

گزارش دستور کار سوم آزمایشگاه سیستم‌های عامل

نگار موقتیان، ۹۸۳۱۰۶۲

۱. جمع دو عدد ورودی، محاسبه عدد بزرگ‌تر و چاپ راهنما در صورت نامعتبر بودن اعداد

در این قسمت از آزمایش یک اسکریپت ساده می‌نویسیم که دو ورودی را به عنوان آرگومان دریافت می‌کند. سپس بررسی می‌کند ورودی‌های داده شده اعداد صحیح هستند یا خیر. در صورتی که ورودی‌های داده شده اعداد صحیح نبودند پیغام خطای مناسب چاپ می‌شود، در غیر این صورت ابتدا حاصل جمع دو عدد و پس از آن با مقایسه دو عدد، عدد بزرگ‌تر چاپ می‌شود.

```
#!/bin/bash

if [[ $1 =~ ^-?[0-9]+$ ]] && [[ $2 =~ ^-?[0-9]+$ ]] ; then
    echo "Sum: $((($1+$2))";
    if [ $2 -gt $1 ] ; then
        echo "Max: $2"
    else
        echo "Max: $1"
    fi
else
    echo "Invalid Input: Please enter two integer numbers as argument.";
fi
```

* در ابتدای تمام برنامه‌های نوشته شده در این آزمایش دستور `#!/bin/bash` را قرار می‌دهیم تا برنامه توسط `bash` تفسیر شود.

به علاوه در این اسکریپت با استفاده از `regex` فرمت آرگومان‌های ورودی را بررسی می‌کنیم. الگوی این `regex` مانند زیر می‌باشد:

`^-?[0-9]+$`

که در آن `^` ابتدای الگو را مشخص می‌کند. پس از آن می‌تواند یک علامت `-` وجود داشته باشد (در صورتی که عدد منفی باشد) و به دنبال آن باید یک یا بیش‌تر عدد بین `۰` تا `۹` بیاید. در نهایت نیز `$` انتهای الگو را مشخص می‌کند.

برای آشنایی با نحوه استفاده از `regex` در `bash script` از لینک زیر استفاده شده است:

<https://7thzero.com/blog/bash-101-part-5-regular-expressions-in-conditional-statements>

نمونه ورودی و خروجی این برنامه به صورت زیر می باشد:

```
OSLab03_E1.sh
~/Desktop/OS Lab/Lab03

#!/bin/bash

if [[ $1 =~ ^-?[0-9]+$ ]] || [[ $2 =~ ^-?[0-9]+$ ]] || [[ $1 == $2 ]]; then
    echo "Sum: $(( $1 + $2 ))"
    echo "Max: $(( ${1#$-} ))"
else
    echo "Invalid Input: Please enter two integer numbers as argument."
fi
```

۲. طراحی ماشین حساب با استفاده از case

در این قسمت از آزمایش می خواهیم یک ماشین حساب با ۴ عمل اصلی طراحی کنیم.

```
#!/bin/bash

case $2 in
    '+')
        echo "$1 + $3 = $(( $1 + $3 ))"
        ;;
    '-')
        echo "$1 - $3 = $(( $1 - $3 ))"
        ;;
    'x')
        echo "$1 x $3 = $(( $1 * $3 ))"
        ;;
    '/')
        echo "$1 / $3 = $(( $1 / $3 ))"
        ;;
    *)
        echo "Invalid Operation!"
        ;;
esac
```

در این برنامه سه آرگومان به عنوان ورودی گرفته می‌شود. آرگومان اول (\$1) عدد اول، آرگومان دوم (\$2) عملیاتی که باید انجام شود و آرگومان سوم (\$3) عدد دوم می‌باشد. بنابراین switch بر روی آرگومان دوم زده شده است. پس از آن به تناسب عملیاتی که باید انجام شود نتیجه این عملیات را چاپ کرده‌ایم.

نکته‌ای که در این جا وجود دارد این است که کاراکتر * در bash معنای خاصی دارد و به عنوان یک wildcard برای رشته‌ها استفاده می‌شود. بنابراین برای دستور ضرب بجای * از کاراکتر x استفاده شده و در انتها برای بیان حالت default (هر آنچه جز حالات بالا بود) از * استفاده شده است.

نمونه ورودی و خروجی این برنامه به صورت زیر می‌باشد:

```

OSLab03_E2.sh
~/Desktop/OS Lab/Lab03

#!/bin/bash

case $2 in
    '+')
        $3 + $4
    ;;
    '-')
        $3 - $4
    ;;
    'x')
        $3 x $4
    ;;
    '/')
        $3 / $4
    ;;
    *)
        Invalid Operation!
    ;;
esac

maxeu@ubuntu: ~/Desktop/OS Lab/Lab03
File Edit View Search Terminal Help
maxeu@ubuntu:~/Desktop/OS Lab/Lab03$ bash OSLab03_E2.sh 1 + 2
1 + 2 = 3
maxeu@ubuntu:~/Desktop/OS Lab/Lab03$ bash OSLab03_E2.sh 13 - 7
13 - 7 = 6
maxeu@ubuntu:~/Desktop/OS Lab/Lab03$ bash OSLab03_E2.sh 21 x 4
21 x 4 = 84
maxeu@ubuntu:~/Desktop/OS Lab/Lab03$ bash OSLab03_E2.sh 52 / 4
52 / 4 = 13
maxeu@ubuntu:~/Desktop/OS Lab/Lab03$ bash OSLab03_E2.sh 45 a 3
Invalid Operation!
maxeu@ubuntu:~/Desktop/OS Lab/Lab03$

```

۳. معکوس کردن عدد ورودی و چاپ مجموع ارقام آن

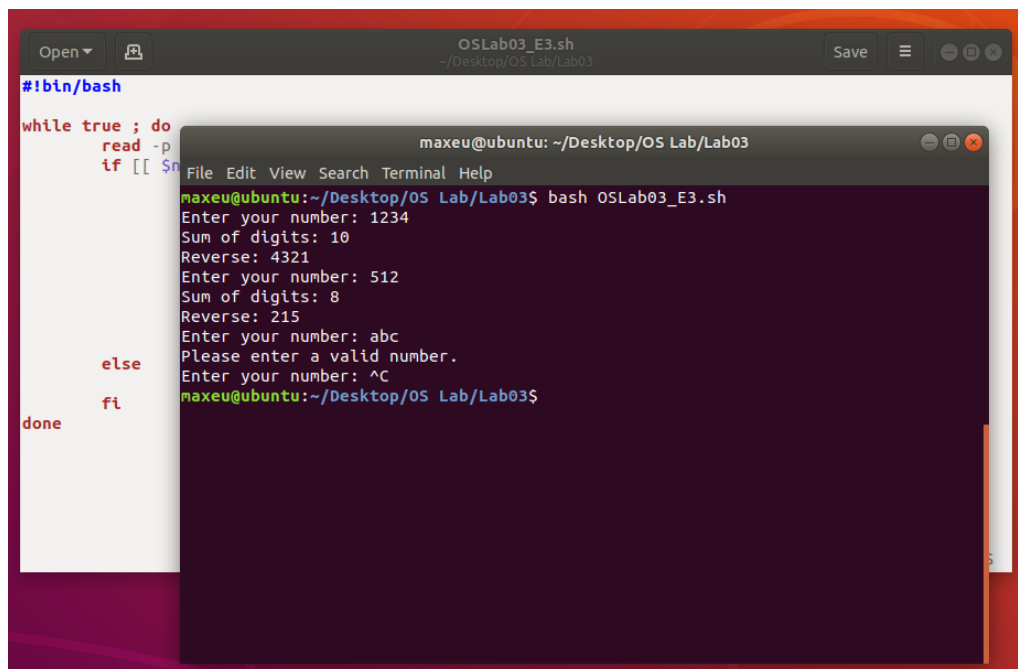
در این قسمت از آزمایش می‌خواهیم به طور متوالی اعدادی را از کاربر دریافت کنیم. سپس معکوس عدد و جمع ارقامش را چاپ کنیم. برای دریافت متوالی اعداد از یک while true استفاده می‌کنیم. این حلقه همواره تکرار خواهد شد زیرا شرط آن همواره برقرار است. پس از آن دقیقاً مانند آزمایش اول ورودی را بررسی می‌کنیم تا کاربر به جای عدد حروف وارد نکند (البته این بار تنها اعداد مثبت را دریافت می‌کنیم و شرط -؟ ابتدایی را حذف می‌کنیم). در صورت نامعتبر بودن عدد پیغامی چاپ می‌کنیم و در غیر این صورت در یک حلقه ارقام عدد را با استفاده از باقی‌مانده گرفتن بر ۱۰ جدا می‌کنیم. از این باقی‌مانده (که نمایانگر رقم یکان عدد در هر مرحله است)

برای محاسبه مجموع ارقام و ساخت عدد معکوس مطابق زیر استفاده می‌کنیم. در انتهای هر مرحله نیز عدد را بر ۱۰ تقسیم می‌کنیم تا در مرحله بعد بتوانیم رقم بعدی را جدا کنیم. این کار را تا زمانی ادامه می‌دهیم که عدد از ۱ کوچک‌تر شود، و این یعنی تمام ارقام آن را بررسی کرده‌ایم.

```
#!/bin/bash

while true ; do
    read -p 'Enter your number: ' number
    if [[ $number =~ ^[0-9]+$ ]] ; then
        sum=0
        reverse=0
        while [ $number -ge 1 ] ; do
            sum=$((sum+(number%10)))
            reverse=$((reverse*10)+(number%10))
            number=$((number/10))
        done
        echo "Sum of digits: $sum"
        echo "Reverse: $reverse"
    else
        echo "Please enter a valid number."
    fi
done
```

نمونه ورودی و خروجی این برنامه به صورت زیر می‌باشد (با استفاده از `ctrl + c` از برنامه خارج شده‌ایم):



```
OSLab03_E3.sh
~/Desktop/OS Lab/Lab03

#!/bin/bash

while true ; do
    read -p 'Enter your number: ' number
    if [[ $number =~ ^[0-9]+$ ]] ; then
        sum=0
        reverse=0
        while [ $number -ge 1 ] ; do
            sum=$((sum+(number%10)))
            reverse=$((reverse*10)+(number%10))
            number=$((number/10))
        done
        echo "Sum of digits: $sum"
        echo "Reverse: $reverse"
    else
        echo "Please enter a valid number."
    fi
done

maxeu@ubuntu: ~/Desktop/OS Lab/Lab03
maxeu@ubuntu:~/Desktop/OS Lab/Lab03$ bash OSLab03_E3.sh
Enter your number: 1234
Sum of digits: 10
Reverse: 4321
Enter your number: 512
Sum of digits: 8
Reverse: 215
Enter your number: abc
Please enter a valid number.
Enter your number: ^C
maxeu@ubuntu:~/Desktop/OS Lab/Lab03$
```

۴. دریافت دو عدد x و y در ورودی، دریافت نام یک فایل و نمایش خط x تا y فایل مورد نظر

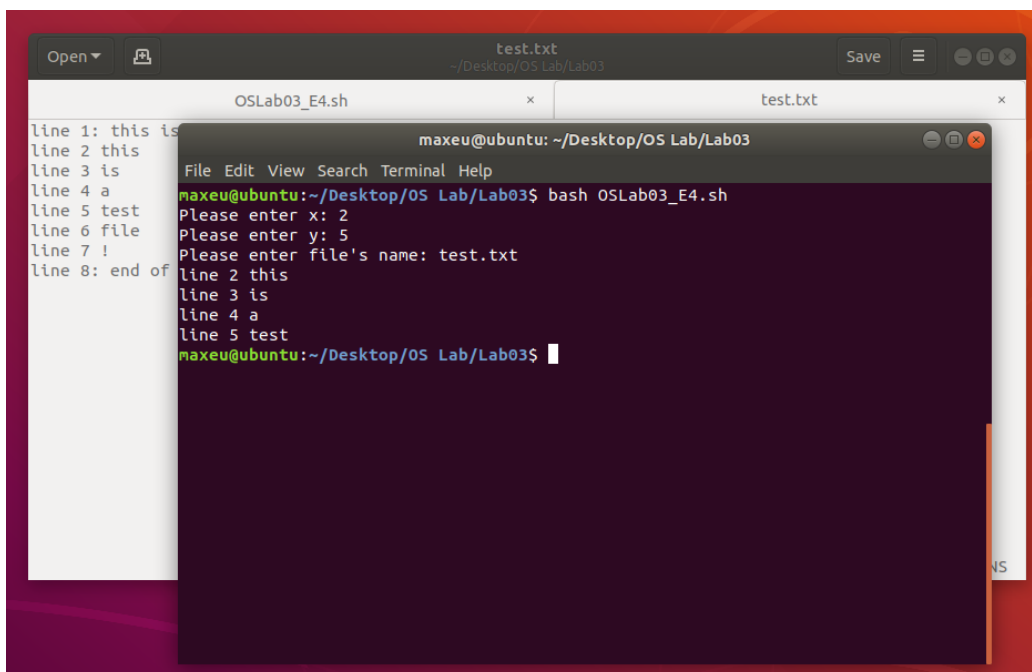
در این قسمت از آزمایش می‌خواهیم دو عدد x و y را از ورودی دریافت کرده، نام یک فایل را دریافت کنیم و سپس خط x تا y ام فایل را نمایش دهیم.

```
#!/bin/bash

read -p 'Please enter x: ' x
read -p 'Please enter y: ' y
read -p "Please enter file's name: " name
count=1
while read -r line; do
    if [ $count -ge $x ] && [ $count -le $y ] ; then
        echo $line
    fi
    count=$((count+1))
done < $name
```

در ابتدا ورودی‌ها را دریافت کرده‌ایم. سپس از یک حلقه `while` استفاده می‌کنیم تا تمام خطوط فایل با اسم داده شده را تا رسیدن به انتهای فایل به ترتیب بخوانیم. پیش از ورود به این حلقه یک شمارنده را با صفر مقداردهی می‌کنیم و با خواندن هر خط از فایل مورد نظر این شمارنده را افزایش می‌دهیم. اگر این شمارنده عددی بین x و y بود خط خوانده شده از برنامه را چاپ می‌کنیم.

نمونه ورودی و خروجی این برنامه به صورت زیر می‌باشد:



```
maxeu@ubuntu: ~/Desktop/OS Lab/Lab03
File Edit View Search Terminal Help
maxeu@ubuntu:~/Desktop/OS Lab/Lab03$ bash OSLab03_E4.sh
Please enter x: 2
Please enter y: 5
Please enter file's name: test.txt
line 2 this
line 3 is
line 4 a
line 5 test
maxeu@ubuntu:~/Desktop/OS Lab/Lab03$
```

همچنین برای آشنایی با نحوه خواندن خط به خط یک فایل متنی در bash script از سایت زیر استفاده شده است:

<https://stackoverflow.com/questions/10929453/read-a-file-line-by-line-assigning-the-value-to-a-variable>

۵. رسم شکل مناسب با توجه به عدد دریافت شده از کاربر

در این قسمت از آزمایش در ابتدا یک عدد از کاربر دریافت می‌کنیم که الگوی شکلی که قرار است رسم شود را مشخص می‌کند. پس از آن نیز عدد n را دریافت می‌کنیم که مشخص می‌کند شکل مورد نظر چند سطر دارد. سپس با استفاده از دستور switch روی عدد اول شکل مورد نظر را رسم می‌کنیم.

برای رسم هر یک از شکل‌ها نیاز به دو حلقه تو در تو داریم. برای آشنایی با syntax حلقه for به صورتی که بازه آن وابسته به یک متغیر باشد (در اینجا از 1 تا n یا 0 تا $i-1$) از لینک زیر استفاده شده است:

<https://www.cyberciti.biz/faq/bash-for-loop/>

توضیح کوتاهی در رابطه با نحوه رسم هر شکل در ادامه آمده است:

۱. برای رسم شکل اول یک حلقه بیرونی به اندازه n داریم (n سطر شکل). در سطر i ام به اندازه i بار عدد i را چاپ می‌کنیم، بنابراین حلقه دوم باید از 0 تا $i-1$ حرکت کند و هر بار i را چاپ کند.
۲. شکل دوم را می‌توان به یک مثلث که قائده آن پایین، و یک مثلث که قاعده آن بالا قرار دارد تقسیم کرد. در هر قسمت ابتدا به تعداد مناسب کاراکتر space را چاپ کرده و سپس به تعداد مناسب ستاره چاپ می‌کنیم. در قسمت اول در هر مرحله یکی به تعداد ستاره‌ها افزوده و یکی از space ها کم می‌کنیم، در قسمت دوم نیز در هر مرحله یکی از تعداد ستاره‌ها کم کرده و یکی به space ها اضافه می‌کنیم.
۳. در شکل سوم در هر مرحله به تعداد یکی کمتر از شماره سطر شکل کاراکتر ' | ' را چاپ کرده و در نهایت نیز یک ' _ ' چاپ می‌کنیم.

```

#!/bin/bash

read -p "Pattern's number: " pn
read -p "n: " n

case $pn in
    '1')
        for (( i=1; i<=n; i++ )) ; do
            for (( j=0; j<i; j++ )) ; do
                echo -n $i
            done
            echo ''
        done
        ;;
    '2')
        for (( i=1; i<=n; i++ )) ; do
            for (( j=0; j<n-i; j++ )) ; do
                echo -n ' '
            done
            for (( j=0; j<i; j++ )) ; do
                echo -n '*'
            done
            echo ''
        done
        for (( i=n; i>=1; i-- )) ; do
            for (( j=0; j<n-i; j++ )) ; do
                echo -n ' '
            done
            for (( j=0; j<i; j++ )) ; do
                echo -n '*'
            done
            echo ''
        done
        ;;
    '3')
        for (( i=1; i<=n; i++ )) ; do
            for (( j=1; j<i; j++ )) ; do
                echo -n "| "
            done
            echo '|_'
        done
        ;;
esac

```

نمونه ورودی و خروجی این برنامه به صورت زیر می باشد:

shell به کار می‌رود و standard output یک دستور را به standard input دستور دیگر متصل می‌نماید. در کد زیر از همین قابلیت استفاده شده است. برای مثال در دستور زیر:

```
echo "$1 + $3" | bc
```

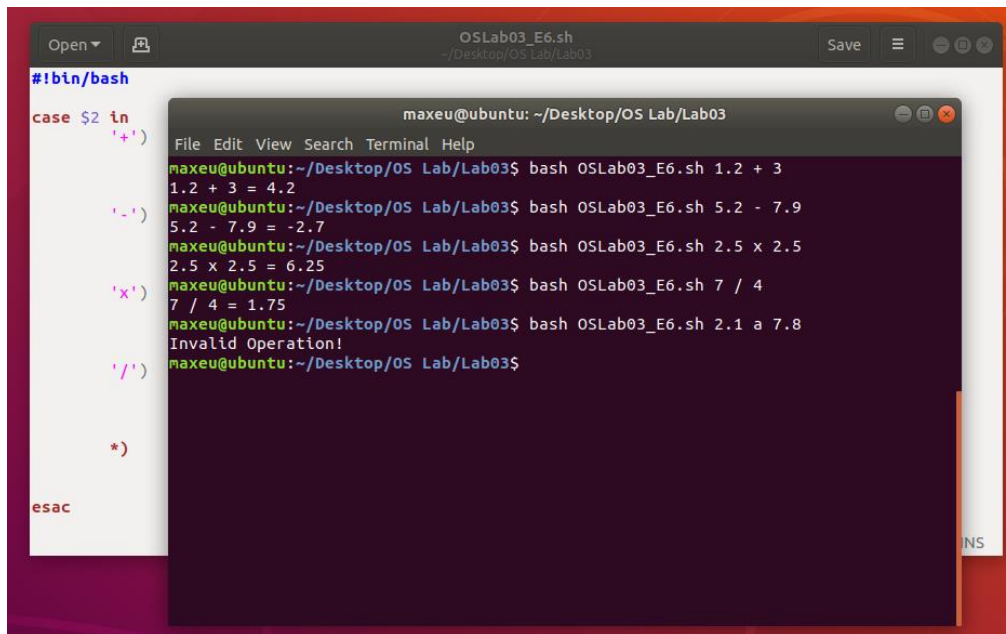
با استفاده از | عبارت "\$1 + \$3" به ورودی BC داده شده، عملیات مربوطه بر روی آن انجام گرفته، BC مقدار حاصل جمع را برگردانده و در نهایت مقدار حاصل توسط echo چاپ می‌شود.

در قسمت مربوط به تقسیم علاوه بر خود عملیات یک مقدار scale نیز به BC داده شده است. این مقدار مشخص می‌کند محاسبات تا چند رقم اعشار انجام شوند. همچنین یک آرگومان -1 به آن داده شده است که یک کتابخانه استاندارد برای اعمال ریاضی را لود می‌کند.

```
#!/bin/bash

case $2 in
    '+')
        echo -n "$1 + $3 = "
        echo "$1 + $3" | bc
        ;;
    '-')
        echo -n "$1 - $3 = "
        echo "$1 - $3" | bc
        ;;
    'x')
        echo -n "$1 x $3 = "
        echo "$1 * $3" | bc -l
        ;;
    '/')
        echo -n "$1 / $3 = "
        echo "result = $1 / $3; scale = 2; result / 1" | bc -l
        ;;
    *)
        echo "Invalid Operation!"
        ;;
esac
```

نمونه ورودی و خروجی این برنامه به صورت زیر می‌باشد:



```
OSLab03_E6.sh
~/Desktop/OS Lab/Lab03

#!bin/bash

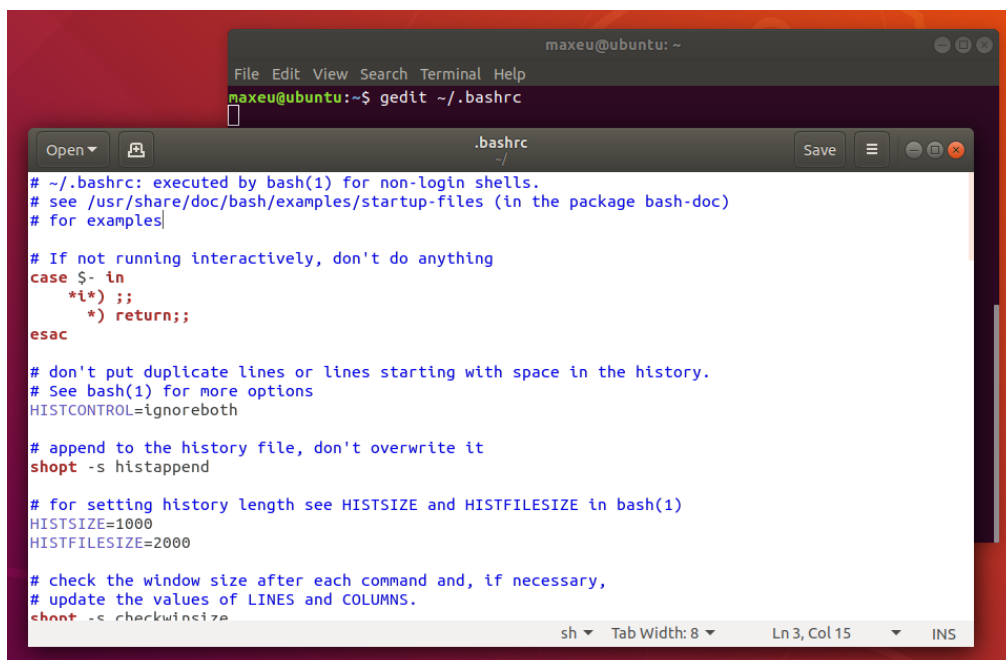
case $2 in
    '+')
        maxeu@ubuntu:~/Desktop/OS Lab/Lab03$ bash OSLab03_E6.sh 1.2 + 3
        1.2 + 3 = 4.2
    '-' )
        maxeu@ubuntu:~/Desktop/OS Lab/Lab03$ bash OSLab03_E6.sh 5.2 - 7.9
        5.2 - 7.9 = -2.7
    'x')
        maxeu@ubuntu:~/Desktop/OS Lab/Lab03$ bash OSLab03_E6.sh 2.5 x 2.5
        2.5 x 2.5 = 6.25
    '/' )
        maxeu@ubuntu:~/Desktop/OS Lab/Lab03$ bash OSLab03_E6.sh 7 / 4
        7 / 4 = 1.75
    'a')
        maxeu@ubuntu:~/Desktop/OS Lab/Lab03$ bash OSLab03_E6.sh 2.1 a 7.8
        Invalid Operation!
    *)
        maxeu@ubuntu:~/Desktop/OS Lab/Lab03$
esac
```

۷. افزودن command جدید به bash

برای انجام این کار باید دستور مربوطه را به فایل `bashrc` اضافه کنیم، بنابراین ابتدا با استفاده از دستور:

```
gedit ~/.bashrc
```

این فایل را با استفاده از text editor باز می‌کنیم تا بتوانیم آن را تغییر دهیم.



```
maxeu@ubuntu: ~
File Edit View Search Terminal Help
maxeu@ubuntu:~$ gedit ~/.bashrc

.bashrc
# ~/.bashrc: executed by bash(1) for non-login shells.
# see /usr/share/doc/bash/examples/startup-files (in the package bash-doc)
# for examples

# If not running interactively, don't do anything
case $- in
    *l*) ;;
    *) return;;
esac

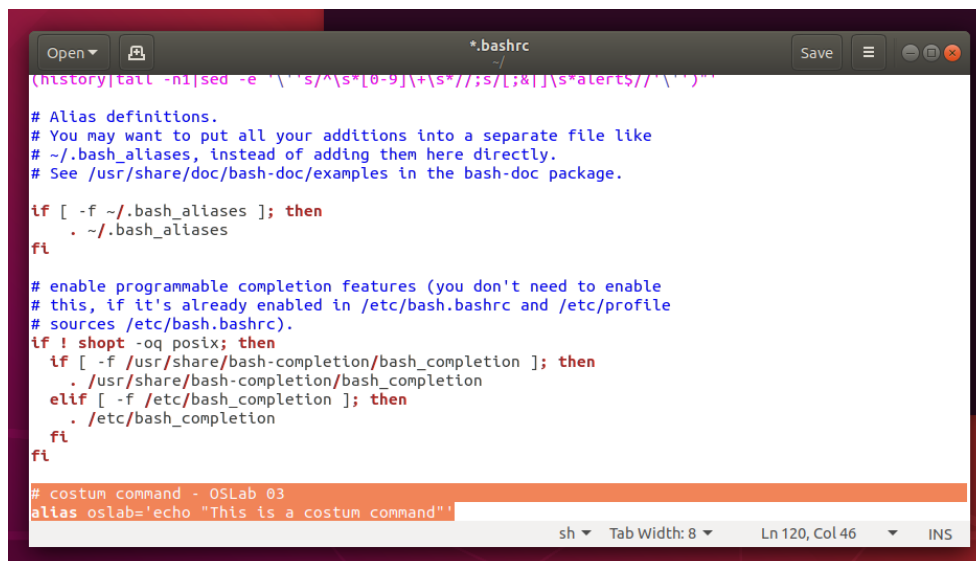
# don't put duplicate lines or lines starting with space in the history.
# See bash(1) for more options
HISTCONTROL=ignoreboth

# append to the history file, don't overwrite it
shopt -s histappend

# for setting history length see HISTSIZE and HISTFILESIZE in bash(1)
HISTSIZE=1000
HISTFILESIZE=2000

# check the window size after each command and, if necessary,
# update the values of LINES and COLUMNS.
shopt -s checkwinsize
```

حال تنها کافیه یک alias جدید برای آن تعریف کنیم. با این کار هر زمان که دستور oslab را تایپ کنیم دستور 'This is a custom command' echo اجرا شده و این عبارت چاپ می‌شود. این alias مانند شکل زیر (خطوط highlight شده) به انتهای فایل .bashrc اضافه شده است.



```
Open | Save | [Icons] | *bashrc
(history|tail -n|sed -e '\s/\s*[0-9]\{+\}\s*//;s/[:&|]\s*alert$//'\s*)

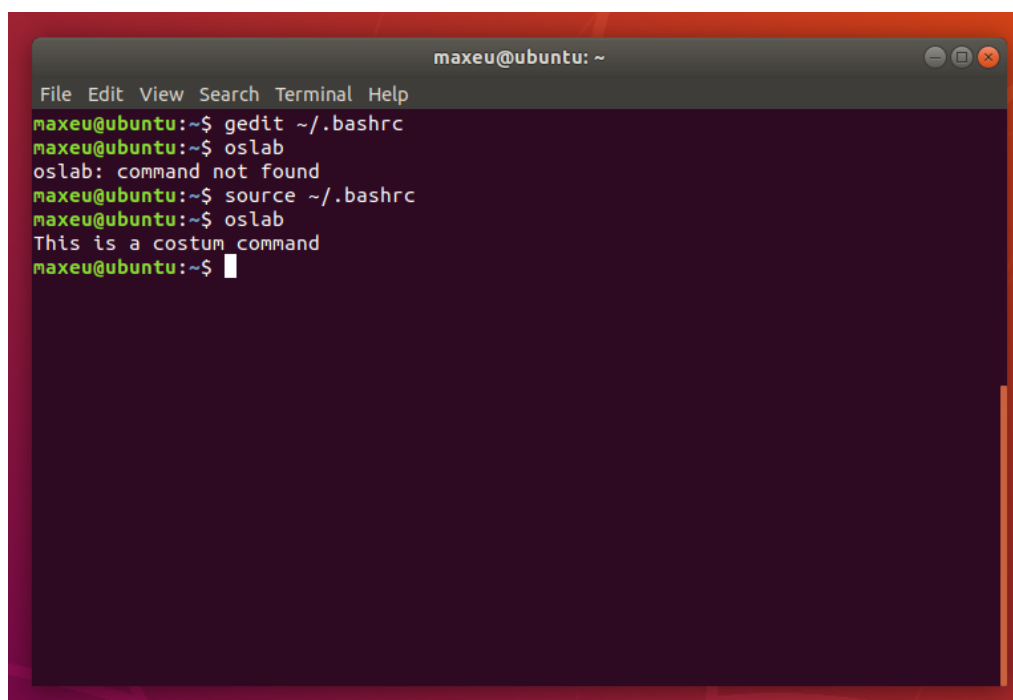
# Alias definitions.
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
    if [ -f /usr/share/bash-completion/bash_completion ]; then
        . /usr/share/bash-completion/bash_completion
    elif [ -f /etc/bash_completion ]; then
        . /etc/bash_completion
    fi
fi

# costum command - Oslab 03
alias oslab='echo "This is a costum command"'
sh Tab Width: 8 Ln 120, Col 46 INS
```

این فایل را save کرده و به ترمینال باز می‌گردیم. دستور oslab را داخل ترمینال تایپ می‌کنیم. این دستور ابتدا شناخته نمی‌شود، بنابراین با استفاده از دستور source ~/.bashrc مطمئن می‌شویم که فایل .bashrc آپدیت و لود شده است. حال بار دیگر دستور oslab را اجرا می‌کنیم و مشاهده می‌شود همانطور که انتظار داشتیم عبارت This is a custom command چاپ می‌شود.



```
maxeu@ubuntu: ~
File Edit View Search Terminal Help
maxeu@ubuntu:~$ gedit ~/.bashrc
maxeu@ubuntu:~$ oslab
oslab: command not found
maxeu@ubuntu:~$ source ~/.bashrc
maxeu@ubuntu:~$ oslab
This is a costum command
maxeu@ubuntu:~$
```

برای انجام این قسمت از آزمایش از لینک زیر کمک گرفته شده است:

<https://dev.to/mollynem/4-simple-steps-for-custom-bash-commands-4c58>

سوالات داخل متن دستور کار

* تمام اسکریپت‌های نوشته شده در این قسمت در فایل OSLab03_etc.sh موجود است.

۱. مقدار سایر متغیرهای خاص را بیابید:

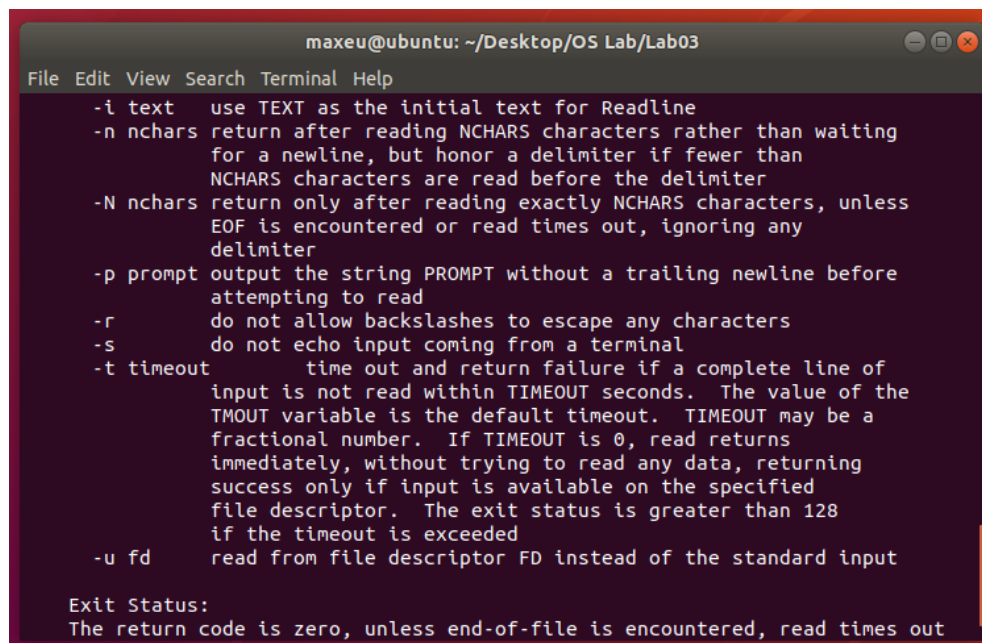
Variable	Description
\$0	The filename of the current script.
\$n	These variables correspond to the arguments with which a script was invoked. Here n is a positive decimal number corresponding to the position of an argument (the first argument is \$1, the second argument is \$2, and so on).
\$\$	The process ID of the current shell. For shell scripts, this is the process ID under which they are executing.
\$#	The number of arguments supplied to a script.
\$@	All the arguments are individually double quoted. If a script receives two arguments, \$@ is equivalent to \$1 \$2.
\$*	All the arguments are double quoted. If a script receives two arguments, \$* is equivalent to \$1 \$2.
\$?	The exit status of the last command executed.
\$_	The process ID of the last background command.
\$!	The last argument of the previous command.
\$USER	Current user name.

۲. اگر بیش از ۱۰ آرگومان ورودی باشد، چگونه باید به مقدار دهمین آرگومان دست یافت؟
کافیست از {} دو طرف شماره آرگومان استفاده کنیم. برای مثال:

```
echo "${10}"
```

دهمین آرگومان ورودی را چاپ می کند.

۳. -p و -sp چه امکانی را فراهم می کنند؟ مقدار متغیرهای uservar و passvar را در فایلی ذخیره کنید.
اگر داخل ترمینال دستور --help read را اجرا کنیم راهنمای زیر نشان داده می شود:



```
maxeu@ubuntu: ~/Desktop/OS Lab/Lab03
File Edit View Search Terminal Help
-i text    use TEXT as the initial text for Readline
-n nchars  return after reading NCHARS characters rather than waiting
           for a newline, but honor a delimiter if fewer than
           NCHARS characters are read before the delimiter
-N nchars  return only after reading exactly NCHARS characters, unless
           EOF is encountered or read times out, ignoring any
           delimiter
-p prompt  output the string PROMPT without a trailing newline before
           attempting to read
-r         do not allow backslashes to escape any characters
-s         do not echo input coming from a terminal
-t timeout time out and return failure if a complete line of
           input is not read within TIMEOUT seconds. The value of the
           TMOUT variable is the default timeout. TIMEOUT may be a
           fractional number. If TIMEOUT is 0, read returns
           immediately, without trying to read any data, returning
           success only if input is available on the specified
           file descriptor. The exit status is greater than 128
           if the timeout is exceeded
-u fd      read from file descriptor FD instead of the standard input

Exit Status:
The return code is zero, unless end-of-file is encountered, read times out
```

طبق این راهنما آرگومان -p برای خواندن ورودی از کاربر و ذخیره آن در یک متغیر استفاده می شود، به صورتی که برنامه برای دریافت ورودی به خط جدید نمی رود. همچنین با استفاده از آرگومان -s ورودی کاربر بر روی ترمینال نمایش داده نمی شود، به همین دلیل این آرگومان در گرفتن پسوردها کاربرد دارد.
اسکرپت زیر دو رشته username و password را از کاربر گرفته و آن را در یک فایل به نام info.txt ذخیره می کند.

```
read -p 'Username: ' username
echo "Username: $username" > info.txt
read -sp 'Password: ' password
echo ''
echo "Password: $password" >> info.txt
```

برای ذخیره سازی username از >> استفاده شده است تا محتویات فایل قبلی پاک شود و username جدید در آن قرار گیرد اما در گرفتن password از > استفاده شده تا مقدار password پس از مقداری که برای username نوشته ایم ذخیره شود.

۴. برای انجام محاسبات شیوه های مختلفی وجود دارد. به مثال های زیر توجه کنید. کد را اجرا کنید و نتیجه را گزارش کنید.

```
let a=10+8
echo $a
expr 5 \* 4
expr 5 / 4
expr 11 % 2
a=$( expr 10 - 3 )
echo $a
b=$(( a + 3 ))
echo $b
((b++))
echo $b
```

دستور expr برای انجام عملیات ریاضی و چاپ نتیجه آن ها استفاده می شود.

۱. در خط اول مقدار a با استفاده از دستور let برابر با ۱۸ قرار داده شده، بنابراین در خط بعدی مقدار ۱۸ چاپ می شود.

۲. پس از آن مقدار $20 = 5 \times 4$ چاپ می شود.

۳. در خط بعدی بخش صحیح حاصل تقسیم ۵ بر ۴ یعنی ۱ چاپ می شود.

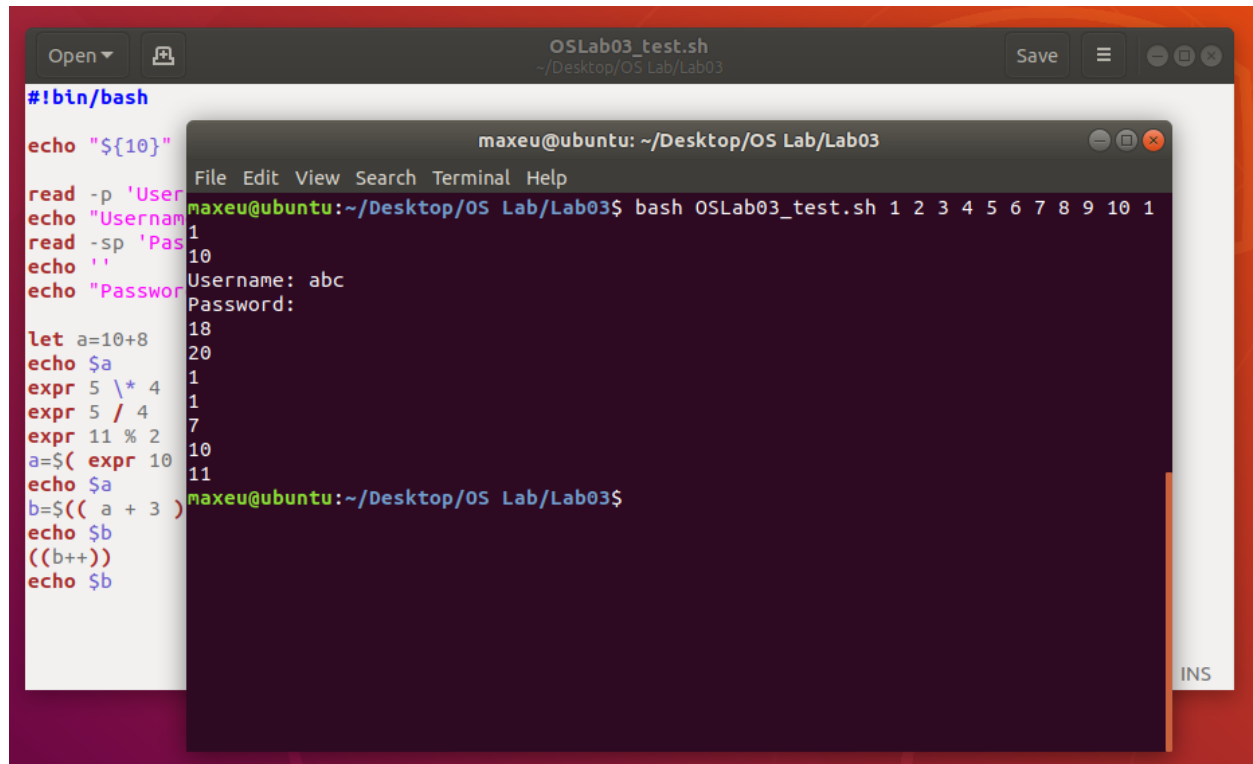
۴. در خط بعد باقی مانده ۱۱ بر ۲ یعنی ۱ چاپ می شود.

۵. در خط بعد مقدار متغیر a برابر با مقدار عبارت $10 - 3 = 7$ قرار داده شده و در خط بعد این مقدار چاپ می شود.

۶. پس از آن مقدار b برابر با مقدار $a + 3$ یعنی $7 + 3 = 10$ قرار داده شده و در خط بعد این مقدار چاپ می شود.

۷. در نهایت توسط دستور $((b++))$ مقدار متغیر b یکی افزایش یافته و در خط بعد مقدار ۱۱ چاپ می شود.

نمونه ورودی و خروجی تمام آزمایش‌های این بخش در ادامه آمده‌است:



```
OSLab03_test.sh
~/Desktop/OS Lab/Lab03

#!/bin/bash
echo "${10}"
read -p 'Username: ' username
echo "Username: $username"
read -sp 'Password: ' password
echo "Password: $password"

let a=10+8
echo $a
expr 5 \* 4
expr 5 / 4
expr 11 % 2
a=$(( expr 10
echo $a
b=$(( a + 3 )
echo $b
((b++))
echo $b

maxeu@ubuntu: ~/Desktop/OS Lab/Lab03$ bash OSLab03_test.sh 1 2 3 4 5 6 7 8 9 10 1
1
10
Username: abc
Password:
18
20
1
1
1
7
10
11
maxeu@ubuntu:~/Desktop/OS Lab/Lab03$
```