Командни файлове в LINUX

1. *Определение*: Командни файлове в Linux са файлове, които съдържат команди на операционна система, обръщение към изпълними файлове и Shell-команди. За да могат да бъдат изпълнени трябва да се маркират като изпълними. Това се извършва чрез командата **chmod**.

2. *Създаване на команден файл*

1 н-н: Чрез използване на командата **cat** в режим на пренасочване на изхода.

Пример: `cat > abc`

.....

`Ctrl+D`

Забележка: Използването на този начин не е много удобен за въвеждането на по-дълги скриптове, поради причината, че след преминаването на следващия ред с натискане на клавиша Enter, става невъзможно редактирането на предишния ред.

2 н-н: Чрез използване на текстов редактор – vi, mc, mcedit, nano и др.

3. *Задаване на права за изпълнение на създаден файл* (обявяването му за изпълним) – със командата **chmod**

I синтаксис: **chmod [потребител] <действие><достъп> <име на файл>**

потребител: (ако липсва настройката е за всички потребители)

u - собственик

g – членове на група, към която е причислен собственика

o – останалите потребители

a – всички потребители

действие:

\+ - разрешение за

\- - забрана за

достъп:

r – право за четене

w – право за запис

x – право за изпълнение

II синтаксис: **chmod <трицифрено осмично число> <име на файл>**

При този начин на използване на командата за всеки вид потребители се сформира по едно осмично число (0..7), образувано от сумата на цифрите, съответстващи на дадените права за достъп на съответния потребител.

owner/user			group			other		
r	w	x	r	w	x	r	w	x
4			4			4		
	2			2			2	
		1			1			1

Ако някое от правата липсва, то на негово място се счита 0 (нула), иначе стойностите са r=4, w=2, x=1.

Пример: За да се опишат атрибутите rwxr-x—x , съответно осмичното число е 751, където 7=4(r)+2(w)+1(x); 5=4(r)+0(w)+1(x); 1=0(r)+0(w)+1(x)

Тъй като в Linux няма значение дали файловете са с разширение или не, единствения начин да се укаже, че даден файл е изпълним е чрез командата **chmod**.

Например: **`chmod +x <име на файл>`** - дефинира файла като изпълним (дали той наистина е изпълним зависи от потребителя, но операционната система го интерпретира като такъв).

4. *Променливи на средата*

Както останалите програмни езици Shell обвивката позволява потребителя да дефинира променливи и да им присвоява стойности.

4.1. Име на променлива – идентификатор т.е. последователност от букви и цифри или символ за подчертаване, започваща с буква или знак за подчертаване.

Примери за правилни имена: `length, a5, HOMEDIR, _cflag`

Примери за неправилни имена: `5c5, .length, file_name`

4.2. Присвояване на стойности на променливи – няма дефиниране на типове на променливите. Действията със стойностите се извършва в контекста на операцията, в която се използват. За предпочитане е низовете да се заградят в двойни кавички.

Пример:
```
length=80
f_name=200
```

4.3. Цитиране на стойност на променлива – преди името на променливата се поставя знак $. За извеждане на стойност на променливи се използва се команда **echo**:

Пример:

```
length=80
echo $length
80
```

```
length=70
echo "The length is $length"
The length is 70
```

Когато обвивката срещне символ $, последван от име на променлива, тя поставя стойността на тази променлива на това конкретно място.

Променливи се използват най-често като аргументи в команди на обвивката. Обръщението към тях става с **$<име>.** Тъй като променливите нямат типове, то всички променливи на средата се интерпретират като символни низове. Навсякъде където се използва името на променлива, то се заменя със съответната му символна стойност.

5. *Използване на кавички в обвивката*

Някои от символите имат специално значение за обвивката. Такива са: `/, -, =,*, ?, [, ], >, <, |, \, $, ` и др. Съществуват 4 различни начина за премахване на специалното значение на символите:

- двойни кавички **„" ;**
- апостроф **'** ;
- обратно наклонена черта **** ;
- обратен апостроф **``** ;

5.1 Обратни апострофи

Когато дадена команда се включи в обратни апострофи, тя се изпълнява, и резултата от изпълнението й се поставя на мястото, където е била изписана командата.

5.2. Двойни кавички

Всеки символ, който притежава някакво специално значение за обвивката (с изключение на символите `$, `, \, „`), поставени в двойни кавички губи своето специално значение.

Забележка1: Ако в символния низ, заграден в двойни кавички има име на променлива, предшествана от $, тя се заменя със стойността на променливата.

Забележка2: Ако в символния низ, заграден в двойни кавички има команда, заградена с обратен апостроф, тя се заменя с резултата от изпълнението си.

5.3 . Апостроф

Апострофите са по-силни по действие от двойните кавички и премахват специалното значение на абсолютно всички специални символи заключени между тях.

5.3. Обратно наклонена черта

Тя премахва специалното значение само на символа, пред който е употребена.

6. *Параметри на командните файлове*

Използват се за подаване на конкретни входни данни при изпълнението на командния файл.

Ролята на формални параметри в LINUX играят **$1 $2...$9.** Те се използват в команден файл като входни параметри, като номера съответства на номера на подадения параметър при стартиране на файла. Значението на $0 е името на командния файл.

Когато при стартиране на команден файл се зададат повече от един параметър, те се разделят с интервал. За обработване на повече от 9 подадени параметъра от командния ред се използва командата shift, която измества параметрите с един наляво, като действието й е необратимо.

7. *Променливите* **$#, $* и $?**

Използват се в съдържанието на командни файлове за извличане на помощна информация, като:

$# дава броя на подадените параметри, при стартиране на командния файл

$* дава списък на подадените параметри

$? съдържа състоянието на системата след последната изпълнена Linux команда (съответства на кода на грешка след изпълнение на командата)

Пример:

Първа стъпка – създаване на команден файл

```
cat > opit
echo “Първи параметър $1”
echo “Втори параметър $2”
echo “Броя на параметрите е $#”
echo “Те са $*”
^D
```

Втора стъпка – дефинирането на файла като изпълним

```
chmod +x opit
```

Трета стъпка – стартиране на файла и извеждане на резултата

```
sh opit abc abv
```

Първи параметър abc

Втори параметър abv

Броя на параметрите е 2

Те са abc abv

8. *Извършване на изчисления*

Аритметичните операции с променливи на Shell се извършват с командата **expr**. Резултата от изпълнението на командата се извежда на стандартния изход.

Примери:

```
expr 5 + 10  → 15
```

`expr 6 * 2` → грешка, поради специалното значение на символа *. Правилното изпълнение на командата следва да премахне значението на *, като специален символ с помощта на „” или \ т.е. `expr 6 “*” 2` или `expr 6 \* 2`

```
br=1
expr $br + 1 → 2
```

Ако се налага използването на стойността получена с expr в променлива трябва да се използва следния запис:

```
br=`expr $br + 1`
```

9. *Сравняване на стойности*

Съществува една команда в Linux, която може да извършва сравнение на стойности – **test**. В зависимост от извършваната проверка операторите за сравнение могат да бъдат различни и са разделени в групи:

Синтаксисът на командата сравнение на цели числа е: **`test value –option value`**

Сравнение на цели числа	Функция	Пример
-gt	по-голямо от	`$n –gt 2`
-ge	по-голямо или равно	`$n –ge 3`
-lt	по-малко от	`$n –lt 5`
-le	по-малко или равно	`$n –le 4`
-eq	равно на	`$n –eq 4`
-ne	различно от	`$n –ne 3`

Синтаксисът на командата сравнение на низове е: **`test string comp string`**

Сравнение на низове	**Функция**	**Пример**
-z	проверка за празен низ	`–z $1`
-n	низът съществува (не е празен)	`–n $1`
=	равни низове	`$1 = ”abc”`
!=	различни низове	`$1 != ”xyz”`

Синтаксисът на командата при файлови проверки е**: `test –option value`**

Файлови проверки	**Функция**	**Пример**
-f	файлът съществува	`–f abc`
-s	файлът не е празен	`–s abc`
-r	файлът може да се чете	`–r abc`
-w	във файлът може да се записва	`–w abc`
-x	файлът може да се изпълнява	`–x abc`
-d	името на файлът е име на директория	`–d abc`

Синтаксисът на командата при логически операции е: **`test cond –option cond`**

Логически оператори	**Функция**	**Пример**
-a	логическо „И”	`–f $1–a –s $1`
-o	логическо „ИЛИ”	`–z $1 –o –d $1`
!	логическо отрицание	`! –d abc`

Обикновено в скриптовете се използва по-различен начин за изписване на сравненията. Вместо `test value -option value`, се записва **`[ value -option value ]`**

10. *Оператор if*

Три варианта на синтаксис на оператора:

I синтаксис	II синтаксис	III синтаксис
if [условие] **then списък_от_команди** **fi**	**if [условие]** **then списък_от_команди** **else списък_от_команди** **fi**	**if [условие]** **then списък_от_команди** **elif [условие]** **then списък_от_команди** **.....** **elif [условие]** **then списък_от_команди** **else списък_от_команди** **fi**

Пример: Да се напише команден файл .profile, който поздравява потребителя по име спрямо текущия час

```
name=`whoami`
time=`date | cut -c12-13`
if [ $time -gt 0 -a $time -lt 12 ]
then  echo "Good morning $name"
elif [ $time -ge 12 -a $time -lt 18 ]
  then  echo "Good afternoon $name"
  else  echo "Good evening $name"
fi
```

11. *Оператор case*

Операторът се използва в случаите, когато в зависимост от стойност на израз могат да бъдат извършвани няколко действия.

Синтаксиса е следния:

```
case стойност
in
шаблон 1) списък_от_команди;;
шаблон 2) списък_от_команди;;
.................................
шаблон N) списък_от_команди;;
       *) списък_от_команди;;
esac
```

Забележки:

1. Всяка отделна част от оператора case задължително завършва с **;;**

2. Служебния шаблон ***)** не е задължителен и се използва тогава, когато при несъвпадение на останалите шаблони се изпълняват командите описани след него.

3. В шаблона могат да участват и следните символи ***** и **|**

Пример: Да се напише команден файл, който в зависимост от подадения параметър изпълнява команда ls с различна опция

```
case $1
in
-l) ls -l;;
-s) ls -s;;
*) echo "Nekorektna opcia";;
esac
```

12. *Четене на данни от клавиатурата*

Използва се оператора **read**

read <име на променлива>

Забележка: При посочването на променливите, в които ще се въвежда стойност **не** трябва да се използва знака $. Той ще е необходим, когато се използва въведената стойност в друг израз.

13. *Оператор за цикъл for*

Операторът за цикъл **for** не е традиционния оператор за цикъл с повторения, които потребителя определя с начална и крайна стойност. В този случай цикълът се използва по-скоро за обхождане на елементите от списък, който обикновено се генерира от подадените към командния файл входящи параметри или съдържание на директория. Синтаксиса на оператора е:

```
for променлива in списък
do
   списък_от_команди
done
```

Пример: Да се напише команден файл, който копира всички изпълними файлове от текущата директория в директория ~/exec. Ако директорията exec не съществува, то да се създаде.

```
if [ ! -d ~/exec ]
then  mkdir ~/exec
fi
for i in *
do
  if [ -x $i -a ! -d $i ]
  then  cp $i ~/exec
  fi
done
```

Пример: Да се напише команден файл, който копира файловете, изброени като параметри на командния ред в директорията ~/newcopy.

```
if [ ! -d ~/newcopy. ]
then  mkdir ~/newcopy.
fi
if [ $# -eq 0 ]
then echo "No parameter"
else
  for i in $*
  do
  cp $i ~/newcopy
  done
 fi
```

14. *Оператор за цикъл while*

Операторът за цикъл **while** има по-скоро стандартно приложение в Shell. Синтаксисът е:

```
while [ условие ]
do
   списък_от_команди
done
```

Пример: Да се напише файл, който преброява въведени думи от вида a*p

```
echo “Въведи дума”
read a
br=0
while [ “$a” !=  “” ]
do
  case “$a”
  in
  a*p) br=`expr $br + 1`;;
  esac
echo “Въведи дума”
read a
 done
echo “Брой думи $br”
```