دستور کار جلسه دوم

پوسته (Shell)

پوسته محیطی است که کاربر اطلاعات خود را در آن وارد می کند، وظیفه پوسته ترجمه ی این دستورات با دستورات سیستمی و در نهایت ارسال آنها به هسته است.

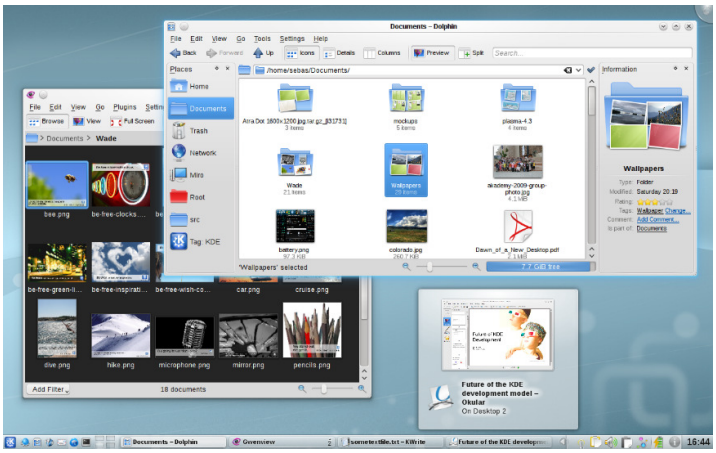
پوسته ها به دو دسته تقسیم می شوند :

- Graphical User Interface (GUI):

در این نوع رابطی گرافیکی وجود دارد که اطلاعات دریافتی از ورودی های مختلف به دستوری قابل فهم برای هسته تبدیل می شود و اجرا می شود. در زیر تصاویری از مطرحترین رابطهای گرافیکی موجود آورده شده است :

Gnome 3.1

KDE Plasma Desktop



- Command Line Interface (CLI)

در این محیط کاربر تمامی دستورات را با صفحه کلید وارد می کند، عیب این کار به خاطر سپردن تعداد زیادی از دستورات و گاهی خطاهایی است که در هنگام وارد کردن دستورات رخ می دهد اما مزیت بزرگ این کار استفاده از پوسته برای خودکارسازی فرآیندهای تکراریست

در زیر ۴ پوسته مشهور ذکر شده اند :

- Bourne Shell (sh)
- C Shell (csh)
- Bourne Again Shell (bash)
- Korn Shell (ksh)

متغیر های محلی (Environmental Variables)

هر پوسته قبل از اجرا شدن تعداد متغیر محلی را از فایل های پیکربندی می خواند، این متغیرها برای تمامی فرآیندهای ساخته شده در آن پوسته قابل دسترسی می باشند (مقادیر آنها به ارث می رسد). در این حالت تغییر دادن مقدار متغیر در فرآیند مقدار آن را در پوسته تغییر نخواهد داد.

در محیط پوسته می توان متغیر های کنونی را مقدار دهی کرد. همچنین می توان متغیرهای محلی جدید و متغیرهای پوسته جدید را تعریف کرد، متغیرهای پوسته همانند متغیرهای محلی تعریف می شوند با این تفاوت که فقط در همان پوسته قابل دسترسی هستند و فرآیندهایی که در پوسته ساخته می شوند به متغیرهای پوسته دسترسی ندارند.

۱. تعریف متغیر پوسته

برای تعریف متغیر در پوسته نیازی به تعریف نوع آن (رشته، صحیح، اعشاری و...) نیست :

```
variable_name=”value”
```

نکته : در هر دو طرف = نباید فاصله ای وجود داشته باشد.

نکته : اگر مقدار متغیر یک قسمت داشته باشد لزومی به استفاده از " " در دو طرف آن نیست ولی برای مقادیری که بین آنها جداکننده ای وجود دارد، قرار دادن " " الزامی است.برای مثال اگر متغیر device را داشته باشیم و بخواهیم مقدار pc را در آن ذخیره کنیم می توانیم به شکل زیر بنویسیم :

```
device=”laptop”
device=$device” pc”
echo $device
laptop pc
```

نکته : پوسته همه متغیرها را به عنوان رشته (string) در نظر می گیرد ولی خود قابل تفکیک اعداد و مقادیر حسابی از رشته ها را داراست و در مواقع لزوم می توان عملیات حسابی را بر روی متغیر ها اعمال کرد.

```
device=pc یا device=”pc”
```

ولی برای ذخیره مقدار laptop pc در آن باید به این شکل نوشته شود :

```
device=”laptop pc”
```

۲. export

محدوده متغیرهای محلی که در حالت قبل تعریف می شوند در یک پوسته است و پوسته های اجرا شده در پوسته کنونی از مقدار آنها بی اطلاعند. اگر بخواهیم متغیری محلی تعریف کنیم که در پوسته هایی که از این پس اجرا می شوند نیز قابل دسترسی باشند :

```
export variable_name=value
```

با اضافه کردن export متغیر به عنوان متغیری محلی شناخته می شود و به فرآیندهایی که در این پوسته ساخته می شوند به ارث

می رسد.

۳. echo

برای نمایش مقدار یک متغیر به کار می رود :

```
device=”laptop”
echo $device
laptop
```

نکته : در صورتی که رشته در یک خط قابل نمایش باشد نیازی به استفاده از “ ” در دو طرف آن نیست ولی اگر لازم باشد رشته در بیشتر از یک خط نشان داده شود باید از " " در ابتدا و انتهای رشته استفاده کرد.

مثال :

```
echo this is example یا echo “this is example”
this is example

echo this
is
example
error
echo “this
is
example”
this
is
example
```

۴. set

نمایش همه متغیرهای تعریف شده در پوسته

۵. alias

گاه دستوراتی استفاده می کنیم که بسیار طولانی بوده و خود از چندین دستور دیگر تشکیل می شوند، در اینصورت هربار تکرار این دستور طولانی و پیچیده احتمال خطا را بالا برده و همچنین وقت بسیاری را تلف می کند

راه حلی که پوسته در اختیار قرار می دهد به این شکل است که یک دستور طولانی را می توان در قالب یک متغیر محلی ذخیره کرد و هر بار به جای اجرای دستور طولانی، معادل کوتاه شده آن را به کار برد. دستور معادل کوتاه شده را alias (نام مستعار) می نامند و به صورت زیر تعریف می کنند :

```
alias name=”command sequence”
```

نکته : در هر دو طرف علامت = نباید هیچ فاصله ای وجود داشته باشد، همچنین وجود " " و یا ‘ ’ در سمت راست تساوی الزامی ست.

مثال :

دستور زیر لیست فایل های دایرکتوری جاری را با جزییات آنها گرفته و با استفاده از خط لوله به دستور less منتقل می کند تا در صفحه ای جداگانه نشان داده شود :

```
alias list=”ls -l | less”
```

حال این سوال مطرح است که اگر بخواهیم متغیرهای محلی یا دستورات مستعار را برای همه پوسته ها تعریف کنیم تا هر پوسته پس از راه اندازی سیستم از این مقادیر آگاهی داشته باشد چگونه و در کجا این مقادیر را تعریف کنیم؟ پاسخ این سوال در بخش های بعدی آورده شده است.

نکته : برای اضافه کردن مقادیر جدید به یک متغیر و همچنین حفظ مقادیر قبلی آن باید به شکل زیر عمل کرد . برای مثال متغیر device تعریف شده ومقدار ”pc“ در آن ذخیره شده، اگر بخواهیم مقدار "laptop" را نیز به انتهای آن اضافه کنیم :

```
device=$device” laptop”
echo $device
pc laptop
```

۶. unset

در صورتی که بخواهیم یک متغیر تعریف شده مقدارش را از دست داده و از این پس تعریف شده نباشد دستور unset را اجرا می کنیم :

```
deivce=”laptop”
unset device
```

متغیرهای محلی تعریف شده

HOME	مسیر دایرکتوری خانه برای کاربر
IFS	تعیین کننده Internal Field Separator، کاراکتری که به عنوان جدا کننده کلمات در پوسته به کار می رود.
LD_LIBRARY_PATH	اولین مسیر جستجوی objectها برای *Dynamic Linking
PATH	مسیر جستجوی برنامه ها و دستورات برای اجرا هر مسیر با : از مسیر دیگر تفکیک داده می شود.
PWD	مسیر کنونی (دایرکتوری کنونی)
RANDOM	مقداری تصادفی بین ۰ تا ۳۲۷۶۷ ایجاد می کند.
SHLVL	هر بار که یک پوسته جدید درون پوسته کنونی اجرا شود به مقدار این متغیر یکی اضافه می شود در حالت عادی پس از وارد شدن به سیستم (login) اولین پوسته اجرا شده و مقدار آن ۱ است.
TZ	منطقه ی زمانی سیستم
UID	شناسه عددی کاربر کنونی

** Dynamic Linking :

اغلب به هنگام استفاده از کتابخانه های بزرگ برای برنامه نویسی، کتابخانه به عنوان قسمتی از کد برنامه استفاده می شود ولی همراه با آن کامپایل نخواهد شد، در چنین حالتی linking در حین اجرای برنامه اتفاق می افتد به این طریق که کتابخانه به صورت مجموعه ای از object ها در کنار برنامه قرار گرفته و مسیر آن در کد اصلی برنامه ذکر می شود. این مسیر اغلب به عنوان متغیری محلی تعریف می شود.

اگر لازم باشد متغیرهای محلی جدید تعریف کنیم و مقدار آنها به صورت خودکار برای هر کاربر تعیین شود، باید متغیر در یکی از فایل های زیر نوشته شود :

- /etc/profile

اسکریپت نوشته شده در این فایل برای همه کاربران سیستم اجرا می شود، متغیرهای مشترک برای همه کاربران در این فایل تعریف می شوند.

- ~/.profile

پس از /etc/profile این اسکریپت برای هر کاربر اجرا می شود، مقادیر ویژه ی هر کاربر باید در این فایل تعریف شود.

نکته : برای متغیر محلی که در هر دو فایل تعریف شده باشد، مقدار تعیین شده در ~/.profile را در خود ذخیره می کند.

ویرایش فایل و ویرایشگر vi

Vi ویرایشگری است در محیط خط فرمان است که در ۱۹۷۶ توسط Bill Joy نوشته شده است. طی سالیان متمادی vi به عنوان ویرایشگر پیش فرض همراه با همه سیستم عامل های بر پایه Unix (Unix-Base) ارائه شده است.

Vi ویرایشگری ساده است اما قابلیت پیکربندی و انعطاف آن به قدری بالاست که از محبوبترین ویرایشگرهای جهان به شمار می آید. نسخه های مختلفی از این ویرایشگر وجود دارند که در این آزمایشگاه از ویرایشگر vim استفاده خواهد شد.

Vim (Vi Improved)

این ویرایشگر کارکرد و ساختار خود را از vi به ارث برده ولی قابلیت های بسیار بیشتری دارد که از مهم ترین آنها می توان به
syntax highlighting اشاره کرد.

فایل پیکربندی vim برای هر کاربر در مسیر ~/.vimrc قرار دارد، در جدول زیر مجموعه ای از مقادیر قابل پیکربندی آورده شده اند :

`:set nocompatible`	
`:set backspace=eol,start,indent`	
`syntax on`	
`colorscheme github`	
`set pumheight=10`	
`set nu`	
`set autoindent`	
`set cindent`	
`set shiftwidth=4`	
`set ts=4`	

vim - ویرایش فایل

برای ویرایش فایل ۲ حالت می توان متصور شد:

۱. فایل در حال حاضر وجود دارد

در این حالت بایستی به صورت مقابل عمل کرد :

```
vim path_to_file
```

در این حالت path_to_file مسیر دسترسی به فایل ذکر شده است.

۲. فایل در حال حاضر وجود ندارد

```
vim path_to_file
```

که در آن path_to_file مسیر مورد نظر ما برای ایجاد فایل و ذخیره آن خواهد بود.

عملکرد Vim شامل دو حالت زیر می باشد :

- Command-Mode

در این وضعیت می توان در فایل جابجا شد، مقداری را جستجو کرد، تغییرات نوشته شده در وضعیت insert را ذخیره کرد، از فایل خارج شد و سایر موارد دستوری را اعمال کرد.

- Insert-Mode

در این حالت می توان مقادیر نوشته شده در فایل را تغییر داد.

نکته : برای جابجا شدن از وضعیت Command-Mode به Insert-Mode می بایست کلید insert یا کلید i فشرده شود.

نکته : برای جابجا شدن از وضعیت Insert-Mode به Command-Mode می بایست کلید esc(ape) فشرده شود.

vim - مثال هایی از وضعیت Command-Mode

در این حالت vim منتظر دستوری برای ایجاد یک تغییر می شود.	:
نمایش راهنما برای vim، برای خروج از راهنما باید q: را اجرا نمود	:help
ذخیره سازی تغییرات اعمال شده	:w
خروج در صورتی که تغییر اعمال نشده	:q
در صورتی که تغییراتی اعمال شده باشد ولی مایل به ذخیره سازی آن نباشیم از این دستور استفاده می کنیم	:q!
ذخیره سازی و بعد از آن خروج	:wq
پاک کردن یک خط	d
انتخاب یک خط کامل	shift+v
رفتن به وضعیت visual mode، در این حالت کلمات در فاصله ای که اشاره گر اکنون قرار دارد تا هر کجا که قرار بگیرد انتخاب می شود	v
خنثی کردن آخرین عمل انجام شده ، مشابه عمل undo در سایر ویرایشگرها	u

خنثی کردن آخرین ۵ عمل انجام شده	5u
مشابه u ولی آخرین تغییر خنثی شده را دوباره اعمال می کند، مشابه redo در سایر ویرایشگرها	ctrl+r
اعمال دوباره ۶ تغییر آخر که خنثی شده اند	6 + ctrl +r
خنثی کردن همه تغییرات انجام شده از ابتدای بازکردن فایل	:edit!
انتقال کلمات انتخاب شده به حافظه و پاک کردن آنها	d
انتقال کلمات خط جاری از جایی که اشاره گر قرار دارد تا انتهای آن وهمچنین ۳ خط بعدی به حافظه و پاک کردن آنها	3d
کپی کردن کلمات انتخاب شده به حافظه	y
کپی کردن کلمات از جایی که اشاره گر قرار دارد تا انتهای خط جاری و همچنین ۸ خط بعدی به حافظه	8y
کلمات منتقل شده به حافظه را در جایی که اشاره گر قرار دارد درج می کند	p
کلمات منتقل شده به حافظه را ۳ بار از جایی که اشاره گر قرار دارد درج می کند	3p
انتقال اشاره گر به اولین خط	gg
انتقال اشاره گر به آخرین خط	G
انتقال به خط ۱۱	11G
پرش به کلمه بعدی (کلمه بعدی می تواند در خطوط بعدی نیز باشد)	w
پرش به چهارمین کلمه بعدی (کلمه بعدی می تواند در خطوط بعدی نیز باشد)	4w
غیر فعال کردن نتایج آخرین جستجو	:noh

vim - جستجو (search) در متن

ابتدا باید کاراکتر / وارد شود، در اینصورت vim برای جستجو آماده می شود.

بعد از آن باید عبارت یا الگوی مورد جستجو را وارد کرده و enter فشرده شود.

در صورت یافتن اولین تطابق، اشاره گر به ابتدای کلمه یافته شده منتقل می شود.

در صورتی که بیش از یک نتیجه برای جستجوی ما وجود داشته باشد با فشردن n به نتیجه بعدی و با فشردن N به نتیجه قبلی منتقل می شویم.

حالت مشابه دیگری نیز برای جستجو وجود دارد که در آن به جای / کاراکتر ? وارد می شود.

مثال :

vim - جایگزینی (substitution)

جایگزینی همانند جستجوست با این تفاوت که هر بار الگو یا عبارت مورد نظر یافت شود، عبارتی دیگر جایگزین آن می شود.

مثال :

```
:%s/expression1/expression2/gc
```

%s:	جستجو در همه خط ها
expression1	عبارت مورد جستجو
expression2	عبارت جایگزین شونده با expression1
g	جستجو برای همه مواردی که در یک خط یافت می شوند، در صورتی که نوشته نشود تنها اولین مورد یافته شده در هر خط جایگزین شده و پس از آن جستجو به خط بعدی می رود.
c	قبل از هر بار جایگزینی از کاربر برای انجام شدن یا نشدن جایگزینی پرسش می شود.

حالات بیشتری از تغییر و جستجو در آدرس زیر ذکر شده اند :

http://vim.wikia.com/wiki/Search_and_replace

اسکریپت نویسی (Script)

به مجموعه ای از دستورات خط فرمان که در یک فایل نوشته شده باشند، اسکریپت گفته می شود.

با وجود داشتن پوسته و خط فرمان چه نیازی به اسکریپت نویسی داریم ؟ پاسخ اینست که گاه لازمست یک فعالیت تکراری که شامل تعداد زیادی دستور خط فرمان است را برای ورودی های مختلف و بر روی ماشین های مختلف اجرا کنیم.

اسکریپت نویسی نه تنها زمان بسیار کمتری می گیرد بلکه در صورتی که اسکریپت به خوبی نوشته شده باشد کار را با دقتی بسیار بالاتر به انجام می رساند.

نکته : همیشه اسکریپت را با داده هایی مشابه داده های واقعی تست کنید تا از عملکرد آن مطمئن شوید، زیرا اغلب بازگرداندن تغییرات ایجاد شده توسط هر اسکریپت بسیار دشوار است.

زبان اسکریپت نویسی

همه پوسته های موجود در Unix به عنوان یک زبان اسکریپت نویسی قابل استفاده هستند.

در حال حاضر اغلب زبان های Python و Perl برای نوشتن اسکریپت به کار می روند، در اینجا روش نوشتن اسکریپت در پوسته توضیح داده می شود.

برای جزییات بیشتر به آدرسهای زیر مراجعه کنید :

http://tldp.org/LDP/abs/html/refcards.html
http://www.gnu.org/software/bash/manual/bashref.html

انتخاب پوسته برای اجرای اسکریپت

اسکریپت نوشته شده بایستی یک پوسته از پوسته های نصب شده در سیستم را انتخاب کرده و در آن محیط اجرا شود، در صورتی که این پوسته در اسکریپت ذکر نشده باشد اسکریپت در پوسته ی جاری شروع به کار می کند(پوسته ای که در زمان اجرا اسکریپت فعال باشد)

اولین خط در اسکریپت تعیین کننده پوسته انتخابی است :

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

در این حالت !# اعلام کننده مسیر پوسته مورد نظر برای اجراست، در اینجا پوسته ی bash انتخاب شده که در آدرس /bin/bash قرار دارد.

نکته : در برخی موارد لازمست که اسکریپت از هر جای سیستم قابل دسترسی باشد، به این منظور باید آن را در یکی از مسیرهای تعیین شده در PATH اضافه کرد و یا مسیر کنونی اسکریپت را به PATH افزود.

- متغیر ها در پوسته (variables)

تعریف متغیر همانند نعریف متغیر در پوسته است

در صورتی که متغیری در پوسته تعریف شود، پس از اجرای اسکریپت مقدار آن همچنان در پوسته قابل دسترسی است مگر آنکه به طریقی مقدار آن unset شود.

مثال :

```
vim script1.sh
device=”laptop”
cpu=”intel”
echo device
```

- آرگومان (arguments)

همانند توابع برای هر اسکریپت نیز می توان از آرگومان های ورودی آن استفاده کرد.

آرگومان ها مقادیری هستند که در رشته ی فراخوانی اسکریپت آورده می شوند، ترتیب دسترسی به آنها نیز به ترتیب وارد شدن آنها می باشد (ویرایش)

آرگومان ۰ نام اسکریپت فراخوانی شده است و با مقدار $0 قابل دسترسی است

مثال:

```
./script.sh     hello world with  arguments
$0 = script1
$1 = hello
$2 = world
$3 = with
$4 = arguments
```

متغیرهایی با مقادیر ویژه در پوسته وجود دارند، توضیحات مربوطه در جدول زیر آورده شده اند:

متغیر	مقدار
$0	نام اسکریپت اجرا شده
$1	مقدار آرگومان اول
${10}	مقدار ۱۰ مین آرگومان در اسکریپت کنونی
$#	تعداد کل آرگومان های ورودی
${#*}	تعداد کل آرگومان های ورودی
${#@}	تعداد کل آرگومان های ورودی
"$*"	تمامی آرگومان های ورودی در یک رشته
"$@"	آرایه ای از تمامی آرگومان های ورودی در اسکریپت کنونی، نام اسکریپت در این آرایه قرار ندارد و آرگومان شماره ۰ در واقع اولین آرگومان ورودی می باشد
$?	مقدار بازگشتی آخرین دستور اجراشده، اغلب در صورت موفقیت در اجرای دستور این مقدار برابر ۰ است
$$	شماره فرآینده (PID=Process IDentifier) اسکریپت کنونی

مثال :

خواندن آرگومان های با شماره فرد در یک اسکریپت

```
arg_value=(“$@”)
arg_num=”$#”
for ((i=۰;i<arg_num;i+=2));
do
      echo ${args($i)}
done
```

- عبارت شرطی (if)

- if … then

```
#!/bin/bash
if [ "foo" == "foo" ]; then
   echo expression evaluated as true
fi
```

- if … then … else

```
#!/bin/bash
if [ "foo" == "foo" ]; then
   echo expression evaluated as true
else
   echo expression evaluated as false
fi
```

```
#!/bin/bash
   T1="foo"
   T2="bar"
   if [ "$T1" == "$T2" ]; then
       echo expression evaluated as true
   else
       echo expression evaluated as false
   fi
```

موارد ذکر شده در جدول زیر می توانند به عنوان شرط **if** قرار بگیرند :

[-a FILE]	True if FILE exists.
[-e FILE]	True if FILE exists.
[-f FILE]	True if FILE exists and is a regular file.
[STRING1 == STRING2]	True if the strings are equal.
[STRING1 != STRING2]	True if the strings are not equal.
[STRING1 < STRING2]	True if "STRING1" sorts before "STRING2" lexicographically in the current locale.
[STRING1 > STRING2]	True if "STRING1" sorts after "STRING2" lexicographically in the current locale.

- **حلقه**

```
for (condition);
```

```
      do
      works to do
      done
```

مثال : نمایش نتیجه اجرای دستور ls و چاپ کردن خط به خط آن

```
#!/bin/bash
for i in $( ls ); do
      echo item: $i
done
```

مثال : نمایش مقدار همه آرگومان ها

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

مثال:

C-like for

```
#!/bin/bash
for i in `seq 1 10`; do
      echo $i
done
```

مثال :

```
#!/bin/bash
for (( c=1; c<=5; c++ ))
do
   echo "Welcome $c times"
done
```

While :

```
#!/bin/bash
COUNTER=0
while [  $COUNTER -lt 10 ]; do
      echo The counter is $COUNTER
```

```
      let COUNTER=COUNTER+1
done
```

Until :

```
#!/bin/bash
COUNTER=20
until [  $COUNTER -lt 10 ]; do
      echo COUNTER $COUNTER
      let COUNTER-=1
done
```

- تابع

```
#!/bin/bash
function quit {
      exit
      }
function hello {
      echo Hello!
         }
hello
quit
echo foo
```

ابزارهای برنامه نویسی

- glibc

http://www.gnu.org/software/libc/index.html

هر سیستم عامل مشابه Unix (Unix-like) نیاز به کتابخانه ای به زبان C دارد چرا که ساختار های اصلی Unix به زبان C نوشته شده اند از جمله آنها فراخوانی های سیستمی (System Call) که در ادامه دستور کار به تفصیل از آنها استفاده شده است.

Gnu C Library یا glibc کتابخانه ای استاندارد به زبان C که توسط بنیاد GNU نگهداری می شود، این کتابخانه با استانداردهای C11 و POSIX.1-2008 سازگاری کامل دارد.

در نوشتن کد برنامه های دستور کار تماما از کتابخانه glibc استفاده شده است.

- gcc

مراحل کامپایل کردن در gcc:

- Parser: بررسی token ها و تحلیل معنایی (Semantic Analysis)
- ایجاد کد میانی و بهینه سازی(کد تولید شده در این مرحله بسیار سطح پایین است ولی همچنان به زبان ماشین ترجمه نشده است)
- Assembler: ایجاد فایل های object با پسوند o.
- Linker: ادغام فایل های object در یکدیگر و در نهایت تهیه یک فایل اجرایی (executable) و یا یکDynamic Library ، فایل تولید شده در این مرحله به صورت پیش فرض نام a.out را خواهد داشت.

`gcc -c code.c`	فایل code.c را به عنوان ورودی گرفته، فایل object با نام code.o را خواهد ساخت
`gcc code.c`	فایل code.c را به عنوان ورودی گرفته، فایل اجرایی با نام a.out را خواهد ساخت
`gcc code.c -o app_name`	فایل code.c را به عنوان ورودی گرفته، فایل اجرایی app_name را خواهد ساخت

کد خود را به زبان C در فایلی نوشته و ذخیره می کنیم، در این مثال نام فایل program.c انتخاب شده است، سپس با دستور زیر برنامه کامپایل خواهد شد:

```
gcc program.c
a.out
```

نتیجه فایل اجرایی a.out است، در صورتی که بخواهیم نام فایل اجرایی را خود انتخاب کنیم باید مطابق دستور زیر عمل کنیم :

```
gcc program.c -o app
app
```

در اینجا نام app برای فایل اجرایی انتخاب شده است.

اجرای دستورات خط فرمان در برنامه

کتابخانه stdlib.h تابعی با نام `system(char * str)` را ارائه می کند که بوسیله آن می توان دستورات خط فرمان را در برنامه به زبان C اجرا کرد. عیب این روش کند بودن آن و همچنین عدم دسترسی به نتیجه اجرای دستور است.

در مثال زیر دستور ls در پوسته اجرا کننده برنامه app اجرا می شود، عیب بزرگ این روش اینست که اگر لازم باشد نتایج دستور ls در همین برنامه استفاده شوند، باید ابتدا این مقادیر را در یک فایل ذخیره کرده و از فایل بازخوانی شوند.

```
//app.c
#include <stdio.h>
#include <stdlib.h>

int main()
{
char command="ls";
system (command);
return 0;
}
```

بکارگیری ابزار Make در فرآیند برنامه نویسی

برای کامپایل کردن تعداد محدودی از فایل های object، کد برنامه و تعیین آدرس کتابخانه های داینمیک لازمست هر بار دستورات لازم برای کامپایل را وارد کنیم که اینکار احتمال خطای بالا و اتلاف وقت به همراه دارد. برای اجتناب از این وضعیت می توان دستورات لازم را در یک فایل نوشته و از ابزاری به نام make بهره برد.این کار نوعی اسکریپت نویسی است با این تفاوت که نتیجه اجرای دستورات به کامپایل یک برنامه منتهی می شود.

از مزایای make بررسی تغییرات ایجاد شده در فایل هاست، به این معنی که اگر در یکی از فایل هایی که در

آن ذکر شده اند تغییری ایجاد شده باشد، نیازی به کامپایل همه منابع نیست و فقط فایل تغییر یافته مجددا کامپایل خواهد شد.

در حالت پیش فرض پس از اجرای دستور `make`، فایلی با نام Makefile در همان مسیر جستجو شده و دستورات داخل آن توسط make اجرا خواهد شد.

هر Makefile از چند قسمت تشکیل شده که به ترتیب تعیین شده دستورات هر قسمت اجرا می شوند :

```
[target]: dependencies
	commands
```

مثال :

```
vim Makefile
#example:
all: app_name

app_name: code1.o code2.o
	clang code1.o    code2.o    app_name
code1.o: code1.c
	clang -c   code1.c
code2.o: code2.c
	clang -c code2.c
clean:
	rm -rf *o code1.o code2.o app_name
```

نکته : اگر نامی غیر از Makefile برای فایل مورد نظر انتخاب شده باشد باید از دستور `make -f file_name` استفاده کرد که در آن file_name مسیر دسترسی به فایل مورد نظر ماست.

همچنین می توان متغیرهایی در Makefile تعریف کرد، اینکار مشابه تعریف متغیر پوسته انجام می گیرد، برای دسترسی به مقدار متغیر بایستی متغیر را در $(var) به کار برد.

```
CC=clang
$(CC) code.c -o app
```

نکته : همواره لازم نیست همه دستورات یک Makefile اجرا شوند، می توان برای اجرای هر قسمت از دستور make target استفاده کرد که در آن target نام قسمت مورد نظر ماست، در مثال زیر برای اجرای کد قسمت clean کافیست دستور زیر را اجرا کنیم :

```
make clean
```

apt و نصب بسته های نرم افزاری (Package)

- مدیریت بسته های نرم افزاری در Ubuntu بوسیله apt صورت می گیرد:

به روز رسانی پایگاه داده apt	`apt-get update`
جستجوی بسته هایی که نام آنها شامل pkg است	`apt-cache search <pkg>`
نصب بسته ای با نام pkg	`Apt-get install <pkg>`
نصب مجدد بسته ای با نام pkg	`Apt-get install –reinstall <pkg>`