**گزارش دستورکار سوم آزمایشگاه سیستم­های عامل**

نگار موقتیان، 9831062

1. **جمع دو عدد ورودی، محاسبۀ 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 مقایسۀ 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:

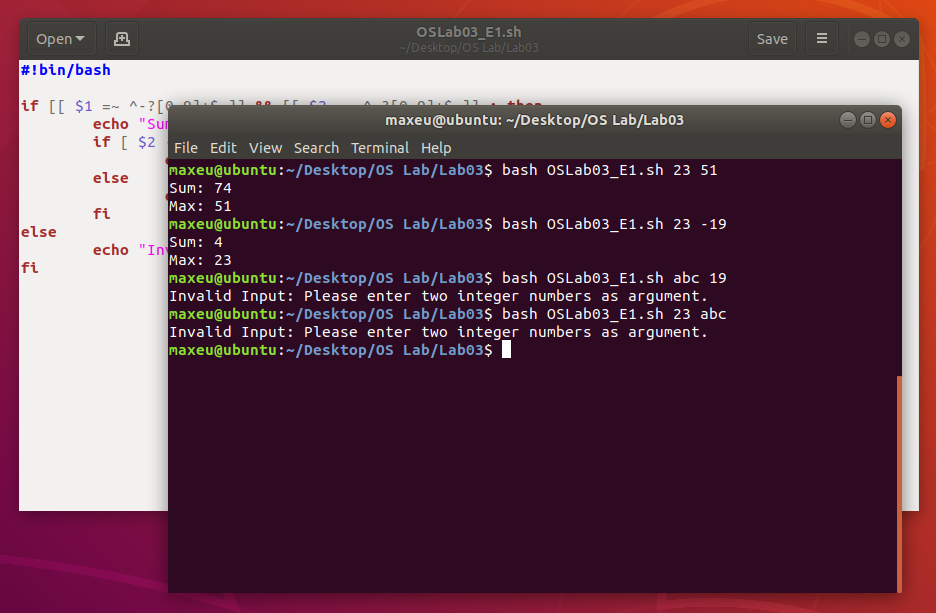
^-?[0-9]+$

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 0 تا 9 بیاید. در نهایت نیز $ انتهای الگو را مشخص می­کند.

برای آشنایی با نحوۀ 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>

نمونۀ ورودی و خروجی این برنامه به صورت زیر می­باشد:



1. **طراحی ماشین حساب با استفاده از case**

در این قسمت از آزمایش می­خواهیم یک ماشین حساب با 4 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!"

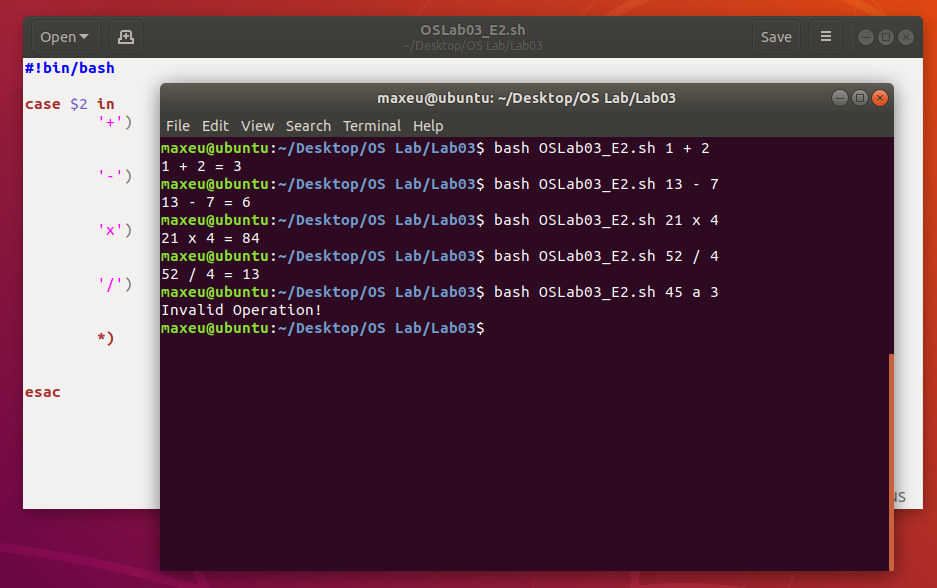
;;

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.

نمونۀ ورودی و خروجی این برنامه به صورت زیر می­باشد:



1. **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 10 جدا می­کنیم. از این باقی­مانده (که نمایانگر رقم یکان عدد در هر مرحله است) برای محاسبۀ مجموع ارقام و ساخت عدد معکوس مطابق زیر استفاده می­کنیم. در انتهای هر مرحله نیز عدد را بر 10 تقسیم می­کنیم تا در مرحلۀ بعد بتوانیم رقم بعدی را جدا کنیم. این کار را تا زمانی ادامه می­دهیم که عدد از 1 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"

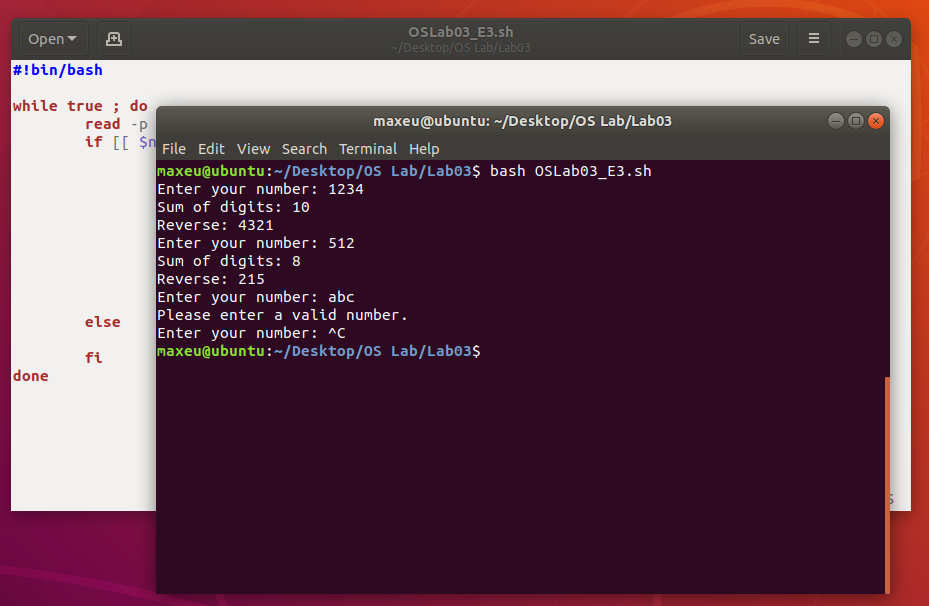
else

echo "Please enter a valid number."

fi

done

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



1. **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 ام 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

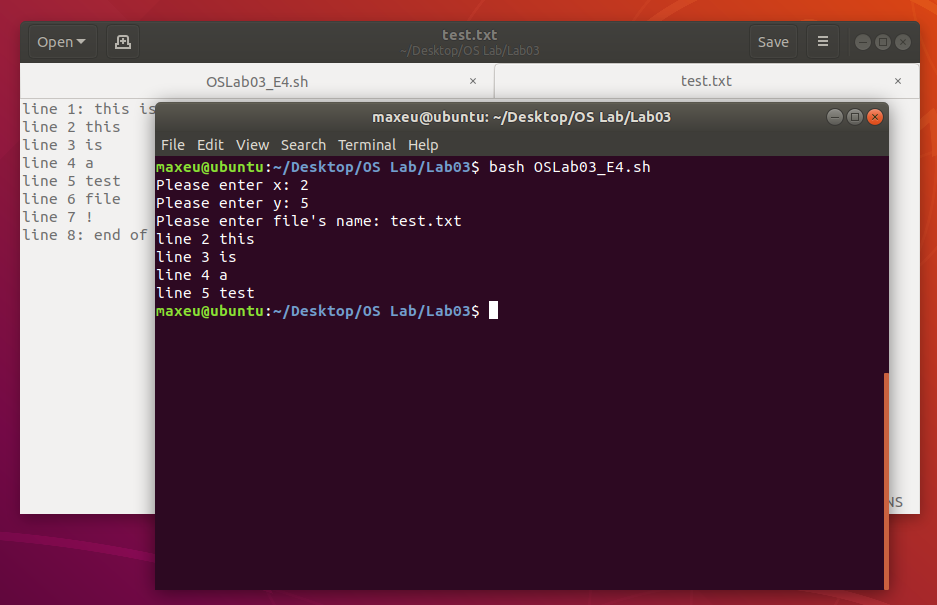
fi

count=$((count+1))

done < $name

در ابتدا ورودی­ها را دریافت کرده­ایم. سپس از یک حلقۀ 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 خواند 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.

نمونۀ ورودی و خروجی این برنامه به صورت زیر می­باشد:



همچنین برای آشنایی با نحوۀ 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>

1. **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 حلقۀ تو در تو داریم. برای آشنایی با syntax حلقۀ for به صورتی که بازۀ 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 نحوۀ رسم هر شکل در ادامه آمده است:

1. برای رسم شکل اول یک حلقۀ بیرونی به اندازۀ n داریم (n سطر شکل). در سطر i ام به اندازۀ i بار عدد i را چاپ می­کنیم، بنابراین حلقۀ دوم باید از 0 تا i-1 حرکت کند و هر بار i را چاپ کند.
2. شکل دوم را می­توان به یک مثلث که قائدۀ آن پایین، و یک مثلث که قاعدۀ 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.
3. 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

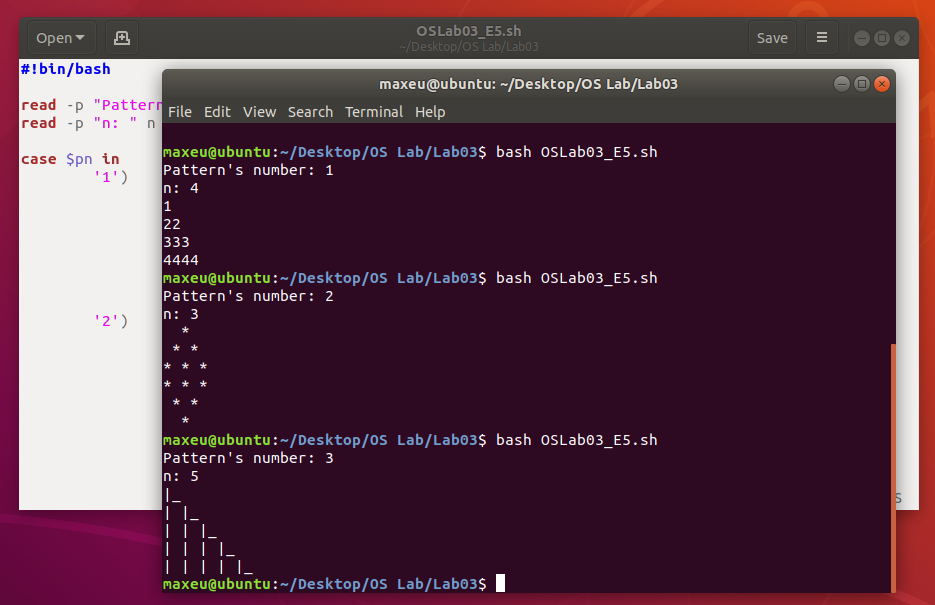
echo '|\_'

done

;;

esac

نمونۀ ورودی و خروجی این برنامه به صورت زیر می­باشد:



1. **طراحی ماشین حساب برای اعداد حقیقی:**

در این قسمت از آزمایش مانند آزمایش چهارم می­خواهیم یک ماشین حساب با 4 عمل اصلی طراحی کنیم، با این تفاوت که این بار ورودی­ها به جای اعداد صحیح اعداد حقیقی هستند.

برای آشنایی با نحوۀ کار با اعداد اعشاری در bash script از سایت­ زیر استفاده شده ­است:

<https://japp.io/shell-scripting/shell-script-to-add-two-float-numbers/>

همچنین برای برطرف کردن مشکل عمل تقسیم (نمایش حاصل تقسیم اعشاری به صورت عدد صحیح) از سایت زیر استفاده شده است:

https://stackoverflow.com/questions/12722095/how-do-i-use-floating-point-arithmetic-in-bash#:~:text=While%20you%20can't%20use,to%20get%20the%20fractional%20part.

از آن­جایی که کار با اعداد اعشاری در bash script پشتیبانی نمی­شود از ابزاری به نام BC (مخفف   
Basic Calculator) برای این کار استفاده می­شود. این ابزار جزئی از خود shell نیست، اما عبارت ریاضی را از standard input خود خوانده و پاسخ را بر روی standard output خود می­نویسد. عملگر | برای پایپ کردن در 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 –l به آن داده شده است که یک کتابخانۀ 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

;;

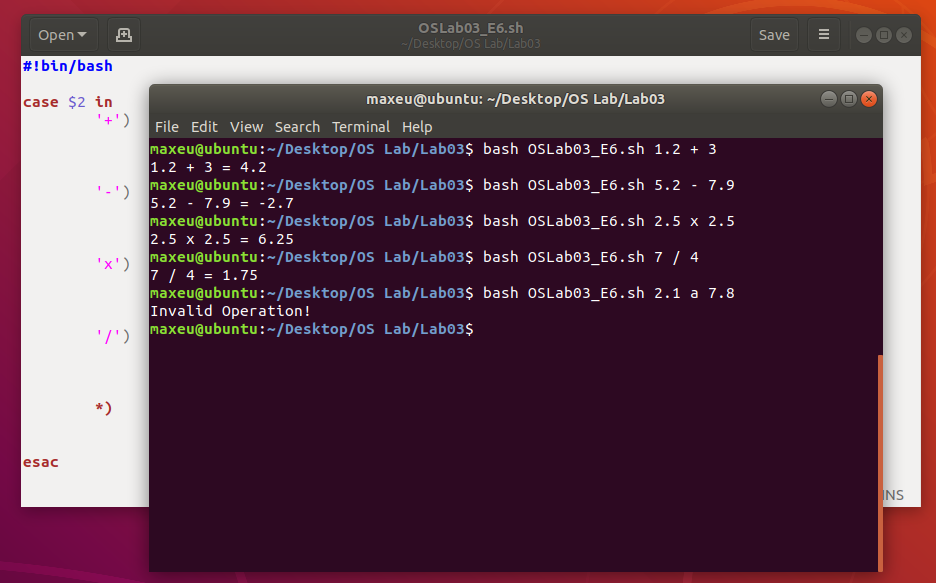
\*)

echo "Invalid Operation!"

;;

esac

نمونۀ ورودی و خروجی این برنامه به صورت زیر می­باشد:

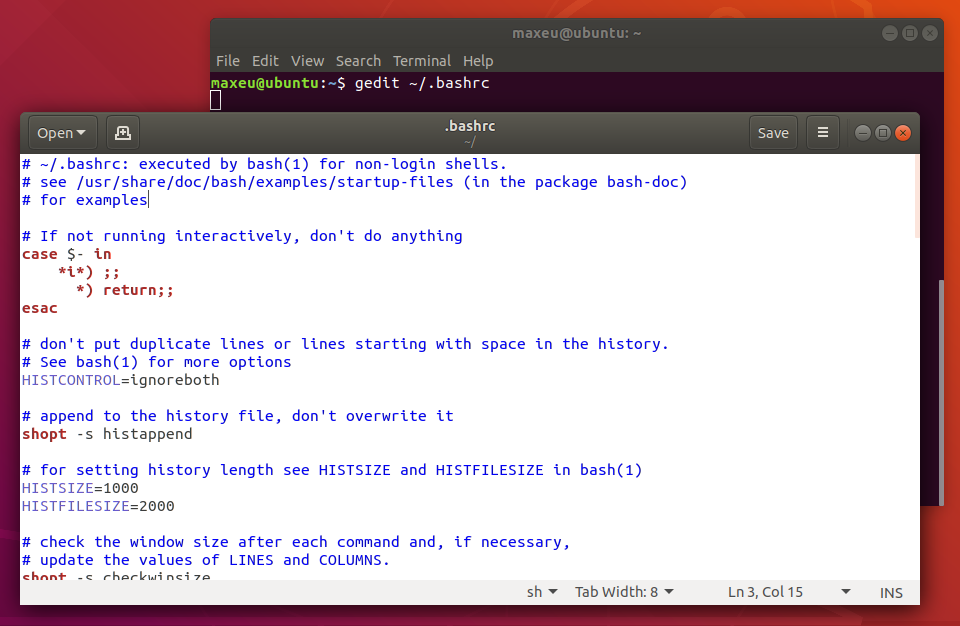


1. **افزودن command جدید به bash**

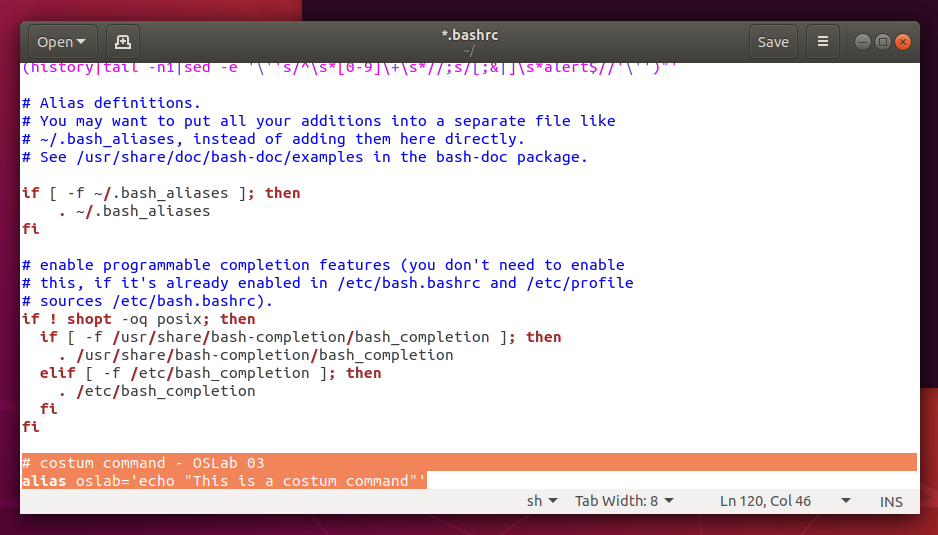
برای انجام این کار باید دستور مربوطه را به فایل .bashrc اضافه کنیم، بنابراین ابتدا با استفاده از دستور:

gedit ~/.bashrc

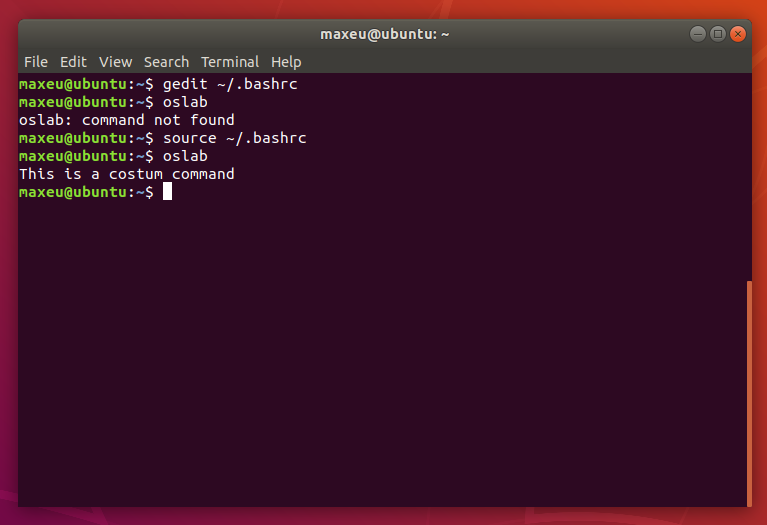
این فایل را با استفاده از text editor باز می­کنیم تا بتوانیم آن را تغییر دهیم.



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



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.



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.

1. **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. |

**2. اگر بیش از 10 آرگومان ورودی باشد، چگونه باید به مقدار دهمین آرگومان دست یافت؟**

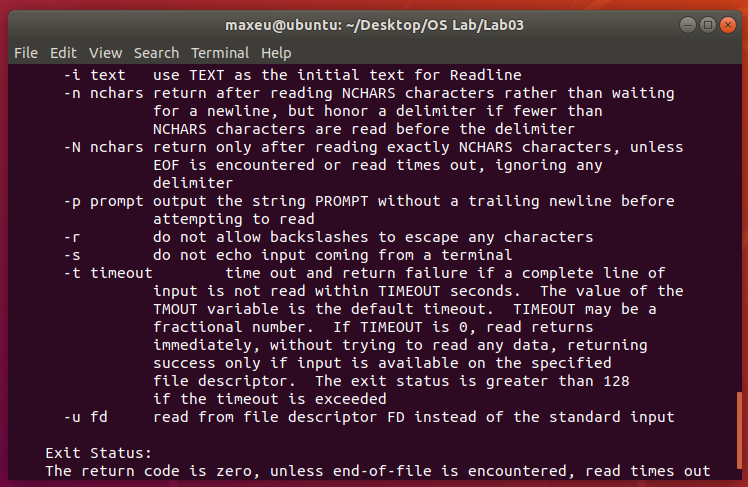
کافیست از {} دو طرف شمارۀ آرگومان استفاده کنیم. برای مثال:

echo "${10}"

دهمین آرگومان ورودی را چاپ می­کند.

**3. –p و –sp چه امکانی را فراهم می­کنند؟ مقدار متغیرهای uservar و passvar را در فایلی ذخیره کنید.**

اگر داخل ترمینال دستور read --help را اجرا کنیم راهنمای زیر نشان داده می­شود:



طبق این راهنما آرگومان –p برای خواندن ورودی از کاربر و ذخیرۀ آن در یک متغیر استفاده می­شود، به صورتی که برنامه برای دریافت ورودی به خط جدید نمی­رود. همچنین با استفاده از آرگومان –s ورودی کاربر بر روی ترمینال نمایش داده نمی­شود، به همین دلیل این آرگومان در گرفتن پسوردها کاربرد دارد.

اسکریپت زیر دو رشته username و password را از کاربر گرفته و آن را در یک فایل به نام info.txt ذخیره می­کند.

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 نوشته­ایم ذخیره شود.

1. **برای انجام محاسبات شیوه­های مختلفی وجود دارد. به مثال­های 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 نتیجۀ آن­ها استفاده می­شود.

1. در خط اول مقدار a با استفاده از دستور let برابر با 18 قرار داده شده، بنابراین در خط بعدی مقدار 18 چاپ می­شود.
2. پس از آن مقدار 20 = 4 × 5 چاپ می­شود.
3. در خط بعدی بخش صحیح حاصل تقسیم 5 بر 4 یعنی 1 چاپ می­شود.
4. در خط بعد باقی­ماندۀ 11 بر 2 یعنی 1 چاپ می­شود.
5. در خط بعد مقدار متغیر a برابر با مقدار عبارت 7 = 3 – 10 قرار داده شده و در خط بعد این مقدار چاپ می­شود.
6. پس از آن مقدار b برابر با مقدار a + 3 یعنی 10 = 3 + 7 قرار داده شده و در خط بعد این مقدار چاپ می­شود.
7. در نهایت توسط دستور ((b++)) مقدار متغیر b یکی افزایش یافته و در خط بعد مقدار 11 چاپ می­شود.

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

