ISBN 978-954-00-0170-8

ЮГОЗАПАДЕН УНИВЕРСИТЕТ
„НЕОФИТ РИЛСКИ”
Благоевград

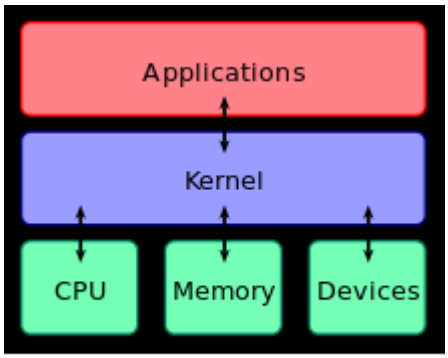
Въведение в Linux

Радослав Мавревски

Съдържание:

Въведение

Операционната система (ОС) е софтуер на ниско ниво, който планира и разпределя задачите, разпределя памет и осигурява интерфейсите към периферния хардуер, като например принтери, дискови устройства, екран, клавиатура и мишка. Операционната система има две основни части: ядро и системни програми (виж **Фиг. 1**). Ядрото разпределя машинните ресурси, включително памет, дисково пространство и CPU цикли, за всички други програми, които работят на компютъра. Системните програми включват драйвери за устройства, библиотеки, помощни програми, черупки (командни интерпретатори), конфигурационни скриптове и файлове, приложни програми, сървъри, и документация. Те извършват задачи от по-високо ниво, често в качеството на сървъри в клиент / сървър връзка. Много от библиотеки, сървъри, и полезни програми са написани от проекта GNU.



Фиг. 1. Основни части на ОС

Основни функции на ОС са:

- Управление на процесите: пускане, унищожаване на процес; разпределяне ресурсите между различни процеси (разбира се ресурсът “време на процесора”, който се дели между едновременно работещи процеси в изчислителната система). Тази функция е задължителна за ОС;

- Управление на паметта: Тази функция е задължителна за ОС. Тя включва: Разпределение на паметта между процесите – ОС се грижи за разпределението на наличната в компютъра памет за процесите, които работят едновременно в ИС. Защита на паметта – ОС се грижи един процес да няма достъп до паметта на друг процес.

- Управление на файловата система: Файл: минимална наименувана порция информация, която се пази на външен носител (може да е и в паметта на машината - “RAM – дискове”: файл в RAM паметта на компютъра. Тази памет е бърза, но енергозависима.) Логическият вид на файловата система (файловете на компютъра) е йерархичен, дървовиден. ОС се грижи за логическият файл да се представи физически върху дисковото устройство и дава възможност за работа с файлове (достъп). Физически файлът представлява разпръснати блокове от байтове върху целия носител, а ОС се грижи тези разпръснати блокове да изглеждат като последователност от байтове за програмите.

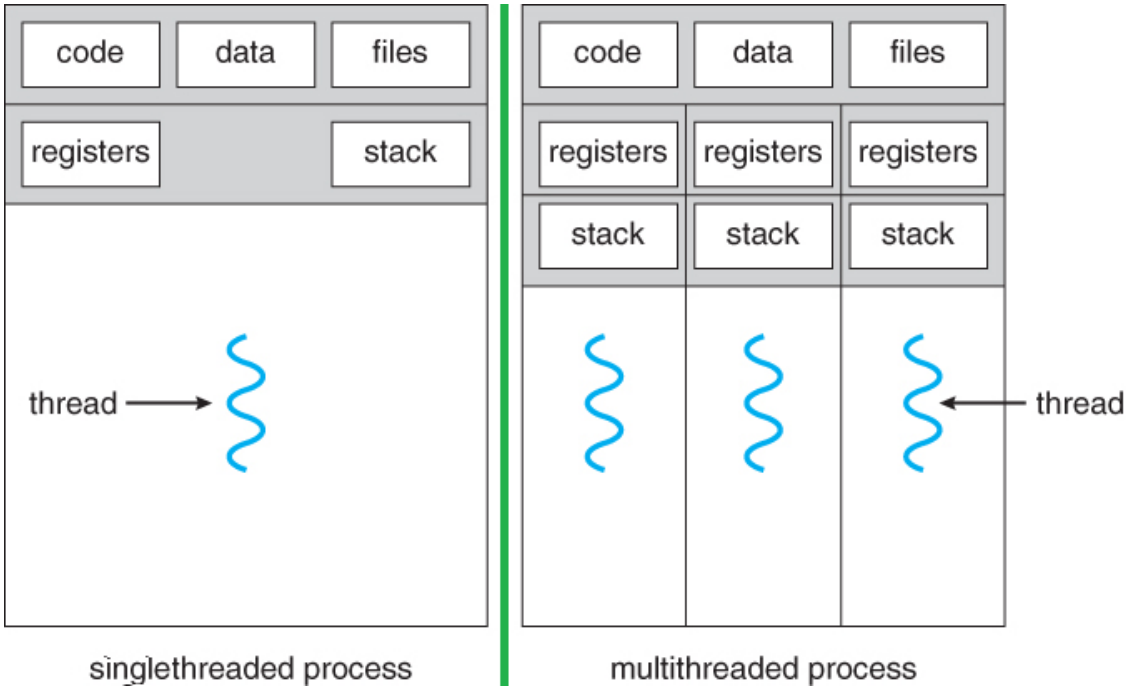
- Управление на периферията: само чрез ОС може да се управлява хардуера. Например отпечатването на принтер, сканиране, запис на CD ROM, управлението на звука (звукови карти) са услуги предоставени от ОС.

- Команден интерпретатор (синоним на команден процесор): това се програми, които чакат въвеждането на команди (например от клавиатурата, може и от файл) и ги изпълняват;

ОС се разделя по това дали поддържа само един процес или повече (виж **Фиг. 2**):

- Еднопроцесни: при тези ОС за да се стартира един процес, трябва предишният да е завършил;
- Многопроцесни: при тези ОС може един (или повече) процеси да стартират, преди да е завършил друг, т.е. възможно е няколко

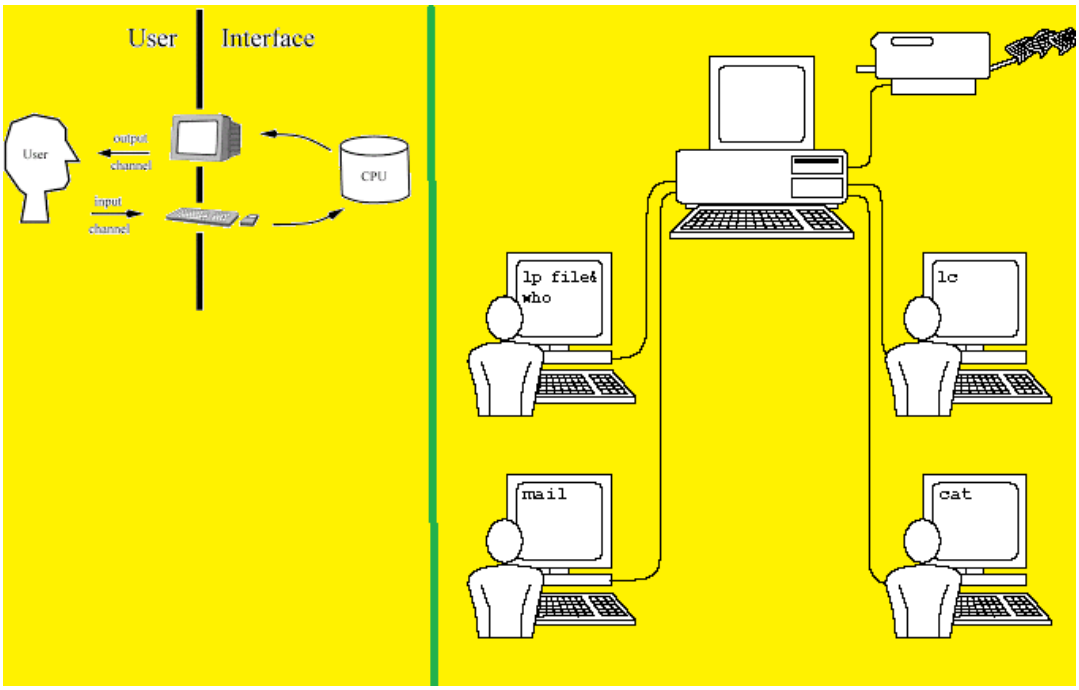
процеса да работят едновременно. Всеки процес му се струва, че разполага с цялата машина. Физически е възможно една машина да бъде с един процесор – еднопроцесорна и с повече процесори – многопроцесорна.



Фиг. 2. Еднопроцесни и многопроцесни ОС

ОС се разделя и по това колко потребителя могат да я използват(виж **Фиг. 3**):

- Многопотребителски ОС: това са тези ОС, при които може много потребители да ползват една ОС едновременно, и всеки от тези потребители има собствена среда (например собствен интерфейс, който се определя в зависимост от името, с което влиза конкретния потребител);
- Еднопотребителски ОС. Обслужва само един потребител. Пример за такава е MS DOS.

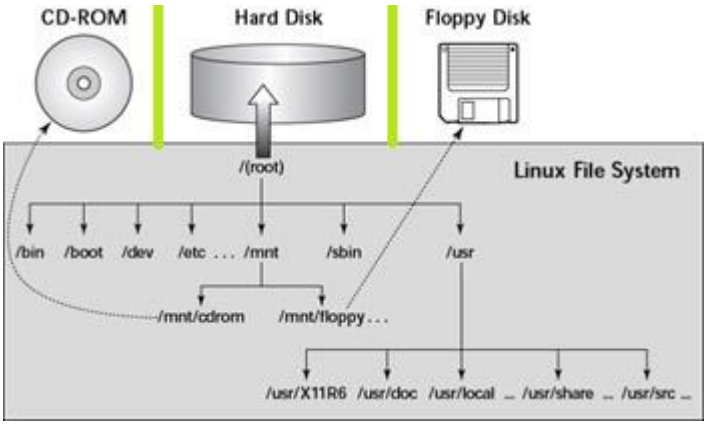


Фиг. 3. Еднопотребителска и многопотребителска ОС

Linux (линукс) е многопотребителска и многозадачна операционна система. Като многозадачна операционна система Linux във всеки един момент от работата си поддържа няколко активни програми в паметта. Освен многозадачна Linux е и многопотребителска операционна система. Тя може да съдържа набор от потребители, които се идентифицират пред нея чрез потребителско име и парола. Всеки един потребител има собствена директория, в която може да пази своите файлове (включително и конфигурационните файлове на различните програми), като тази директория е недостъпна за другите потребители. Всяка една Linux система има един свръхпотребител наречен root. Той администрира система и има пълните права върху нея. Този свръхпотребител се създава при инсталирането на дистрибуцията, като след това той създава другите потребители.

Цялата тази информация(а и още много друга) се пази на твърдият диск на компютъра. Файлова система, най-общо казано се нарича набор от

файлове, структурирани по някакъв начин и съхранени върху запомнящо устройство (виж **Фиг. 4**).



Фиг. 4. Линукс файлови системи

Едно от основните предимства на Linux е, че поддържа множество файлови системи. Това го прави особено гъвкав при съвместно съжителство с други операционни системи. Linux без проблем поддържа ext и ext2, msdos, ntfs и vfat, xia, minix, umsdos, iso9660, ufs, proc, sysv, ncp, smb, affs, hpfs, raiserfs и др.

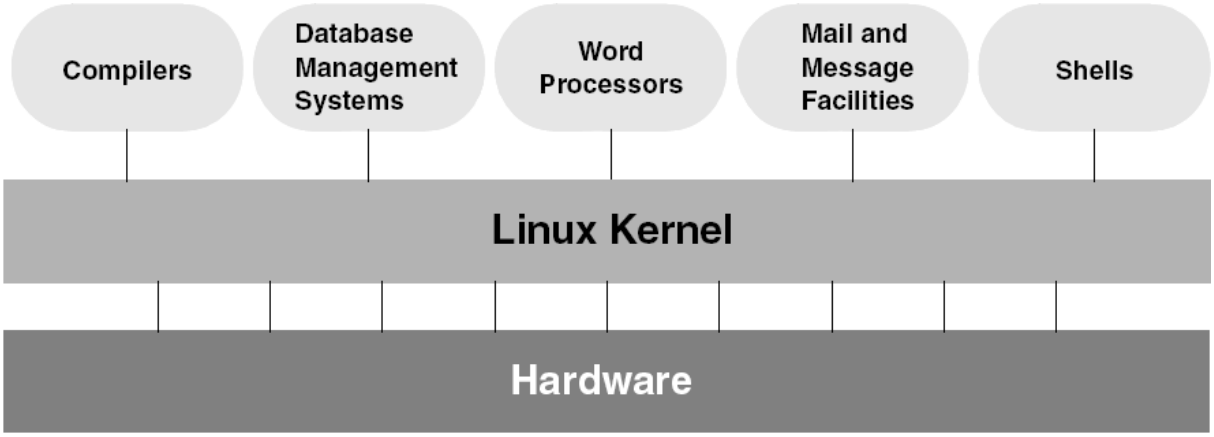
Ядрото на Linux е разработено от финландски студент Linus Torvalds, които използва интернет, за да предостави изходния код веднага на разположение на другите безплатно. Torvalds пуска Linux версия 0.01 през септември 1991 година. Програмисти по целия свят побързаха да разширят ядрото, развият други инструменти и добавят функционалност, намираща се в BSD UNIX и UNIX System V (SVR4), както и нова функционалност. Името Linux е комбинация от Linus и UNIX.

Linux операционна система, която е разработена чрез сътрудничеството на много, много хора по света, продукт на интернет е, и е безплатна (с отворен код) операционна система. С други думи, целия изходен код е а безплатен. Вие сте свободни да го изучавате, да го разпространявате и да го променяте. Като резултат, кода е достъпна безплатно – няма такса за софтуер, сорс код, документация и поддръжка.

Повече от 95 на сто на Linux операционна система е написана на езика за програмиране C.

GPL лицензa под който се разпространява Linux казва, че имате право да копирате, променяте и разпространявате кода предмет на споразумението. Когато разпространявате код, обаче, трябва да разпространявате същия лиценз с кода, като по този начин на код и лиценз са неразделни.

Изгледа на слоевете на Linux операционна система е показан на **Фиг. 5.**



Фиг. 5. Изгледа на слоевете на Linux операционна система

Упражнения:

1. *Какво е свободен софтуер? Избройте три характеристики на свободен софтуер.*
2. *Защо Linux е популярна? Защо е популярна в академичните среди?*
3. *Какви са многопотребителски системи? Защо те са успешни?*
4. *На какъв език е написан Linux?*
5. *Какво е помощна програма?*
6. *Какви са основните условия на GNU General Public License?*

Основни команди в Линукс

Когато Linus Torvalds и дълго време след това, Linux не е имал графичен потребителски интерфейс (GUI). Тя използва текстов интерфейс, също като на командния ред (CLI). Всички инструменти се използват от командният ред. Днес Linux GUI е важен, но много хора, особено на системните администратори изпълнява много от комуналните услуги в командния ред. Инструментите от командния ред често са по-бързо, по-силни, или по-пълна отколкото съответстващите им такива в GUI. Понякога GUI не е полезен и някой хора предпочитат да работят с командния ред.

Преди да започнете работа с черупки (shell), е важно да разберете нещо за символите, които са специални за обвивката. Тези знаци са споменати тук, така че можете да избегнете случайно да ги използват като редовни знаци, докато не разберете как черупката ги интерпретира.

& ; | * ? ' " ‘ [ ] ( ) $ < > { } # / \ ! ~

Системна информация

date – показва текущата дата и час

cal – показва календара за текущия месец

```
[student@shell ~]$ cal
     April 2010
Su Mo Tu We Th Fr Sa
             1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30

[student@shell ~]$
```

whoami – показва името, с което сте влезли в системата

```
[student@shell ~]$ whoami
student
[student@shell ~]$
```

uname -a – показва информация за ядрото

```
[student@shell ~]$ uname -a
FreeBSD shell.cjb.net 8.0-RELEASE FreeBSD 8.0-RELEASE #0: Wed Dec  2 21:08:33 MS
T 2009     root@shell.cjb.net:/usr/obj/usr/src/sys/CJB  amd64
[student@shell ~]$
```

history - история на въведените команди в конзолата

```
  468  cd students
  469  cd
  470  clear
  471  history
[student@shell ~]$
```

man *име_на_команда* – пълно описание на съответната команда

```
[student@shell ~]$ man whoami
WHOAMI(1)               FreeBSD General Commands Manual              WHOAMI(1)

NAME
     whoami -- display effective user id

SYNOPSIS
     whoami

DESCRIPTION
     The whoami utility has been obsoleted by the id(1) utility, and is equiv-
     alent to ``id -un''.  The command ``id -p'' is suggested for normal
     interactive use.

     The whoami utility displays your effective user ID as a name.

EXIT STATUS
     The whoami utility exits 0 on success, and >0 if an error occurs.

SEE ALSO
     id(1)

FreeBSD 8.0                     June 6, 1993                     FreeBSD 8.0
[student@shell ~]$
```

du – показва използваното пространство в текущата директория

```
[student@shell ~]$ du
492     ./psybnc/help
258     ./psybnc/lang
6       ./psybnc/log
114     ./psybnc/menuconf/help
242     ./psybnc/menuconf
4       ./psybnc/motd
4       ./psybnc/scripts/example
8       ./psybnc/scripts
1048    ./psybnc/src
98      ./psybnc/tools
8       ./psybnc/key
2540    ./psybnc
82      ./.,/randfiles
1394    ./.,
4514    .
[student@shell ~]$
```

echo *“текст”*- извежда текст на нов ред

```
[student@shell ~]$ echo "Az izuchavam Linux"
Az izuchavam Linux
[student@shell ~]$
```

echo *"специален_символ"*- извежда специален символ на нов ред

```
[student@shell ~]$ echo "*"
*
[student@shell ~]$
```

echo *"текст" > име_на_файл* – записва текста във файл с задаено име

```
[student@shell ~]$ echo "Az izuchavam linux" > Linux.txt
[student@shell ~]$
```

nano – стартира текстов редактор с който можете да редактирате файлове

```
  GNU nano 2.0.9                New Buffer

^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

clear - изчиства показваната информация в конзолата

Работа с файлове

ls – показва списък с файловете в директорията

```
[student@shell ~]$ ls
Linux.txt               psybnc                  psybnc-freebsd.tar
[student@shell ~]$
```

ls -l – показва форматиран списък с файловете в директорията

```
[student@shell ~]$ ls -l
total 580
-rw-------   1 student  users      19 Apr 15 08:04 Linux.txt
drwx------  11 student  users     512 Mar 30 15:14 psybnc
-rw-------   1 student  users  570423 Apr 24  2009 psybnc-freebsd.tar
[student@shell ~]$
```

ls -al – показва форматиран списък, заедно със скритите файлове

```
[student@shell ~]$ ls -al
total 652
drwx------     4 student  users     512 Apr 15 08:11 .
drwx------     3 student  users     512 Feb 10 15:18 .,
drwxr-xr-x  3698 0        0       67072 Apr 15 08:15 ..
-rw-------     1 student  users    1113 Apr 15 06:46 .bash_history
-rw-------     1 student  users      19 Apr 15 08:04 Linux.txt
drwx------    11 student  users     512 Mar 30 15:14 psybnc
-rw-------     1 student  users  570423 Apr 24  2009 psybnc-freebsd.tar
[student@shell ~]$
```

cat *име_на_файл* – показва съдържанието на файла

```
[student@shell ~]$ cat Linux.txt
Az izuchavam linux
[student@shell ~]$
```

cat > *име_на_файл* – пренасочва стандартния вход към файл (създава се файл)

```
[student@shell ~]$ cat > specialnost
KST
III Kusr
```

less *име_на_файл* - показва съдържанието на файлът по страници

```
In order to understand the popularity of Linux, we need to travel back in time,
about 30 years ago...

Imagine computers as big as houses, even stadiums. While the sizes of those comp
readme.txt
```

touch *име_на_файл* – създава файл със зададеното име

```
[student@shell ~]$ touch student.txt
[student@shell ~]$
```

cp *файл1 файл2* – копира файл1 във файл2

```
[student@shell ~]$ cp Linux.txt Linux_copy.txt
[student@shell ~]$
```

mv *файл1 файл2* – преименува или премества файл1 във файл2. Ако файл2 е съществуваща директория, премества файл1 в нея

```
[student@shell ~]$ mv Linux.txt Linux_New.txt
[student@shell ~]$
```

rm *име_на_файл* – изтрива файл

```
[student@shell ~]$ rm Linux
[student@shell ~]$
```

file *име_на_файл* - показва формата на файла

```
[student@shell ~]$ file readme.txt
readme.txt: ASCII English text, with very long lines
[student@shell ~]$
```

pwd – показва текущата директория

```
[student@shell ~]$ pwd
/users/home/student
[student@shell ~]$
```

mkdir *име_на_директория* – създава директория

```
[student@shell ~]$ mkdir students
[student@shell ~]$
```

cp -r *дир1 дир2* – копира дир1 в дир2; създава дир2 ако тя не съществува

```
[student@shell ~]$ cp -r students students_copy
[student@shell ~]$
```

rmdir *име_на_директория* – изтрива директория (директорията трябва да е празна и трябва да се намираме над директорията която искаме да изтрием). За изтриване на директория заедно с нейното съдържание се използва командата **rm -rf**

```
[student@shell ~]$ rmdir students
[student@shell ~]$
```

cd *име_на_директория* – влиза в директория

```
[student@shell ~]$ cd students
[student@shell ~/students]$
```

cd .. – връща една директория на горе

```
[student@shell ~/students]$ cd ..
[student@shell ~]$
```

cd - връща в главната директория

```
[student@shell ~/students/kst]$ cd
[student@shell ~]$
```

cd - - връща ви в предишната работна директория, преди последната cd команда

```
[student@shell ~/students/kst]$ cd
[student@shell ~]$ cd -
/users/home/student/students/kst
[student@shell ~/students/kst]$
```

/ - индикация за пълен път (*cp /home/student/.../<файл> /home/student/.../<файл>*)

Права за достъп до файлове

chmod *octal* *име_на_файл* – променя позволението за използване на файл или директория. Octal е три цифри - по една за потребител, група и всички останали, събирайки:

4 – за четене (**r**)

2 – за писане (**w**)

1 – за стартиране (**x**)

Пример:

chmod 777 – rwx за всички

chmod 755 – rwx за собственика, rx за групата и за всички

Например за да дадете права за стартиране на файла hello.sh напишете:

```
[student@shell ~]$ chmod 700 hello.sh
[student@shell ~]$
```

Работа с архиви

gzip *име_на_файл* – компресира файл с име *име_на_файл* и го преименува във *име_на_файл.gz*

```
[student@shell ~]$ gzip readme.txt
[student@shell ~]$
```

gzip -d *име_на_файл.gz* – декомпресира файл с име *име_на_файл.gz* и го преименува във *име_на_файл*

```
[student@shell ~]$ gzip -d readme.txt
[student@shell ~]$
```

tar czf *име_на_архив.tar.gz имена_на_файлове* – създава архив с компресия Gzip

```
[student@shell ~]$ tar czf readme_archive.tar.gz readme.txt Linux.txt
[student@shell ~]$
```

tar xzf *име_на_архив.tar.gz* – разархивира Gzip архив

```
[student@shell ~]$ tar xzf readme_archive.tar.gz
[student@shell ~]$
```

Бързи клавиши

Ctrl+C – прекъсва текущата команда

Ctrl+Z – стопира текущата команда. За стартиране, използвайте fg и bg

Ctrl+D – прекъсване на сесията, като **exit**

Ctrl+W – изтрива една дума от текущия ред

Ctrl+U – изтрива целия ред

Ctrl+R – търсене в последните команди

!! - повтаря последната команда

Копиране на текст *- селектира се с мишката и се поставя на новото място чрез натискане на средния бутон (или двата бутона заедно).*

Упражнения:

1. *Да се изведе на екрана текущата дата и час.*
2. *Да се изведе на екрана календара за текущия месец.*
3. *Да се изведе на екрана името, с което сте влезли в системат.*
4. *Да се изведе на екрана информация за ядрото.*
5. *Да се изведе на екрана история на въведените команди в конзолата.*
6. *Да се изведе на екрана пълно описание на команда date.*

7. *Да се изведе на екрана използваното пространство в текущата директория.*
8. *Да се изведе на екрана текст „Welcome to Linux”.*
9. *Да се запише текста „Welcome to Linux” във файл с име welcome.txt.*
10. *Да за изчисти показаната информация в конзолата.*
11. *Да се изведе списък на файловете в директорията home/username.*
12. *Да се изведе форматиран списък на файловете в директорията home/username.*
13. *Да се изведе форматиран списък, заедно със скритите файлове в директорията home/username.*
14. *Да се покаже текущата директория.*
15. *Да се създаде директория с име вашия факултетен номер.*
16. *Създайте файл с име вашето име в директорията home/username /вашия_факултетен_номер.*
17. *Да се стартира текстовия редактор „nano”. Да се отвор файла който създадохте (файла с име вашето име) и да се въведат трите ви имена. Запишете файла.*
18. *Да се преименува файла с име вашето име във файл с име myfile.*
19. *Да се покаже формата на файла с име myfile.*
20. *Да се създаде архив на файла myfile.*
21. *Да се изтрие файла с име myfile.*
22. *Да се изтрие архива на файла myfile.*
23. *Да се изтрие директорията с име вашия факултетен номер.*

Директориите в Линукс

Всяка една операционна система има определена структура на директориите и всяка директория има определена функция.

В Линукс има два типа директории: системни и потребителски. Системни са тези директории, които се създават при инсталирането на операционната система.

Такива например са директориите **/dev, /bin, /lib** и т.н. Потребителски са директориите, които се създават от потребителите на операционната система с цел да пазят собствена информация в тях. Такива са директориите */home/username* и всички поддиректории вътре в нея.

Основните системни директории (виж. **Фиг. 6**) са:

/bin – съдържа основните програми на операционната система, без които тя не би функционирала правилно. Повечето програми от тази директория може да се изпълняват от всички потребители. Тя е присъства и в пътя по подразбиране на всички потребители.

/boot – съдържа ядрото на операционната система, както и други файлове, необходими за правилното функциониране на ядрото.

/dev – съдържа файловете отговарящи за хардуерните устройства на компютъра. Файловете в тази директория са достъпни само за четене от потребителите.

/etc – съдържа глобалните конфигурационни файлове на операционната система и нейните програми. Някои от програмите може да пазят конфигурационни файлове и на други места.

/home – съдържа домашните директории на потребителите. Тук за всеки потребител се създава отделна директория в която може да се съхраняват конфигурационни файлове или документи от различен тип.

Специфичното в тези директории е начина на разграничаването на конфигурационните файлове и поддиректории от обикновените, а именно чрез добавяне на точка в началото им. Повечето файлови мениджъри скриват файл или директория, който започва с точка.

/lib – директорията съдържа най-важните споделени библиотеки на операционната система – glibc-solibs. Тези библиотеки се наричат споделени (shared object), защото повечето програми ги използват при изпълнението си.

/lib/modules – също много важна директория. Тук се съхраняват модулите на ядрото, т.е. драйверите на операционната система. Туй като всяка Линукс система може да има повече от едно ядро, то в тази директория се създават поддиректории с номера на всяко едно инсталирано ядро.

/mnt – незадължителна системна директория. Тя се създава автоматично и нейната цел е обединяването на всички монтирани устройства на едно място. Линукс обаче не ви ограничава да монтирате устройства в други директории.

/opt – тук повечето дистрибуции инсталират графичните среди като KDE и GNOME.

/proc – това е специална директория в която няма реални файлове. Посредством тази директория вие имате достъп до различни параметри на ядрото, както и до различна информация пряко предоставяна от ядрото на операционната система.

/root – домашната директория на свръхпотребителя. За обикновения потребител тази директория е заключена и той няма никакъв достъп до файловете в нея.

/sbin – важна системна директория с множество програми за управление на операционната и файловата система. Тази директория не е включена в пътя по подразбиране на обикновения потребител. Всички програми в тази директория искат root привилегии, за да работят.

/tmp – директория в която програмите записват временните си файлове. Тя има неограничен достъп за всички потребители. В нея обаче може да има поддиректории създадени от програми, които да са недостъпни за обикновените потребители.

/usr – директория с множество поддиректории. Тук се пазят повечето инсталирани програми, помощната информация и др.

/usr/X11R6, /usr/X11 – в тази директория се пазят файловете на графичния сървър, шрифтовете му и неговите библиотеки.

/usr/bin – съдържа бинарните файлове на инсталираните програми, които не са важни за нормалното функциониране на операционната система.

/usr/doc – част от документацията на програмите. Тук се съхранява Linux-HOWTO.

/usr/include – тази директория е необходима само ако ще се инсталират програми от изходен код. Тук се пазят заглавните C файлове на инсталираните библиотеки и програми, които са необходими за инсталиране на други програми.

/usr/info – част от документацията на Линукс. Достъпа до нея се осъществява чрез специална програма – info.

/usr/local – директория в която по подразбиране се инсталират програмите компилирани от изходен код. Тази директория копира структурата на /usr.

/usr/man – най-мащабната част от Линукс документацията. Тук се съхранява ръководство (manual page) за почти всички инсталирани програми.

Достъпа до тази информация се осъществява с програмата man.

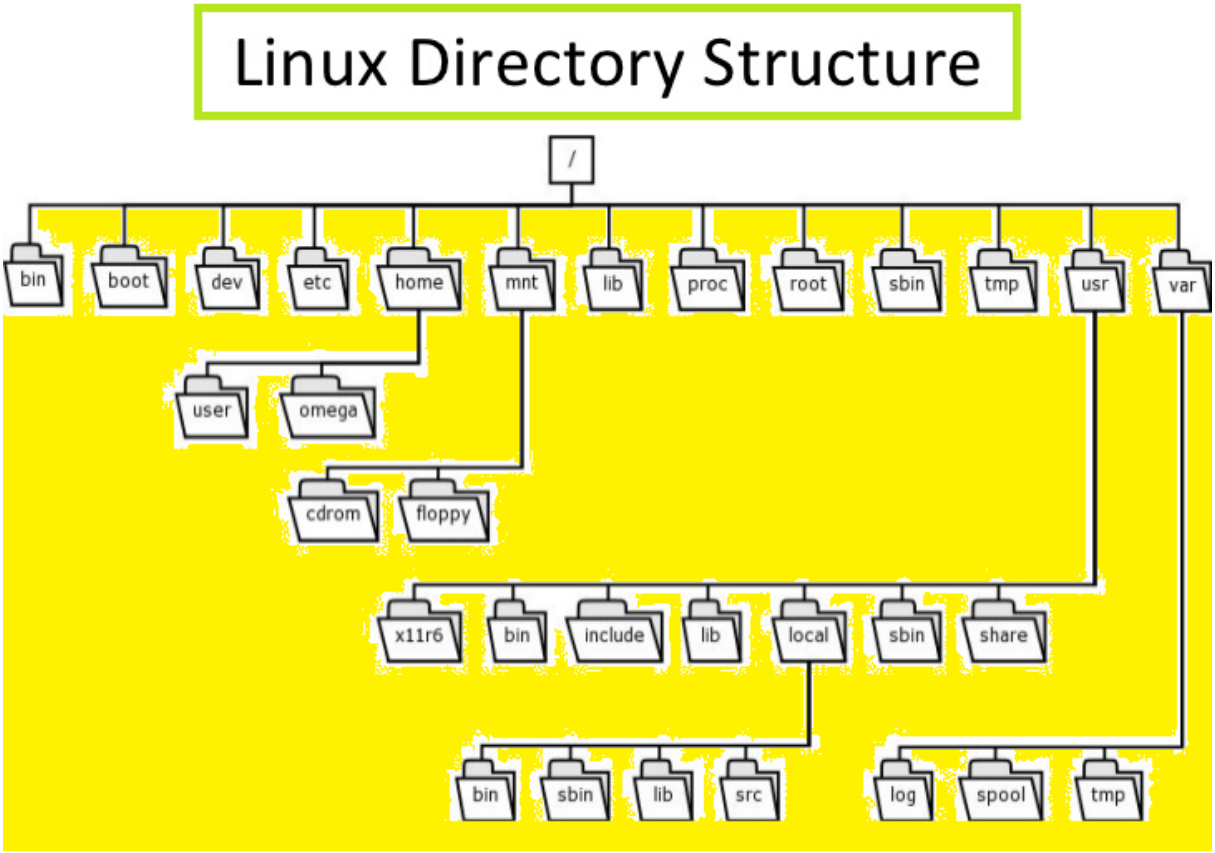
/usr/lib – важна директория, в която се съхраняват повечето инсталирани библиотеки. Тези библиотеки не са необходими за работата на операционната система, а за работата на допълнително инсталирания софтуер.

/usr/share – също важна директория. Тук програмите инсталират файловете, необходими за тяхната правилна работа.

/usr/src – в тази директория се пази изходният код на Линукс ядрото. Той е необходим само когато ядрото ще се прекомпилира. И тук подобно на **/lib/modules** може да има код на повече от едно ядро. Те се съхраняват в директории, различаващи се с номера на версията. В директорията /usr/src присъства и линк linux, който сочи към една от директориите съдържащи изходен код.

/var – системна директория съдържаща предимно логове на програмите и система, както и опашка с отложените или предстоящи задачи за автоматично изпълнение. Директория съдържа и друга информация, като

пристигнали писма, файлове заключващи дадени процеси и сокети на работещи в момента програми.



Фиг. 6. Структура на директориите в Линукс

В повечето директории обикновения потребител има права за четене, но не и за запис. Някои директории (като директорията /root и някои от директориите с логове) обикновения потребител няма никакви права. Обикновения потребител има пълни права единствено над файловете в собствената му директория и над собствените права в /tmp директорията. Тази организация затруднява значително инсталирането на зловредни програми (например вируси), тъй като те ще имат достъп единствено до файловете на потребителя, който ги изпълнява.

Досега многократно се спомена понятието права на потребителя и сега ще изясним какво е това право на потребителя на извърши определено действие, видовете права и това как се задават. Във всяка Линукс система има няколко вида потребители организирани в групи. Всеки файл може да задава три вида права за потребителя, три за групата и три за всички останали.

Тези права са:

право за четене (r)

право за запис (w)

и право за изпълнение (x)

Ето казаното в таблична форма:

Собственик -> rwx **Група -> rwx** **Всички останали -> rwx**

Правата на всеки файл може да се видят с командата **ls -l**.

Освен тези права файла може да има още един флаг – **SUID** и **GUID**. Файл с такива флагове се изпълнява с привилегиите на потребителя или с привилегиите на групата, която изпълнява файла. Тези флагове се считат за много опасни и тяхното използване трябва да се ограничава.

Командите, които променят собственика и/или групата на файла и неговите права са съответно chown и chmod. Ако трябва да се променят правата на test.txt, така че да бъде изпълним от всички, то трябва да се напише следното:

#chmod +x ./test.txt

Синтаксиса на командата е много прост: + включва, а – изключва даден флаг. Чрез r, w и x се уточнява кой флаг трябва да се промени.

По този начин се променя даденото право глобално. А ако трябва да се промени само за собственика или групата например? В този случай пред знака + или – се постява опция чии права трябва да се променят, като опциите са следните:

u – собственика(user)

g – групата(group)

o – всички останали (others)

a – всички (all), като това е опцията по подразбиране.

Ето и пример за премахване на правото за изпълнение от файла за всички останали потребители.

#chmod o-x ./test.txt

Командата позволява свободно комбиниране на опциите, например:

#chmod u+rwx ./test.txt

Начина на образуване на правата е следния:

4 – четене

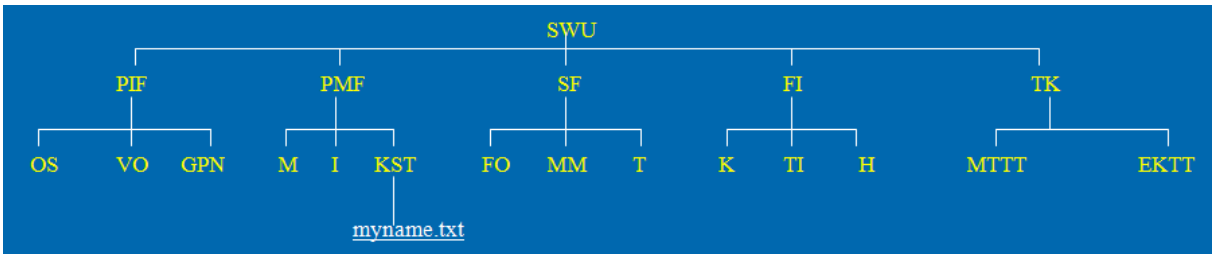
2 – запис

1 – изпълнение/търсене

С лед това е достатъчно да се съберат числата отговарящи за съответните права. Например право за четене и запис се дава като се съберат 4+2 =6.

Упражнения:

1. *Да се създаде следната дървовидна структура от директории в директорията home/username/вашият_фак_номер, като се*



използват съответните команди:

Файлът myname.txt да съдържа трите ви имена.

2. *Да се премахне правото за запис в файла myname.txt за потребителя student.*
3. *Да се създадат три файла с имена file1.txt, file2.txt и file3.sh в директорията home/username/вашият_фак_номер. Да се модифицират правата върху файловете по следния начин:*

```
-r--rw-rw-  1 student  users  file1.txt
-rw-r--r--  1 student  users  file2.txt
-rwxrw-r--  1 student  users  file3.sh
```

4. *Да се създаде файл с име description.txt в директорията home/username/вашият_фак_номер, в който да се опишат функциите на директориите /bin, /home, /dev и /usr.*
5. *Да се изведен на екрана съдържанието на файла description.txt.*
6. *Да се направи справка за съдържанието на директория student. Да се определи броя на директориите и броя на файловете в нея. Да се създаде файл с име count.txt в директорията home/username/вашият_фак_номер който съдържа техния брой.*

Писане на скриптове за BASH шел

Общи сведения

Както всички шелове, който може да намерите за Linux, BASH (Bourne Again SHell) е не само отличен команден интерпретатор но и език за писане на скриптове. Шел (Shell) скриптовете ви позволяват максимално да използвате възможностите на шел интерпретатора и да автоматизирате множество задачи. Много от програмите, който може да намерите за Linux в последно време са шел скриптове. Ако искате да разберете как те работят или как може да ги редактирате е важно да разберете синтаксиса и семантиката на BASH шела. В допълнение познаването на bash езика ви позволява да напишете ваши собствени програми, който да изпълняват точно това което искате.

Програмиране или писане на скрипове? Хората който до сега не са се занимавали с програмиране обикновено не разбират разликата между програмен и скрипт език. Програмните езици обикновено са по-мощни и по-бързи от скрипт езиците. Примери за програмни езици са C, C++, и Java. Програмите който се пишат на тези езици обикновено започват от изходен код (source code) - текст който съдържа инструкции за това как окончателната програма трябва да работи, след което се компилират до изпълним файл. Тези изпълними файлове не могат лесно да бъдат адаптирани за различни операционни системи (ОС). Например ако сте написали програма на C за Linux изпълнимият файл няма да тръгне под Windows XP. За да можете да я използвате тази програма се налага да прекомпилирате изходния код под Windows XP. Скриптовете (програмите писани на скрипт езици) също започват от изходен, но не се компилират в изпълними файлове. При тях се изполва интерпретатор, който чете инструкциите от изходния код и ги изпълнява. За жалост, поради това че

интерпретатора трябва да прочете всяка команда преди да я изпълни, интерпретираните програми вървят като цяло по-бавно спрямо компилираните. Основното предимство на скриптовете е, че лесно могат да бъдат пренаписани за други ОС стига да има интерпретатор за тази ОС. bash е скрипт език. Той е идеален за малки програми. Други скрип езици са Perl, Lisp, и Tcl.

Писането на собствени шел скриптове изисква от вас да знаете най-основните команди на Linux. Трябва да знаете например как да копирате, местите или създавате нови файлове. Едно от нещата което е задължително да знаете е как да работите с текстов редактор. За Linux има множество текстови редактори най-разпространените от който са vi, emacs, pico, nano, mcedit.

Работа с текстовия редактор nano

Nano (виж **Фиг. 7**) е текстов редактор използващ бързи клавиши за основните действия. Някои от тях са:

Ctrl+O - запис на файла.

Ctrl+R - вмъкване съдържанието на друг файл след позицията на курсора. Използва се за отваряне на файлове.

Ctrl+G - показва помощта.

Ctrl+X - изход от програмата.

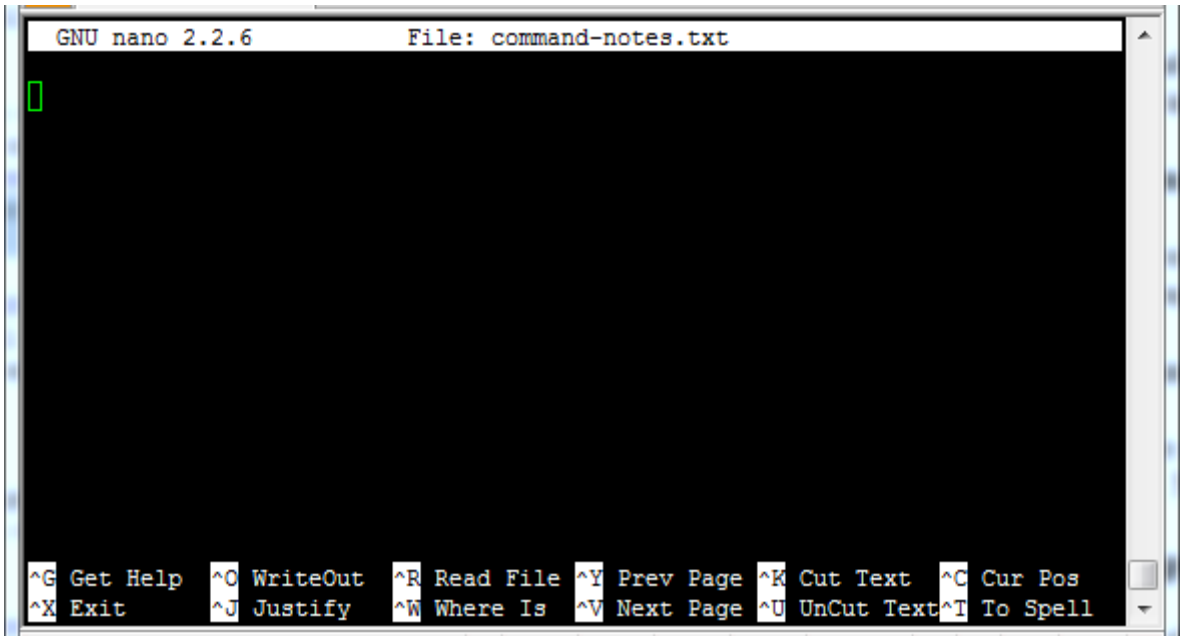
Ctrl+W - търсене.

Ctrl+C - текуща позиция на курсора.

Ctrl+K - изтрива текущия ред и го запазва в паметта.

Ctrl+U - поставя текста от паметта на текущия ред.

Ако имате желание да научите повече за този редактор вижте вградената помощ или прочетете ръководството на адрес http://www.nano-editor.org/dist/v2.1/nano.html



Фиг. 7. Изглед на текстовия редактор nano

Вашият първи BASH скрипт

Първият скрипт който ще напишете е класическата "Hello World" програма. Тази програма изпечатва единствено думите "Hello World" на екрана. Отворете текстовият редактор и напишете:

```
#!/bin/bash
echo "Hello World"
```

Първият ред от програмата казва че ще използваме bash интерпретатора. В този случай bash се намира в /bin директорията. Ако bash се намира в друга директория на вашата система тогава направете необходимите премени в първия ред. Изричното споменаване на това кой интерпретататор ще изпълнява скрипта е много важно, тъи като той казва на Linux какви инструкции могат да бъдат използвани в скрипта. Следващото нещо което трябва да направите е да запишете файла под

името hello.sh. Остава само да направите файла изпълним. За целта пишете:

$ chmod 700 ./hello.sh

Прочетете упътването за използване на chmod командата ако не знаете как да променяте правата на даден файл. След като сте направили програмата изпълнима я стартирайте като напишете:

$ ./hello.sh

Резултатът от която ще бъде следният надпис на екрана

Hello World

Това е! Запомнете последователноста от действия - писане на кода, записване на файла с кода, и променянето на файла в изпълним с командата chmod.

Какво точно направи вашата първа програма? Тя изпечата думите "Hello World" на екрана. Но как програмата го направи това? С помоща на команди. Единственият ред с команди беше echo "Hello World". И коя е командата? echo. Командата echo изпечатва всичко на екрана, което е получила като свой аргумент.

Аргумент е всичко което е написано след името на командата. В нашият случай това е "Hello World". Когато напишете ls /home/root, командата е **ls** а нейният аргумент е /home/root.

Друга команда за изпечатване е printf. Командата printf позволява повече контрол при изпечатване на информацията, особено ако сте запознати с програмния език C. Фактически същият резултат от нашият скрипт можем да постигнем просто ако напишем в командния ред:

$ echo "Hello World"
Hello World

Както виждате може да използвате Linux команди при писането на шел скриптове. Вашият bash шел скрипт е колекция от различни програми, специално написани заедно за да изпълнят конкретна задача.

Пример:

Ще напишем скрипт с който да преместим всички файлове от дадена директория в нова директория, след което ще изтрием новата директория заедно със нейното съдържание и ще я създадем отново. Това може да бъде направено със следната последователност от команди:

$ mkdir garbage
*$ mv * garbage*
$ rm -rf garbage
$ mkdir garbage

Вместо да пишем последователно тези команди можем да ги запишем във файл:

```
#!/bin/bash
mkdir garbage
mv * garbage
rm -rf garbage
mkdir garbage
echo "Deleted all files!"
```

Запишете файла под името clean.sh. След като стартирате clean.sh той ще премести всички файлове в директория *garbage*, след което ще изтрие директорията заедно със съдържанието и ще я създаде отново. Дори ще изпечата съобщение на екрана, когато свърши със тези действия. Този

пример показва и как да автоматизирате многократното писане на последователности от команди.

Коментари във скрипта

Коментарите ви помагат да направите вашата програма по-лесна за разбиране. Те не променят нищо в изпълнението на самата програма . Те се използват единствено за да може вие да ги четете. Всички коментари в bash започват със символа: "#", с изключение на първия ред (#!/bin/bash). Първият ред не е коментар. Всички редове след първия който започват със "#" са коментари. Вижте кода на следния скрипт:

```
#!/bin/bash
# tazi programa broi chislata ot 1 do 15:
for i in 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15; do
   echo $i
done
```

Дори и да не знаете bash, вие веднага може да се ориентирате какво прави скрипта, след като прочетете коментара. Добре е при писане на скриптове да използвате коментари. Ще откриете, че ако ви се наложи да промените някоя програма която сте писали преди време, коментарите ще ви бъдат от голяма полза.

Променливи

Променливите, най-общо казано, са "кутии" който съхраняват информация. Вие може да използвате променливи за много неща. Те ви помагат да съхранявате информацията която е въвел потребителя, аргументи или числова информация. Погледнете този код:

```
#!/bin/bash
x=15
echo "Stoinosta na promenliwata x e $x"
```

Всичко което се случва е да присвоим на променливата x стойност 15 и да я отпечатаме тази стойност с комадата echo. echo "Stoinosta na promenliwata x e $x" изпечатва текущата стойност на x. Когато давате стойност на дадена променлива не трябва да има шпации между нея и "=". Ето какъв е синтаксиса:

име_на_променлива=стойност

Стойността на променливата можем да получим като поставим символа "$" пред нея. В конкретния случай за да изпечатаме стойността на x пишем echo $x.

Има два типа променливи - локални променливи и променливи на обкръжението (environment). Променливите на обкръжението се задават от системата и информация за тях може да се получи като се използва env командата.

Например:

$ echo $SHELL

Ще отпечата

/bin/bash

Което е името на шела който използваме в момента. Променливите на обкръжението са дефинирани в /etc/profile и ~/.bash_profile. С echo командата можете лесно да проверите текущата стойност на дадена променлива, била тя от обкръжението или локална. Ако все още се чудите

защо са ни нужни променливи следната програма е добър пример който илюстрира тяхното използване:

```
#!/bin/bash
echo "Stojnosta na x e 10."
echo "Az imam 10 moliwa."
echo "Toj mi kaza che stojnosta na x e 10."
echo "Az sym na 10 godini."
echo "Kak taka stojnosta na x e 10?"
```

Да предположим че в един момент решите да промените стойноста на x от 10 на 8. Какво трябва да направите? Трябва да премените навсяскъде в кода 10 с 8. Да но има редове в който 10 не е стойноста на x. Трябва ли да променяме и тези редове? Не защото те не са свързани с x. Не е ли объркващо? По надолу е същия пример само че се използват променливи:

```
#!/bin/bash
x=10     # stoinosta na promenliwata x e 10 e x
echo "Stoinosta na  x e $x."
echo "Az imam 10 moliwa."
echo "Toj mi kaza che stojnosta na x e $x."
echo "Az sym na 10 godini."
echo "Kak taka stojnosta na x e $x?"
```

В този пример $x ще изпечата текущата стойност на x, която е 10. По този начин ако искате да промените стойноста на x от 10 на 8 е необходимо единствено да замените реда в който пише x=10 с x=8. Другите редовете няма да бъдат променени. Променливите имат и дуги полезни свойства както ще се убедите сами в последствие.

Упражнения:

1. *Да се създаде директория с име вашият_фак_номер в основната директория home/student*
2. *Да се напише прост скрипт с име zad1 в директорията home/student/вашият_фак_номер, които извършва следните действия:*

   - *извежда съобщение „Suzdavam failovete ...”*
   - *извежда празен ред*
   - *създава директория с име име_на _вашата_специалност в директорията home/student/вашият_фак_номер*
   - *създава в директорията име_на _вашата_специалност файловете: name.txt (съдържащ вашето име), kurs.txt (съдържащ курса) и number.txt (съдържащ вашия факултетен номер).*
   - *ивежда празен ред*
   - *ивежда събщение „Gotovo”*

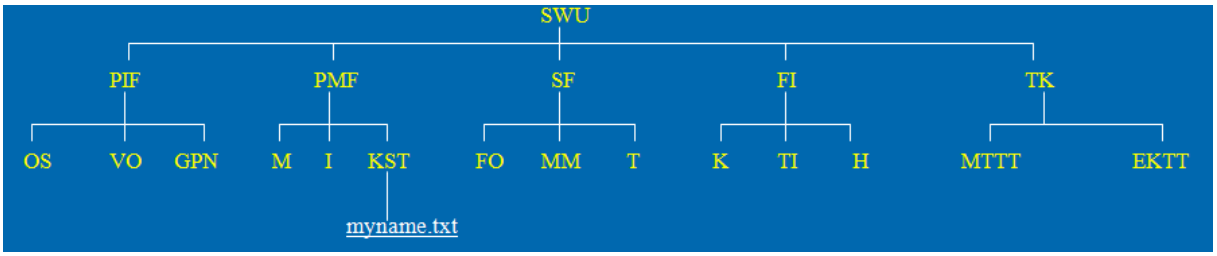
3. *Да се напише прост скрипт с име zad2 в директорията home/student/вашият_фак_номер, които извършва следните действия:*

   - *извежда съобщение „student”*
   - *извежда празен ред*
   - *извежда съобщение „name”*
   - *показва съдържанието на файла name.txt, създаден в задача 1*
   - *извежда съобщение „kurs”*
   - *показва съдържанието на файла kurs.txt, създаден в задача 1*
   - *извежда съобщение „number”*
   - *показва съдържанието на файла number.txt, създаден в задача 1*

4. *Да се напише прост скрипт с име zad3 в директорията home/student/вашият_фак_номер, които извършва следните действия:*

   - *премества файловете name.txt, kurs.txt и number.txt от директория име_на_вашата_специалност в директория вашият_фак_номер намираща се в основната директория home/student*
   - *извежда съобщение „ failovete sa premesteni v direktoriq students!”*
   - *изтрива директорията име_на_вашата_специалност*
   - *извежда съобщение „direktoriqta kst e iztrita !”*

5. *Да се напише прост скрипт с име zad4 в директорията home/student/вашият_фак_номер, която създава дървото:*



Файлът myname.txt да е празен или да съдържа трите ви имена.

Аритметически операции с BASH

Кавички

Кавичките играят голяма роля в шел програмирането. Има три различни вида: двойни кавички: ", единична кавичка: ', и обратно наклонена кавичка: `. Различават ли се те една от друга ? - Да.

Обикновено използваме двойната кавичка, за да обозначим с нея низ от символи и да запазим шпацията. Например, "Този низ съдържа шпации.". Низ заграден от двойни кавички се третира като един аргумент. Вземете следният скрипт за пример:

$ mkdir hello world

$ ls -F

hello/ world/

Създадохме две директории. Командата mkdir взе думите hello и world като два аргумента, създавайки по този начин две директории. А какво ще се случи сега:

$ mkdir "hello world"

$ ls -F

hello/ hello world/ world/

Създадохме една директория, състояща се от две думи. Двойните кавички направиха командата да третира двете думи като един аргумент . Без тях mkdir щеше да приеме hello за първи аргумент и world за втори.

Единична кавичка се използва обикновено, когато се занимаваме с променливи. Ако една променлива е заградена от двойни кавички, то нейната стойност ще бъде оценена. Но това няма да се случи ако използваме единични кавички. За да ви стане по-ясно разгледайте следният пример:

#!/bin/bash

x=5 # stojnosta na x e 5

 # izpolzwame dwojni kawichki

echo "Using double quotes, the value of x is: $x"

 # izpolzwame edinichni kawichki

echo 'Using forward quotes, the value of x is: $x'

Виждате ли разликата ? Може да използвате двойни кавички, ако не смятате да слагате и променливи в низа, който ще заграждат кавичките. Ако се чудите дали единичните кавички запазват шпациите в даден низ, както това правят двойните кавички, погледнете следния пример:

$ mkdir 'hello world'

$ ls -F

hello world/

Обратно наклонените кавички коренно се различават от двойните и единични кавички. Те не се използват, за да запазват шпациите. Ако си спомняте по-рано използвахме следния ред:

x=$(expr $x + 1)

Както вече знаете резултатът от командата expr $x + 1 се присвоява на променливата x. Същият този резултат може да бъде постигнат, ако използваме обратно наклонени кавички:

x=`expr $x + 1`

Кой от начините да използвате? Този, който предпочитате. Ще откриете, че обратно наклонените кавички са по-често използвани от $(...). В много случай обаче $(...) прави кода по-лесен за четене. Вземете предвид това:

$!/bin/bash

echo "I am `whoami`"

Езикът bash ви позволява да смятате различни аритметични изрази. Както вече видяхте, аритметическите операции се използват посредством

командата expr. Тази команда, обаче, както и командата true се смята, че са доста бавни. Причината за това е, че шел интерпретаторът трябва всеки път да стартира true и expr командите, за да ги използва. Като алтернатива на true посочихме командата ":". А като алтернатива на expr ще извозваме следния израз $((...)). Разликата със $(...) е броя на скобите. Нека да опитаме:

```
#!/bin/bash
x=8 # stojnosta na x e 8
y=4 # stojnosta na y e 4
# sega shte priswojm sumata na promenliwite x i y na promenliwata z:
z=$(($x + $y))
echo "Sum na $x + $y e $z"
```

Ако се чувствате по-комфортно с expr вместо $((...)), тогава го използвайте него.

С bash можете да събирате, изваждате, умножавате, делите числа, както и да делите по модул. Ето и техните символи:

ДЕЙСТВИЕ	Събиране	Изваждане	Умножение	Деление	Деление по модул
ОПЕРАТОР	+	-	*	/	%

Всеки от Вас би трябвало да знае какво правят първите четири оператора. Ако не знаете какво означава деление по модул - това е остатъкът при деление на две стойности. Ето и малко bash аритметика:

```
#!/bin/bash
x=5 # initialize x to 5
y=3 # initialize y to 3

add=$(($x + $y)) # sumiraj x sys y i priswoj rezultata na promenliwata add
```

```
sub=$(($x - $y)) # izwadi ot x y i priswoj rezultata na promenliwata sub
mul=$(($x * $y)) # umnozhi x po y i priswoj rezultata na promenliwata mul
div=$(($x / $y)) # razdeli x na y i priswoj rezultata na promenliwata div
mod=$(($x % $y)) # priswoj ostatyka pri delenie na x / y na promenliwata mod

# otpechataj otgoworite:
echo "Suma: $add"
echo "Razlika: $sub"
echo "Projzwedenie: $mul"
echo "Quotient: $div"
echo "Ostatyk: $mod"
```

Отново горният код можеше да бъде написан с командата expr. Например вместо add=$(($x + $y)), щяхме да пишем add=$(expr $x + $y), или add=`expr $x + $y`.

Четене на информация от клавиатурата

Сега вече идваме към интересната част. Вие можете да направите Вашите програми да си взаимодействат с потребителя и потребителят да може да си взаимодейства с програмата. Командата, която Ви позволява да прочетете каква стойност в въвел потребителят е read. read е вградена в bash команда, която се използва съвместно с променливи, както ще видите:

```
#!/bin/bash
# gets the name of the user and prints a greeting
echo -n "Enter your name: "
read user_name
echo "Hello $user_name!"
```

Променливата тук е user_name. Разбира се, може да я наречете, както си искате. read ще Ви изчака да въведете нещо и да натиснете клавиша

ENTER. Ако не натиснете нищо, командата read ще чака, докато натиснете ENTER . Ако ENTER е натиснат, без да е въведено нещо, то ще продължи изпълнението на програмата от следващия ред.

Ето и пример:

```
#!/bin/bash
# gets the name of the user and prints a greeting
echo -n "Enter your name: "
read user_name
# the user did not enter anything:
if [ -z "$user_name" ]; then
echo "You did not tell me your name!"
exit
fi
echo "Hello $user_name!"
```

Ако потребителят натисне само клавиша ENTER, нашата програма ще се оплаче и ще прекрати изпълнението си. В противен случай ще изпечата поздравление. Четенето на информацията, която се въвежда от клавиатурата е полезно, когато правите интерактивни програми, които изискват потребителят да отговори на конкретни въпроси.

Упражнения:

1. *Да се създаде директория с име вашият_фак_номер в основната директория home/student*

2. *Да се напише скрипт с име zad1 в директорията home/student/вашият_фак_номер, който:*

- *Извежда съобщение „Zadavam stoinosti na x, y и z”*

- *Задава стойности съответно 1, 2 и 3 на променливите x, y, z.*

- *Извежда съобщение „Stoinostite na x, y, z sa:”*

- *Извежда съобщение „stoinosta na x e:” съответната стойност*

- *Извежда съобщение „stoinosta na y e:” съответната стойност*

- *Извежда съобщение „stoinosta na z e:” съответната стойност*

3. *Да се напише скрипт с име zad2 в директорията home/student/вашият_фак_номер, които:*

- *Задава стойности на 2 на x*

- *Задава стойности на 4 на y*

- *Присвоява сумата на променливите x и y, на променлива z*

- *Извежда съобщението „Sumata na promenliwite x i y e:”*

- *Извежда на нов ред съобщението „z = ” и отпечатва съответната стойност.*

4. *Да се напише скрипт с име zad3 в директорията home/student/вашият_фак_номер, които:*

- *Извежда съобщение “Molq vuvedete cenata na stokata v stotinki”*

- *Прима вход от клавиатурата (задавате цена на стоката в стотинки)*

- *Пресмята стойноста на стоката в левове*
- *Извежда съобщение „Cenata na stokata v lv e:”*
- *Отпечатва съответната жена и „lv”*

5. *Да се напише скрипт с име zad4 в директорията home/student/вашият_фак_номер, които:*

- *Извежда съобщение “Vavedete ocenkite po predmeti”*
- *Извежда името на съответния предмет (за предметите операционни системи, web дизайн, бази данни, компютърни мрежи, компютърни архитектури) и приема стойност на оценката от клавиатурата.*
- *Изчислява средният успех*
- *Извежда съобщение „Sredniqt vi uspeh e:” и отпечатва съответната стойност*

6. *Да се напише скрипт с име zad5 в директорията home/student/вашият_фак_номер, които:*

- *Извежда съобщение „Vavedete stoinosti za: a, b, c, d”*
- *Приема съответните стойности от клавиатурата*
- *Пресмята стойността на израза: ((a + b)*c + (a + b)*d + a + b + d +c)/2*
- *Отпечатва резулата от пресмятането на горния израз*

- *ДА СЕ ТЕСТВА СЪС СТОЙНОСТИ : 4, 8, 6, 2, СЪОТВЕТНО ЗА a, b, c, d. РЕЗУЛТАТ ТРЯБВА ДА БЪДЕ 58.*

7. Да се напише скрипт с име zad6 в директорията home/student/вашият_фак_номер, които пресмята лицето на триъгълник по зададена страна на триъгълника и височината към нея.

- *Вход от клавиатурта: дължина на една от страните на триъгълника в сантиметри и дължина на височината на триъгълника спусната към съответната страна на триъгълника. Съответните стойности да бъдат цели числа.*

- *Изход на екрана: лицето на триъгълника в квадратни сантиметри.*

Формулата за изчисляване на лице на триъгълник е : $S = \frac{a * h_a}{2}$

Условни оператори в BASH

Условни оператори

Условните оператори ви позволяват вашата програма да "взема решения" и я правят по-компактна. Което е по-важно с тях може да проверявате за грешки. Всички примери до сега започваха изпълнението си от първия ред до последния без никакви проверки. За пример:

```
#!/bin/bash
cp /etc/foo .
echo "Done."
```

Тази малка шел програма копира файлът /etc/foo в текущата директория и изпечатва "Done" на екрана. Тази програма ще работи само при едно условие. Трябва да има файл /etc/foo. В противен случай ще се получи следния резултат:

```
$ ./bar.sh
cp: /etc/foo: No such file or directory
Done.
```

Както виждате имаме проблем. Не всеки който стартира вашата програма има файл /etc/foo на системата си. Ще бъде по-добре, ако вашата програма проверява дали файла /etc/foo съществува и ако това е така да продължи с копирането, в противен случай да спре изпълнението. С "псевдо" код това изглежда така:

```
if /etc/code exists, then
   copy /etc/code to the current directory
   print "Done." to the screen.
```

otherwise,

print "This file does not exist." to the screen

exit

Може ли това да бъде направено с bash? Разбира се! В bash условните оператори са: **if**, **while**, **until**, **for**, и **case**. Всеки оператор започва с ключова дума и завършва с ключова дума. Например **if** оператора започва с ключовата дума if, и завършва с **fi**. Условните оператори не са програми във вашата система. Те са вградени свойства на bash.

if ... else ... elif ... fi

Е един от най-често използваните условни оператори. Той дава възможност на програма да вземе решения от рода на "направи това ако (**if**) това условие е изпълнено, или (**else**) прави нещо друго". За да използвате ефективно условния оператор if трябва да използвате командата test. test проверява за съществуване на файл, права, подобия или разлики.

Ето програмата bar.sh:

```
#!/bin/bash
if test -f /etc/foo
then
   # file exists, so copy and print a message.
   cp /etc/foo .
   echo "Done."
else
   # file does NOT exist, so we print a message and exit.
   echo "This file does not exist."
```

exit

fi

Забележете че редовете след **then** и **else** са малко по-навътре. Това не е задължително, но се прави с цел програмата да бъде по-лесна за четене. Сега стартирайте програмата. Ако имате файл **/etc/foo**, тогава програмата ще го копира в текущата директория, в противен случай ще върне съобщение за грешка. Опцията **-f** проверява дали това е обикновен файл. Ето списък с опциите на командата **test:**

-d проверява дали файлът е директория;

-e проверява дали файлът съществува;

-f проверява дали файлът е обикновен файл;

-g проверява дали файлът има SGID права;

-r проверява дали файлът може да се чете;

-s проверява дали размерът на файла не е 0;

-u проверява дали файлът има SUID права;

-w проверява дали върху файлът може да се пише;

-x проверява дали файлът е изпълним.

else се използва ако искате вашата програма да направи нещо друго, ако първото условие не е изпълнено. Има и ключова дума **elif**, която може да бъде използвана вместо да пишете друг if вътре в първия if. **elif** идва от английското "**else if**". Използва се когато първото условие не е изпълнено и искате да проверите за друго условие.

Ако не се чувствате комфортно с **if** и **test** синтаксиса, който е :

if test -f /etc/foo

then

тогава може да използвате следния вариант:

if [-f /etc/foo]; then

Квадратните скоби формират test командата. Ако имате опит в програмирането на C този синтакс може да ви се стори по-удобен. Забележете, че трябва да има разстояние след отварящата квадратна скоба и преди затварящата. Точката и запетаята: ";" казва на шела че това е края на командата. Всичко след ";" ще бъде изпълнено сякаш се намира на следващия ред. Това прави програмата малко по-четима. Можете разбира се да сложите **then** на следващия ред.

Когато използваме променливи с test е добре да ги заградим с кавички. Например:

if ["$name" -eq 5]; then

while ... do ... done

while оператора е условен оператор за цикъл. Най-общо казано, това което прави е "**while** (докато) това условие е вярно, **do** (изпълни) командите **done** ". Нека да видим следния пример:

```
#!/bin/bash
while true; do
  echo "Press CTRL-C to quit."
done
```

true в действителност е програма. Това което прави тази програма е да се изпълнява безкрайно. Използването на true се смята, че забавя вашата програма, защото шел интерпретатора първо трябва да извика програмата и след това да я изпълни. Вместо това може да използвате командата ":":

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

```
while :; do
  echo "Press CTRL-C to quit."
done
```

Както виждате използваме **test** (записана като квадратни скоби) за да проверим състоянието на променливата x. Опцията **-le** проверява дали x е по-малко (**less**) или равно (**equal**) на 10. На говорим език това се превежда по следния начин "Докато (**while**) x е по-малко или равно на 10, покажи текущата стойност на x, и след това добави 1 към текущата стойност на x.". **sleep** 1 казва на програмата да спре изпълнението си за една секунда. Както виждате това което правим тук е да проверим за равенство. Ето списък с някой опции на test:

Проверка за равенства между променливите x и y, ако променливите са числа:

x **-eq** y Проверява дали x е равно на y

x **-ne** y Проверява дали x не е равно на y

x **-gt** y Проверява дали x е по-голямо от y

x **-lt** y Проверява дали x е по-малко от y

От горния пример единственият ред, който може да ви се стори по-труден за разбиране е следния:

```
x=$(expr $x + 1)
```

Това което прави този ред е да увеличи стойността на x с 1. Но какво значи **$(...)**? Дали е променлива? Не. На практика това е начин да кажете на шел интерпретатора, че ще изпълнявате командата expr **$x + 1**, и резултата

от тази команда ще бъде присвоен на x. Всяка команда която бъде записана в **$(...)** ще бъде изпълнена:

```
#!/bin/bash

me=$(whoami)

echo "I am $me."
```

Опитайте с този пример за да разберете какво се имам предвид. Горната програмка може да бъде написана по-следния начин:

```
#!/bin/bash

echo "I am $(whoami)."
```

Сами си решете кой от начините е по-лесен за вас. Има и друг начин да изпълните команда или да присвоите резултата от изпълнението на дадена команда на променлива. Този начин ще бъде обяснен по-нататък. За сега използвайте **$(...)**.

until ... do ... done

Условния оператор **until** е много близък до **while**. Единствената разлика е, че се обръща смисъла на условието и се взима предвид новото значение. Действието на **until** оператора е "докато (**until**) това условие не е вярно, изпълнявай (do) командите". Ето пример:

```
#!/bin/bash

x=0

until [ "$x" -ge 10 ]; do

   echo "Current value of x: $x"
```

```
x=$(expr $x + 1)

sleep 1

done
```

Разликата между **while** и **until** циклите е, че при **while** цикъла той се изпълнява докато условието е изпълнено, а при **until**, докато то не се изпълни.

Този код може би ви изглежда познат. Проверете го и вижте какво прави. **until** ще изпълнява командите докато стойноста на променливата x е по-голяма или равна на 10. Когато стойноста на x стане 10 цикълът ще спре. Ето защо последната стойност на x която ще се изпечата е 9.

for ... in ... do ... done

for се използва кога искате да присвоите на дадена променлива набор от стойности. Например можете да напишете програма, която да изпечатва 10 точки всяка секунда:

```
#!/bin/bash

echo -n "Checking system for errors"

for dots in 1 2 3 4 5 6 7 8 9 10; do

   echo -n "."

done

echo "System clean."
```

В случай, че не знаете опцията **-n** на командата **echo** спира автоматичното добавяне на нов ред. Пробвайте командата веднъж с **-n** опцията и веднъж без нея за да разберете за какво става дума.

Променливата dots преминава през стойностите от 1 до 10. Вижте следния пример:

```
#!/bin/bash

for x in paper pencil pen; do

    echo "The value of variable x is: $x"

    sleep 1

done
```

Когато стартирате програмата ще видите че x в началото ще има стойност paper, след което ще премине на следващата стойност, която е pencil, и след това pen. Когато свършат стойностите през който минава цикъла изпълнението му завършва.

Ето една доста полезна програма. Тя добавя .html разширение на всички файлове в текущата директория:

```
#!/bin/bash

for file in *; do

    echo "Adding .html extension to $file..."

    mv $file $file.html

    sleep 1

done
```

Ако не знаете "*" е "wild card character". Това ще рече "всичко в текущата директория", което в нашия случай представлява всички файлове

в тази директория. Променливата file ще премине през всички стойности, в този случай файловете в текущата директория. След което командата **mv** преименува стойностите на променливата file във такива с .html разширение.

case ... in ... esac

Условния оператор **case** е близък до **if** . За предпочитане е да се използва когато имаме голям брой условия който трябва да бъдат проверени. Вземете за пример следния код:

```
#!/bin/bash

x=5     # initialize x to 5

# now check the value of x:

case $x in

  0) echo "Value of x is 0."

    ;;

  5) echo "Value of x is 5."

    ;;

  9) echo "Value of x is 9."

    ;;

  *) echo "Unrecognized value."

esac
```

Оператора case ще провери стойност на x на коя от 3-те възможности отговаря. В този случай първо ще провери дали стойността на x е 0, след което ще провери за 5 и 9. Накрая ако никое от условия не е изпълнено ще се изпечата съобщението "Unrecognized value.". Имайте предвид, че "*" означава "всичко", и в този случай това означава "която и да е стойност различна от посочените". Ако стойността на x е различна от 0, 5, или 9, то тогава тя попада в случая "*". Когато използвате **case** всяко условие трябва да завършва с две ";". Може би се чудите защо да използвате **case** когато може да използвате **if**? Ето как изглежда еквивалентната програма написана с **if**. Вижте коя програма е по-бърза и по лесна за четене:

```
#!/bin/bash

x=5     # initialize x to 5

if [ "$x" -eq 0 ]; then

   echo "Value of x is 0."

elif [ "$x" -eq 5 ]; then

   echo "Value of x is 5."

elif [ "$x" -eq 9 ]; then

   echo "Value of x is 9."

else

   echo "Unrecognized value."

fi
```

Упражнения:

1. *Да се създаде директория с име вашият_фак_номер в основната директория home/student*
2. *Да се напише скрипт с име zad1 в директорията home/student/вашият_фак_номер, който проверява дали съществува файл с име readme.txt в основната директория home/student. Ако файлът съществува, да се изведе съобщение, че съществува и да се копира в директория home/student/students. Ако файлът не съществува да се изведе съобщение за това.*
3. *Да се напише скрипт с име zad2 в директорията home/student/вашият_фак_номер, който проверява дали файлът students, намиращ се в директорията home/student/ е директория и да се изведе съответното съобщение от резулата от проверката.*
4. *Да се напише скрипт с име zad3 в директорията home/student/вашият_фак_номер, който отпечатва числата от 1 до 20 през интервал от 2 секунди като се използва оператора while.*
5. *Да се напише скрипт с име zad4 в директорията home/student/вашият_фак_номер, който отпечатва числата от 1 до 20 през интервал от 2 секунди като се използва оператора until.*
6. *Да се напише скрипт с име zad5 в директорията home/student/вашият_фак_номер, който отпечатва числата от 1 до 20 през интервал от 2 секунди като се използва оператора for.*
7. *Да се напише скрипт с име zad6 в директорията home/student/вашият_фак_номер, който проверява стойноста на променлива x с оператора case, и отпечатва нейната стойност. Проверката да бъде за стойности на x: 1 3 6 9 10 и всики останали.*

Функции с BASH

Функции

Функциите правят скрипта по-лесен за поддържане. Най-общо казано, функциите разделят програмата на малки части. Функциите изпълняват действия, които Вие сте дефинирали и може да върне стойност от изпълнението си ако желаете. Преди да продължим, да видим един пример на шел програма, която използва функция:

```
#!/bin/bash

# functiqta hello() samo izpechatwa syobshtenie

hello()

{

echo "Wie ste wyw funkciq hello()"

}

echo "Izwikwame funkciqta hello()..."

# izwikwame hello() funkciqta wytre w shell skripta:

hello

echo "Weche izleznahte ot funkciqta hello()"
```

Опитайте се да напишете тази програма и да я стартирате. Единствената цел на функцията **hello()** е да изпечата съобщение. Функциите естествено могат да изпълняват и по-сложни задачи. В горния пример ние извикахме функцията **hello()** с този ред:

hello

Когато се изпълнява този ред bash интерпретатора претърсва скрипта за ред който започва с **hello()**. След което открива този ред и изпълнява съдържанието на функцията.

Функциите винаги се извикват чрез тяхното име. Когато пишете функция можете да започнете функцията с **function_name()**, както беше направено в горния пример, или да използвате думата **function** т.е **function function_name()**. Другият начин, по който можем да започнем нашата функция е **function hello()**:

function hello()

{

echo "Wie ste wyw funkciq hello()"

}

Функциите винаги започват с отваряща и затваряща скоба"**()**", последвани от отварящи и затварящи къдрави скоби: "**{...}**". Тези къдрави скоби бележат началото и края на функцията. Всеки ред с код затворен в тези скоби ще бъде изпълнен и ще принадлежи единствено на функцията. Функциите трябва винаги да бъдат дефинирани преди да бъдат извикани. Нека погледнем нашата програма, само че този път ще извикаме функцията преди да е дефинирана:

#!/bin/bash

echo "Izwikwame funkciqta hello()..."

 # call the hello() function:

```
hello

echo "Weche izleznahte ot funkciqta hello()"

# function hello() just prints a message

hello()

{

echo "Wie ste wyw funkciq hello()"

}
```

Ето какъв е резултатът, когато се опитаме да изпълним програмата:

```
$ ./hello.sh

Izwikwame funkciqta hello()...

./hello.sh: hello: command not found

Weche izleznahte ot funkciqta hello()
```

Както виждате, програмата върна грешка. Ето защо е добре да пишете Вашите функции в началото на скрипта или поне преди да ги извикате.

Ето друг пример как да използваме функции:

```
#!/bin/bash

# admin.sh - administrative tool

# function new_user() creates a new user account

new_user()
```

```
{

echo "Preparing to add a new user..."

sleep 2

adduser # run the adduser program

}

echo "1. Add user"

echo "2. Exit"

echo "Enter your choice: "

read choice

case $choice in

1) new_user # call the new_user() function

;;

*) exit

;;

esac
```

За да работи правилно тази програма, трябва да сте влезли като root, тъй като adduser е програма, която само root потребителят има право да изпълнява. Да се надяваме, че този кратък пример Ви е убедил в полезноста на фукциите.

Прихващане на сигнали

Може да използвате вградената команда trap, за да прихващате сигнали във Вашата програма. Това е добър начин да излезете нормално от програмата. Непример, ако имате вървяща програма при натискането на CTRL-C ще изпратите на програмата interrupt сигнал, който ще "убие" програмата. trap ще Ви позволи да прихване този сигнал и ще Ви даде възможност или да продължите с изпълнението на програмата, или да съобщите на потребителя, че програмата спира изпълнението си. trap има следния синтаксис:

trap dejstwie signal

dejstwie указва какво да искате да направите, когато прихванете даден сигнал, а **signal** е сигналът, който очакваме. Списък със сигналите може да откриете като пишете **trap -l**. Когато използвате сигнали във вашата шел програма пропуснете първите три букви на сигнала, обикновено те са **SIG.** Например, ако сигналът за прекъсване е **SIGINT**, във Вашата шел програма използвайте само **INT**. Можете да използвате и номера на сигнала. Номерът на сигнала **SIGINT** е 2.

Пробвайте следната програма:

```
#!/bin/bash

# using the trap command

# da se zipylni funkciqta sorry() pri natiskane na CTRL-C:

trap sorry INT

# function sorry() prints a message
```

```
sorry()

{

echo "I'm sorry Dave. I can't do that."

sleep 3

}

# count down from 10 to 1:

for i in 10 9 8 7 6 5 4 3 2 1; do

echo $i seconds until system failure."

sleep 1

done

    echo "System failure."
```

Сега, докато програмата върви и брои числа в обратен ред натиснете **CTRL-C**. Това ще изпрати сигнал за прекъсване на програмата. Сигналът ще бъде прихванат от **trap** командата, която ще изпълни ***sorry()*** функцията. Можете да накарате **trap** да игнорира сигнал като поставите "''" на мястото на действие. Можете да накарате **trap** да не прихваща сигнал като използвате "-".

Например:

```
# izpylni sorry() funkciqta kogato programa poluchi signal SIGINT:

trap sorry INT

# nakaraj programata da NE prihwashta SIGINT signala :
```

trap - INT

 # ne prawi nishto kogato se prihwane signal SIGINT:

trap '' INT

Когато кажете на **trap** да не прихваща сигнала, то програмата се подчинява на основното действие на сигнал, което в конкретния случай е да прекъсне програмата и да я "убие". Когато укажете **trap** да не прави нищо при получаване на конкретен сигнал, то програмата ще продължи своето действие, игнорирайки сигнала.

Упражнения:

1. *Да се създаде директория с име вашият_фак_номер в основната директория home/student*

2. *Да се напише скрипт с име zad1 в директорията home/student/вашият_фак_номер, в които да се реализира функция с име Print_Name, която отпечатва трите ви имена.*

3. *Да се напише скрипт с име zad2 в директорията home/student/вашият_фак_номер, в които да се реализира функция с име Input, която задава стоиност от клавиатурата на променлива n, функция Calculate, която изчислява факториела на въведеното число n и функция Output, която отпечатва резулатат от изчислението на n!. (Пресмятане на факториел: n! = n*(n-1)*(n-2)*...*2*1; 0! = 1)*

4. *Да се напише скрипт с име zad3 в директорията home/student/вашият_фак_номер, в които да се реализира функция с име Min, която намира най-малкото от пет въведени числа.*

5. Да се напише скрипт с име zad4 в директорията home/student/вашият_фак_номер, в които:

- *е реализирана функция с име Counter, която отпечатва числата от 1 до 20 през интервал от 2 секунди*

- *се прихваща сигнала interrupt при натискане на CTRL-C от клавиатурата и се извиква функция с име Choice, която пита потребителя дали желае да спре изпълнението на програмата или да продължи и в зависимост от въведения отговор "y\n" извършва съответното действие.*

Заключение

Основната разлика между Linux и Windows е принципа на отворения код, на базата на който се разработва Линукс ядрото. Софтуер, който се разпространява с отворен код спазва правилата дефинирани в Open Source лиценза.

Освен операционна система персоналният компютър се нуждае от голяма база приложен софтуер. Без голям брой различни по вид и функционалност приложения компютърът е практически неизползваем. При Windows и DOS част от тези приложения идват с операционната система, но при Линукс не е така. Това е така, защото Линукс е само ядрото на операционната система. Той може да управлява устройства и задачи, но сам по себе си е абсолютно неизползваем. Именно за това съществуват Линукс дистрибуции, които съдържат в себе си както ядрото на операционната система, така и приложен софтуер. Всяка една дистрибуция идва с базов набор от най-необходимите програми, като те са еднакви в повечето Линукс дистрибуции. Освен това много Линукс дистрибуции идват и с огромен набор от приложения, които включват браузъри, текстообработващи програми, офис пакети, среди за разработка на приложения, набор от графични среди и т.н.

Всяко Линукс ядро е многопотребителско и многозадачно. Това означава, че в един и същ момент на една Линукс машина може да се стартират множество процеси от множество потребители едновременно, без това да влияе по-някакъв начин на изпълнението на програмите. Тъй като Линукс ядрото е наистина многозадачно, то всяка програма се стартира в собствена област от паметта, като евентуален срив в нея не причинява срив в операционната система.

Литература

1. M. Sobell. A Practical Guide to Linux Commands, Editors, and Shell Programming, SECOND EDITION. Pearson Education, Inc., Boston, 2010.
2. M. Garrels. Introduction to Linux: A Hands on Guide, MIX, Bucharest, 2007.
3. N. Marsh. Introduction to the Command Line (Second Edition): The Fat Free Guide to Unix and Linux Commands, CreateSpace, North Charleston, 2010.
4. R. Rehman, C. Paul. The Linux development platform: configuring, using, and maintaining a complete programming environment, Prentice Hall PTR, United States, 2003.
5. W. Shotts. The Linux Command Line: A Complete Introduction, No Starch Press, San Francisco, 2012.
6. https://bg.m.wikibooks.org/wiki/Писане_на_скриптове_за_BASH_шел
7. http://sergi.blog.bg/technology/2008/01/23/bash-script-bg.155636
8. http://www.linux-bg.org/cgi-bin/y/index.pl?page=article&id=advices&key=319145325
9. http://www.linux-bg.org/cgi-bin/y/index.pl?page=article&id=advices&key=319145325&layout=clean
10. http://www.linux-bg.org/cgi-bin/y/index.pl?page=article&id=advices&key=318107036&layout=clean
11. http://www.linux-bg.org/cgi-bin/y/index.pl?page=article&id=advices&key=318107036&act=add_report&cmd=rate&minus=1