نمایش می دهد.



Sed commands

- شکل کلی فرمانهای sed به صورت زیر است:
➤ `[address[, address]] [!] command [arguments]`
- sed هر خط ورودی را در pattern space کپی می کند:
➤ اگر آدرس فرمان با خط موجود در pattern space تطبیق شد، فرمان بر آن خط اعمال می شود.
➤ اگر فرمان آدرسی نداشت، بر هر خط که در pattern space بیاید اعمال می شود.
➤ اگر فرمانی خط موجود در pattern space را تغییر داد، فرمان بعدی بر خط تصحیح شده اعمال خواهد شد.
- وقتی همه فرمانها خوانده شدند، خط موجود در pattern space در خروجی نوشته شده و خط جدید به داخل pattern space خوانده می شود.



آدرس دهی

- آدرس می تواند شماره خط و یا یک pattern که در اسلش (/pattern/) قرار گرفته است باشد.
- عبارت منظم pattern را توصیف می کند.
- بیشتر فرمانها دو آدرس را قبول می کنند
- اگر فقط یک آدرس مشخص شده باشد فرمان فقط بر آن خط اعمال می شود.
- اگر دو آدرس که با کاما متمایز شده اند مشخص شده باشد فرمان بر روی محدوده ای از خطوط بین اولین و دومین آدرس اعمال می شود.
- عملگر ! برای منفی کردن محدوده آدرس استفاده می شود.
- مثلاً **address ! command** باعث می شود فرمان بر تمام خطوطی که آدرس آنها **address** نیست اعمال شود.



آدرس دهی (ادامه)

- برای اعمال چندین فرمان بر یک آدرس از براکت { } استفاده می شود.

```
[/pattern[/pattern/]]{
```

```
command1
```

```
command2
```

```
command3
```

```
}
```

باید آخرین کاراکتر در خط باشد و فقط کاراکتر
newline بلافاصله بعد از آن بیاید

در این خط فقط باید این کاراکتر بیاید و نه
کاراکتر دیگری



چند مثال آدرس دهی

- d

- 6d

- /^\$/d

blank

- 1,10d

۱۰

- 1,/^\$/d

blank

- /^\$/,10d

۱۰

blank

- /^\$/, \$d

blank

یعنی خط آخر



معرفی چند فرمان sed

- اگر چه sed فرمانهای ویرایش زیادی دارد اما ما زیر مجموعه کوچکی از آنها را بررسی می کنیم:

- s - substitute
- p - print
- a - append
- ! – Don't
- i - insert
- r - read
- c - change
- w – write
- d - delete
- t – transform



Substitute

- Syntax:

➤ [address(es)]s/pattern/replacement/[flags]

↓
n •

۵۱۲

pattern

pattern space

p •

pattern

g •

file

pattern space

w file •



Substitute

```
$ sed 's/is/IS/g' sedin
```

thIS IS a test

for the

next sixty

seconds

thIS station will

conducting a test

```
$ cat sedin
```

this is a test

for the

next sixty

seconds

this station will

be conducting a test



Append, insert and change

- شکل این فرمانها نسبتاً عجیب است زیرا آنها را باید در چندین خط مشخص نمود.

- Append

[address]a\

text

برای escape کردن newline

- Insert

[address]i\

text

- Change

[address(es)]c\

text



Delete

- **Delete** صفر، یک یا دو آدرس می تواند قبول کند و **Pattern space** جاری را حذف کند.
- **Pattern space** که با اولین آدرس تطبیق شد را حذف می کند.
- خطوط در محدوده دو آدرس را حذف می کند.
- وقتی **Delete** اجرا شد فرمان دیگری به **pattern space** اعمال نمی شود. در عوض خط جدید به داخل **pattern space** خوانده شده و اسکرپیت از بالا شروع به اجرا خواهد شد.
- **Delete** تمام خط را حذف می کند. برای حذف قسمتی از خط آن را با **blank** جایگزین کنید.



Delete

```
$ sed '/^next/ d' sedin
```

this is a test

for the

seconds

this station will

be conducting a test

```
$ sed /^next/,/^be/ d' sedin
```

this is a test

for the

```
$ cat sedin
```

this is a test

for the

next sixty

seconds

this station will

be conducting a test



Print

```
$ sed '/^for/,/^seconds/ p' sedin
```

this is a test

for the

for the

next sixty

next sixty

seconds

seconds

this station will

be conducting a test

```
$ cat sedin
```

this is a test

for the

next sixty

seconds

this station will

be conducting a test



Don't

- اگر بعد از آدرس علامت تعجب بیاید، فرمان به تمامی خطوط به غیر از آن آدرس یا محدوده آدرس اعمال می شود.

```
$ sed '2,3 !d' sedin
```

this is a test

this station will

be conducting a test

```
$ cat sedin
```

this is a test

for the

next sixty

seconds

this station will

be conducting a test



Read and Write

- با استفاده از این دو فرمان می توان مستقیماً با فایلها کار کرد.

➤ Read: [address]**r** filename

باید حداقل یک فاصله باشد.

در انتهای نام فایل باید حتماً
کاراکتر newline بیاید.

➤ Write: [address,[address]]**w** filename

- آرگومان هر دو فرمان فقط نام فایل می باشد.

- فرمان read بعد از خواندن خط به آدرس address شروع
به خواندن فایل filename به داخل pattern space
می کند.



Read and Write(cont)

- فرمان **write** آدرس یک خط (یا محدوده ای از خطوط) را گرفته و محتوای **pattern space** را به داخل فایل **filenam** می نویسد.
- در فرمان **read** اگر فایل **filename** موجود نباشد، **sed** پیغام خطایی نخواهد داد.
- در فرمان **write** اگر فایل **filename** موجود نباشد، ایجاد و در غیر اینصورت بازنویسی خواهد شد.



Read and Write(cont)

```
$ sed '2 r test' sedin
```

this is a test

for the

insert text1

insert text2

insert text3

next sixty

seconds

```
$ cat sedin
```

this is a
test

for the

next sixty

seconds

```
$ cat test
```

Insert text1

Insert text2

Inset text3



Read and Write(cont)

```
$ sed '2,3 w test' sedin
```

this is a test

for the

next sixty

seconds

```
$ cat test
```

for the

next sixty

```
$ cat sedin
```

this is a
test

for the

next sixty

seconds



Transform

- این فرمان شبیه **tr** بوده و یک جایگزینی یک به یک (کاراکتر به کاراکتر) انجام می دهد.
- این فرمان صفر، یک یا دو آدرس می تواند قبول کند.

```
$ sed 'y/this/siht/' sedin
```

```
siht ht a sets
```

```
for sie
```

```
nexs thxsy
```

```
$ cat sedin
```

```
this is a test
```

```
for the
```

```
next sixty
```



Sed example

- اسکرپتی بنویسید که اگر پارتیشنی بیشتر از ۹۰٪ آن پر شد، نام آن را به مسئول سایت email بزند.

```
root@localhost:~  
File Edit View Terminal Go Help  
$df  
Filesystem      1K-blocks      Used Available Use% Mounted on  
/dev/hda9        8949884    1672640    6822608   20% /  
/dev/hda8        101057       9325      86515   10% /boot  
none             256900         0     256900    0% /dev/shm  
/dev/hda6       15351128   1135000   14216128    8% /mnt/e  
$
```



Sed example

\$cat fullpartition

#!/bin/bash

MAX=90

df | sed 's/%/ /g' |

awk '{ if (\$5 > \$MAX) ; print \$6 }' |

mail -s "Full Partition Names" root

exit 0



تابع

- یک توالی از فرمانها که می توانند در هر جای یک اسکریپت فراخوانی شوند.
- استفاده برای
 - استفاده مجدد از یک توالی از فرمانها بدون نیاز به نوشتن مجدد آنها
 - صرفه جویی در وقت
 - نوشتن ساده تر برنامه
- تعریف تابع:

```
function function-name {  
    command1  
    comman2  
    :  
}
```

باید فاصله باشد



تابع (ادامه)

```
function-name () {  
    command1  
    command2  
    :  
}
```

و یا

- فراخوانی تابع با صدا زدن نام آن می باشد.
- تعریف تابع باید قبل از اولین فراخوانی آن بیاید.



مثال

```
$ cat demo_func
```

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

```
function echo_it {  
    echo "In function echo_it"  
}
```

بدنه تابع حتماً قبل از
فراخوانی آن باید بیاید

```
echo_it
```

```
exit 0
```

← فراخوانی تابع



ارسال پارامتر به تابع

- یک تابع می تواند آرگومان خط فرمان داشته باشد:

```
$ cat demo_func2  
#!/bin/bash  
echo_it () {  
    echo "Argument1 is $1"  
    echo "Argument2 is $2"  
}  
echo_it arg1 arg2  
exit 0
```



Functions in pipe

- می توان از تابع در pipe استفاده کرد:

```
$ cat demo_func3
```

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

```
ls_sorter () { sort -n +4; }
```

```
ls -al | ls_sorter
```

- تابع فراخوانی شده در pipe در یک پوسته جدید اجرا می شود.

➤ بنابراین متغیرهای تعریف شده در تابع، با خاتمه آن در دسترس نخواهند بود.



Inherited variables

- متغیرهای تعریف شده قبل از فراخوانی تابع، در تابع در دسترس می باشند:

```
$ cat demo_func3
```

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

```
func_y () {
```

```
echo "A is $A"
```

```
return 7
```

```
}
```

```
A="One"
```

```
if [ $? -eq 7 ] ; then .... ; fi
```

متغیر A در تابع تعریف شده است.



Function example

- تابعی به نام **today** بنویسید که با فراخوانی نام آن در خط فرمان تاریخ روز را به صورت زیر نمایش دهد:

This is a Friday 27 in August of 2004(06:57:15 PM)

- اگر بخواهیم هر کاربری بتواند این فرمان را اجرا کند باید تابع را به اسکریپت **/etc/bashrc** اضافه کنیم.
- اما اگر بخواهیم این تابع را کاربر خاصی اجرا کند باید به اسکریپت **~/.bashrc** اضافه شود.
- خطوط زیر را با استفاده از یک ویراستار مانند **vi** به اسکریپت **~/.bashrc** اضافه می کنیم.



Function example(cont)

```
root@localhost:~  
File Edit View Terminal Go Help  
# .bashrc  
  
# User specific aliases and functions  
  
alias rm='rm -i'  
alias cp='cp -i'  
alias mv='mv -i'  
  
# Source global definitions  
if [ -f /etc/bashrc ]; then  
    . /etc/bashrc  
fi  
PS1="\u@\w >"  
today()  
{  
echo This is a `date +"%A %d in %B of %Y(%r)``  
return  
}  
17,1 All
```

خطوط اضافه شده به

این اسکریپت



اشکال زدایی یک اسکریپت

- قبل از رفع مشکل یک اسکریپت رعایت نکات زیر در هنگام نوشتن آن بسیار مفید است:

- فهمیدن چگونگی انجام کار بصورت دستی
- نوشتن اسکریپت در قالب قسمت‌های کوچک
- آزمایش هر قسمت و بعد ادامه کار
 - نمایش متغیرهای تصحیح شده
 - نمایش فرمانی که باید اجرا شود
- استفاده از توضیحات در اسکریپت



اشکال یابی یک اسکریپت (ادامه)

استفاده از توضیحات در اسکریپت

- پوسته رشته ای که بعد از کاراکتر # (به جز #!) بیاید را به عنوان توضیح تلقی می کند و از آن صرف نظر می نماید.
- عنوان اسکریپت

➤ گویای عملی است که اسکریپت انجام می دهد

➤ ورودی های معمول
نویسنده

- هدف
- تاریخ نوشتن
- فایل های استفاده شده
- نام برنامه
- یادداشتها (ویژگیها)
- چگونگی استعمال در خط فرمان
- تاریخ بازنویسی



Debugging options for shell scripts

- -n

- این گزینه باعث می شود که پوسته تمام فرمانهای اسکریپت را بخواند اما آنها را اجرا نکند.

- -v

- این گزینه باعث می شود که پوسته تمام فرمانهای خوانده شده را نمایش دهد.

- -x

- تمام فرمانهای را به همراه آرگومانهای آنها که اجرا می شوند را نمایش می دهد (shell tracing).



Debugging options for shell scripts

- این گزینه های خاص پوسته با استفاده از فرمان `set -option` فعال و با `set +option` غیرفعال می شوند.
- ضمناً می توان قسمتی از اسکریپت را نیز خطایابی نمود، به این صورت که آن قسمت را به شکل زیر مشخص می کنیم:
`set -x ; BuggyPart ; set +x BuggyPart`
- تست کردن مقدار متغیرها نیز فقط با فرمان `echo` ممکن است. بنابراین در جایی که حدس می زنیم مشکلی وجود دارد باید مقدار متغیرها را نمایش داد و رفع مشکل نمود.



Debugging options for shell scripts

• خلاصه

```
$ bash -option script arg1 arg2 ... argN
```

و یا

```
$ cat script
```

```
#!/bin/bash -option
```

```
...
```

```
...
```

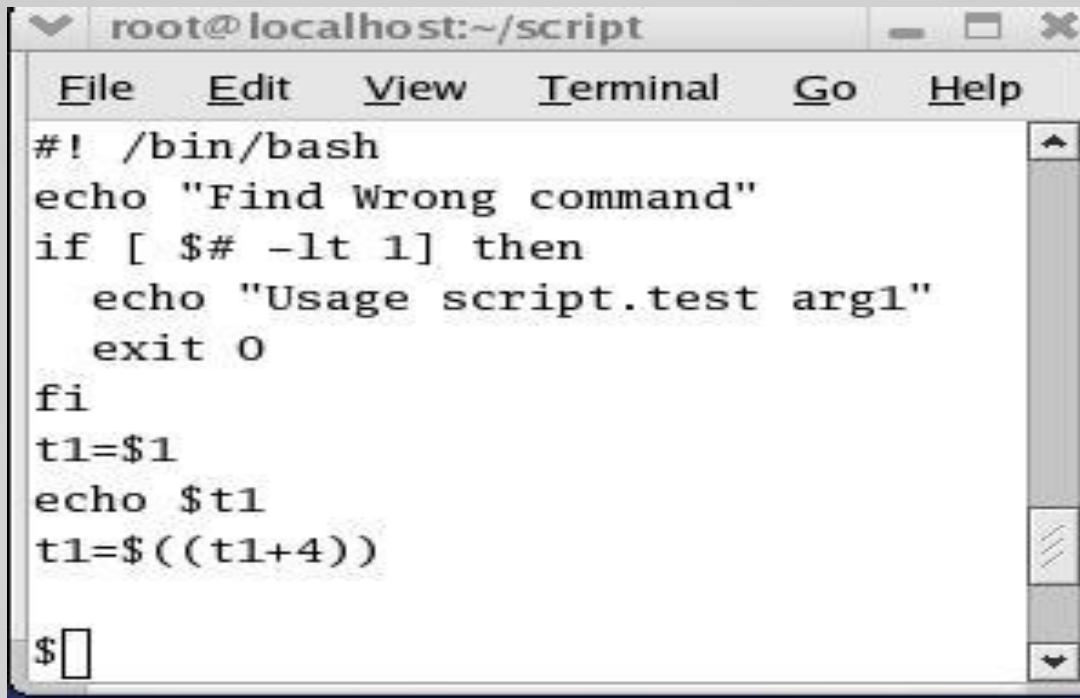
```
$
```

فعال کردن گزینه های خاص



مثال

- پیدا کردن خطای syntax ، در صورت وجود، اسکریپت زیر:



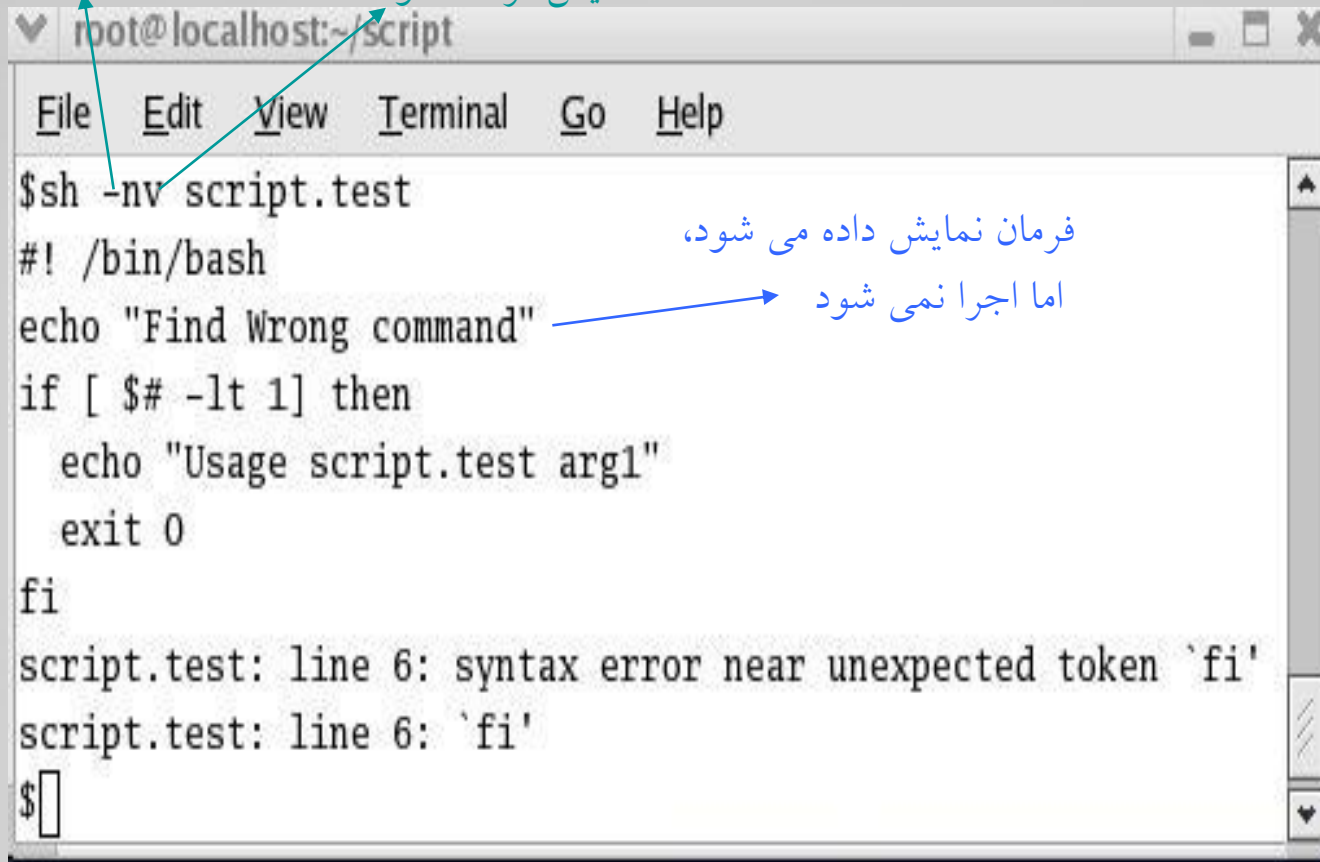
```
root@localhost:~/script
File Edit View Terminal Go Help
#!/bin/bash
echo "Find Wrong command"
if [ $# -lt 1] then
    echo "Usage script.test arg1"
    exit 0
fi
t1=$1
echo $t1
t1=$((t1+4))

$
```



مثال (ادامه)

نمایش فرمان خوانده شده چک کردن **syntax**



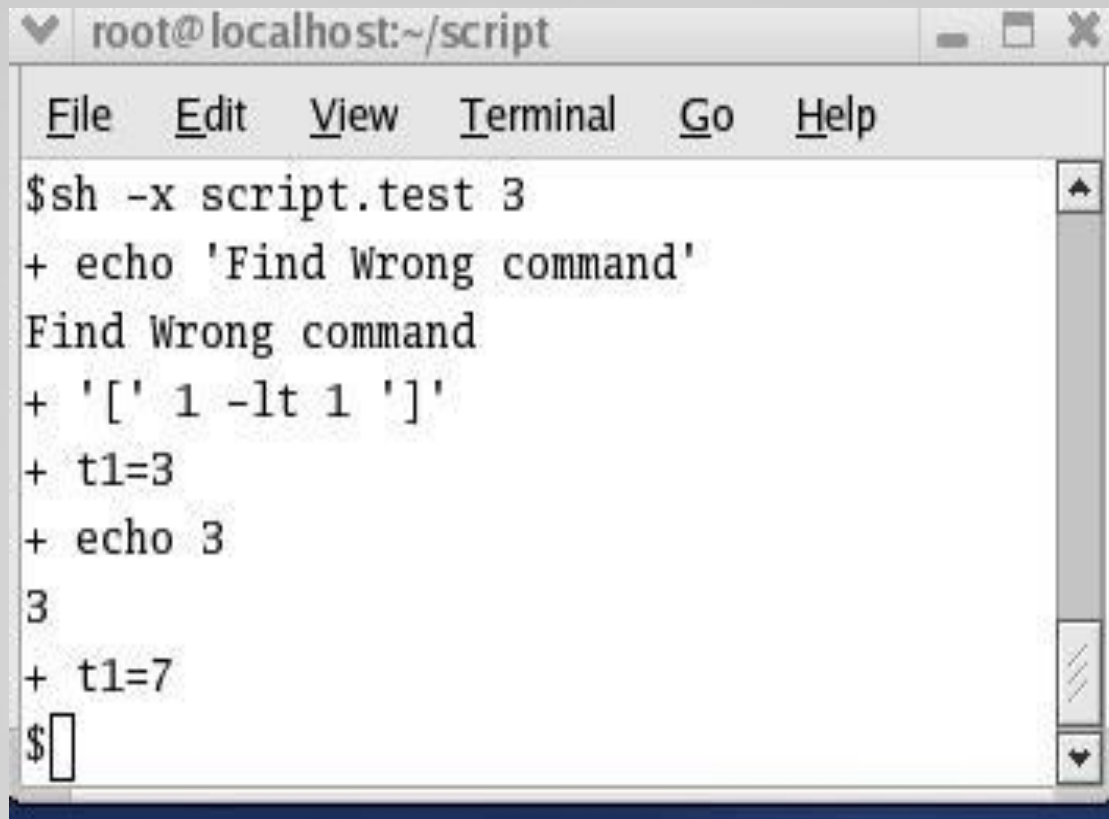
```
root@localhost:~/script
File Edit View Terminal Go Help
$sh -nv script.test
#!/bin/bash
echo "Find Wrong command"
if [ $# -lt 1 ] then
    echo "Usage script.test arg1"
    exit 0
fi
script.test: line 6: syntax error near unexpected token `fi'
script.test: line 6: `fi'
$
```

فرمان نمایش داده می شود،
اما اجرا نمی شود



مثال (ادامه)

• trace کردن اسکریپت



```
root@localhost:~/script
File Edit View Terminal Go Help
$sh -x script.test 3
+ echo 'Find Wrong command'
Find Wrong command
+ '[' 1 -lt 1 ']'
+ t1=3
+ echo 3
3
+ t1=7
$
```



مثال

- trace کردن قسمتی که حدس می زنیم مشکل دارد:

```

root@localhost:~/script
File Edit View Terminal Go Help
$ cat script.test
#!/bin/bash
echo "Find Wrong command"
if [ $# -lt 1 ] ; then
    echo "Usage script.test arg1"
    exit 0
fi

#I think that This part malfunction
set -x
t1=$1
echo $t1
t1=$((t1+4))
set +x

$ ./script.test 4
Find Wrong command
+ t1=4
+ echo 4
4
+ t1=8
+ set +x
$
  
```

قرار دادن قسمت

مورد نظر در بین

set -x

set +x





کودکان



خبرنامه



مقالات



فایل آموزشی



پرسش و
پاسخ



بسته آموزشی



فرصت های
شغلی



فیلم آموزشی



آموزش ایمیلی



باشگاه
دانشجویان



لینوکس



جارا



شبکه



توریسم



توسعه وب



گرافیک



امنیت



پایگاه داده

