# KABUK PROGRAMLAMAYA GİRİŞ

**Linux ve UNIX sistemlerde yaygın kullanılan bazı kabuk(shell)lar;**

- sh (Shell , Bourne Shell): Ilk UNIX kabuğudur ve çoğu UNIX dağıtımı ile birlikte öntanımlı kabuk olarak dağıtılır.
- ksh (Korn Shell): sh uyumlu, birçok ek programlama özelliği de içeren bir kabuk.
- bash(Bourne Again Shell): Kullanım kolaylığı bakımından en çok rağbet gören bash, sh ve kshuyumluluğunu korurken, özellikle etkileşimli kullanıma yönelik (komut tamamlama, gibi) birçok yenilik de içerir.
- csh (C shell): Berkeley Üniversitesi'nde geliştirilen csh'in C diline benzer bir programlama yapısı vardır. Özellikle programcılar tarafından tercih edilir.
- tcsh: csh'ın biraz geliştirilmiş hali diyebiliriz. [3]

B iz uygulamalarımızda hatırlanacağı üzere Bash kabuğunu kullandık. Bash kabuğu güçlü programlama özellikleriyle karmaşık programların rahatça yazılmasına izin verir.

Mantıksal operatörler, döngüler , değişkenler ve modern programlama dillerinde bulunan pek çok özellik bash kabuğunda da vardır ve işleyiş tarzları da hemen hemen aynıdır.

Kabuk programlama deyince bu kabuğun programlanması değil bu kabuğa ait komutlarla, tek komutla yapılması imkansız işlerin bir dosyaya yazılmış komutlar yığınıyla yapılmaya çalışılmasıdır. Bu dosyalara ise genelde **script** adı verilir. Linux'teki script'leri, Windows'taki bat uzantılı dosyalar gibi düşünebiliriz. [2]

Genellikle, bir programı oluşturacak olan komutlar bir dosyaya yazılırlar ve ardından bu dosya çalıştırılır. Herhangi bir editör yardımıyla yazılan program, daha sonra kabuk altında çalıştırılır. Bir kabuk programı diğerler kabuk programlarını da çalıştırabilir. Bu düzende kabuk programlarını daha karmaşık komutların biraraya gelmiş ve yapısallaşmış haline benzetebiliriz.

Bash'in en büyük dezavantajı, derlenerek çalıştırılan dillere göre (C, C++ gibi) daha yavaş olması, sistem kaynaklarını biraz daha fazla tüketmesidir.

Kod yazmaya başlamadan bir ek daha yapalım. Yazı boyunca kodladığımız örnekleri sizlerde herhangi bir Linux altında yazıp çalıştırabilirsiniz. Editör olarak KDE altında Kedit, Gnome altında Gedit kullanmanızı tavsiye ederim. Eğer her işimi konsoldan yapmak istiyorum derseniz pico, nano ya da Vi/Vim gibi konsol tabanlı editörleri de kullanmayı deneyebilirsiniz. [3]

## *Basit Bir Kabuk Programı Örneği*

Kabuk programları, bir veya birden fazla Linux komutunu tutan dosyalardır. Basit bir örnek üzerinde anlatalım.

1. Dosyayı oluştur: Öncelikle “calistir” isimli bir dosya oluşturup içerisine aşağıdaki kodu yazalım:

```
#!/bin/bash
echo "Merhaba Dünya!"
```

2. Dosyaya çalıştırma iznini ver: Bir kabuk programı, çalıştırma bitini 1 yapmak suretiyle "çalıştırılabilir" hale getirilir.

```
[root@localhost Masaüstü]# chmod +x calistir
[root@localhost Masaüstü]# ls -l calistir
-rwxr-xr-x 1 root root 23 Şub 15 20:04 calistir
```

3. Dosyayı çalıştır: Bundan sonra programın ismi yazılıp enter tuşuna basıldığı zaman bir program Linux komutuymuş gibi çalışacaktır.

```
[root@localhost Masaüstü]# ./calistir
Merhaba Dünya!
```

## *Kabuk Programlamanın Kuralları*

### 1. Açıklama Satırı Kullanımı:

Kabuk programları yazarken dosyanın işlevini ve her satırdaki komutun veya komut kümesinin ne amaçla kullanıldığını gösteren açıklama satırları kullanmak işe yarar.

```
#calistir
# aşağıdaki ifade ekrana yazı basmak için kullanılır.
echo "Merhaba Dünya!"
```

Yorum satırı, komutun sonuna da eklenebilir.

```
#calistir
echo "Merhaba Dünya!" # aşağıdaki ifade ekrana yazı basmak için kullanılır.
```

### 2. Değişken Kullanımı:

Bir değişkene değer atandığı anda sistem tarafından tanınır. Değişkenler alfabetik veya nümerik karakterlerden oluşabilirler fakat bir değişken sayısal bir değer ile başlayamaz.

Bunların dışında değişken isminin içinde "_" karakteri de bulunabilir.

Bir değişkene değer ataması "=" işareti yardımıyla yapılır. İçeriği olan bir değişkene başına "$" işareti konularak ulaşılır.

```
[root@localhost Masaüstü]# a=/root
[root@localhost Masaüstü]# echo $a
/root
[root@localhost Masaüstü]# ls $a
anaconda-ks.cfg       ns-allinone-2.30
Belgeler              ns-allinone-2.30.tar.gz
dead.letter           openssh-3.5p1
Geçici                openssh-3.5p1-11.dag.src.rpm
Genel                 openssh-3.5p1.tar.gz
httpd-2.2.14          Resimler
httpd-2.2.14.tar.bz2  sendmail-8.12.8-7.dag.src.rpm
İndirilenler          sendmail-8.14.3
install.log           sendmail.8.14.3.tar.gz
install.log.syslog    Videolar
listele               vsftpd-1.2.1
Masaüstü              vsftpd-1.2.1-3E.15.i386.rpm
Müzik                 vsftpd-1.2.1.tar.gz
```

Değişkenler script dosyalarında da kullanılabilir.

```
#!/bin/bash         #calistir
mesaj="merhaba dünya!!"
echo $mesaj
```

Yukarıdaki dosyanın çalıştırılması ve çıktısı aşağıdaki gibidir:

```
[root@localhost Masaüstü]# ./calistir
merhaba dünya!!
```

## 3. Giriş/Çıkış İşlemleri

Bir kabuk programı çalışırken kullanıcıdan klavye yardımıyla bilgi girmesi sağlanabilir.

```
#!/bin/bash         #calistir
echo "bir sayı giriniz:"
read sayi
echo "girdiğiniz sayi: $sayi"
```

Yukarıdaki dosyanın çalıştırılması ve çıktısı aşağıdaki gibidir:

```
[root@localhost Masaüstü]# ./calistir
bir sayı giriniz:
23
girdiğiniz sayi: 23
```

## 4. Parametre Kullanımı

Kabuk programı çalıştırılırken parametre kullanımı da mümkündür.

Örnek olarak 3 parametreli bir script yazalım. Bu script ilk parametredeki isimde bir kullanıcı oluşturup 2. parametredeki değeri bu kullanıcının ID değeri olarak atayan ve bu kullanıcıyı 3. parametrede belirtilen isimde bir grup oluşturup içine dâhil eden bir script olsun.

Parametreler sıralarıyla aynı isimdeki değişkenler olarak tutulur. Mesela; birinci parametre için $1, ikinci parametre için $2… gibi.

Programın ismine ise program içinde $0 değişkeni ile ulaşılabilir.

```
#!/bin/bash         #calistir
useradd $1
usermod –u $2 $1
groupadd $3
usermod –g $3 $1
```

## 5. Aritmetik İşlemler

Bash kabuğunda matematiksel işlemlere büyük sınırlamalar getirilmiştir. Aritmetik işlemler için eval, awk, let, bc operatörleri gibi farklı alternatifler kullanılabilir. Aritmetik değişken tanımlamanın yollarından biri `typeset` komutu kullanmaktır.

```
#!/bin/bash         #calistir
#calistir
typeset -i sonuc #sonuç değişkeni integer olacak
a=12
 ; b=23
sonuc=$a*$b
echo $sonuc
```

Yukarıdaki dosyanın çalıştırılması ve çıktısı aşağıdaki gibidir:

```
[root@localhost Masaüstü]# ./calistir
276
```

Normal olarak bash, kesirli ve noktalı işlemleri yapamaz. Bunun için **bc** kullanabilirsiniz. **Bc**, çok yüksek duyarlılığa sahip bir hesap makinasıdır.

```
[root@localhost Masaüstü]# a=24.565
[root@localhost Masaüstü]# b=45.676
[root@localhost Masaüstü]# echo "$a*$b" | bc
1122.030
```

## 6. if-else Kalıbı ve Kontrol İşlemleri

Hemen her programlama dilinde olan `if` kalıbı bir Linux komutunun çalışmasını kontrol (`test`) eder. Her `if`, bir `fi` komutuyla bitmelidir. Aşağıda if-then-else komutunun örnek sözdizimi görülüyor.

```
if linux komutu
    then
        komut1
        komut2
        ...
    else
        komut1
        komut2
        ...
fi
```

`if` komutu genellikle kendine `test` komutu ile birlikte kullanım bulur. Mantıksal işlemler ve karşılaştırmalar test komutu ile yapılır. Test komutu ile birlikte karşılaştırma işlemi için opsiyonlar kullanılır. Sözkonusu opsiyonlar şu şekildedir:

| Aritmetik karşılaştırma | |
|---|---|
| -gt | büyük |
| -lt | küçük |
| -ge | büyük eşit |
| -le | küçük eşit |
| -eq | eşit |
| -ne | eşit değil |
| Dizisel karşılaştırma | |
| -z | boş dizi |
| -n | tanımlı dizi |
| = | eşit diziler |
| != | farklı diziler |
| Dosya karşılaştırması | |
| -f | dosya var |
| -s | dosya boş değil |
| -r | dosya okunabilir |
| -w | dosyaya yazılabilir |
| -x | çalıştırılabilir dosya |
| -h | sembolik bağlantı |
| -c | karakter aygıt |
| -b | blok aygıt |

| Mantıksal karşılaştırma | |
|---|---|
| -a | VE |
| -o | VEYA |
| ! | DEĞİL |

Bu opsiyonların nasıl kullanıldığını bazı örneklerle anlatalım. Burada üzerinde durulması gereken nokta mantıksal işlemlerdekinin aksine 0'ın doğru (true), 1'in yanlış (false) olmasıdır.yani komut başarılı olarak işletilmişse çıkış değeri 0, değilse 1'dir. Son çalıştırılan tüm Linux komutlarının çıkış değeri $? değişkeninde tutulur.

```
[root@localhost Masaüstü]# ls
a~          cikti       e~                              ns2 kurulumu~
b~          cikti~      Kabuk Programlamaya Giriş.doc   ornekler
c~          d~          Kitap1.xls                      örnekler
calistir    d1          kullanici hesap~                yeni dosya~
calistir~   dosya.txt~   ns2 kurulumu
[root@localhost Masaüstü]# a=10 ; b=15
[root@localhost Masaüstü]# test $a -eq $b                  (-eq: eşitlik)
[root@localhost Masaüstü]# echo $?       (karşılaştırma sonucunu göster)
1                                          (eşit değil anlamında: 1)
[root@localhost Masaüstü]# a=10 ; b=10

[root@localhost Masaüstü]# test $a -eq $b
[root@localhost Masaüstü]# echo $?
0                                                        (eşittir)
[root@localhost Masaüstü]# b=5
[root@localhost Masaüstü]# test $a -lt $b               (-lt: küçüklük)
[root@localhost Masaüstü]# echo $?
1                                                        (küçük değil)
[root@localhost Masaüstü]# b=15
[root@localhost Masaüstü]# test $a -lt $b
[root@localhost Masaüstü]# echo $?
0                                                          (küçüktür)
[root@localhost Masaüstü]# test $a -gt $b              (-gt: büyüklük )
[root@localhost Masaüstü]# echo $?
1                                                        (büyük değil)
[root@localhost Masaüstü]# test -f calistir         (-f: dosya var mı?)
[root@localhost Masaüstü]# echo $?
0                                                          (dosya var)
[root@localhost Masaüstü]# test -f olmayan_dosya
[root@localhost Masaüstü]# echo $?
1                                                          (dosya yok)
[root@localhost Masaüstü]# a=10 ; b=15
[root@localhost Masaüstü]# test $a -eq $b -a -f calistir
[root@localhost Masaüstü]# echo $?
1
[root@localhost Masaüstü]# test $a -lt $b -a -f calistir
[root@localhost Masaüstü]# echo $?
0
```

`test` komutu yerine köşeli parantezler [ ] de kullanılabilir. Dikkat edilmesi gereken bir nokta, köşeli parantez kullanırken araya boşlukların eklenmesidir. Parantezler başlı başına bir komut olarak görüldüklerinden sağında ve solunda en az bir boşluk bırakılmalıdır.

```
[root@localhost Masaüstü]# a=10 ; b=15
```

```
[root@localhost Masaüstü]# [ $a -lt $b ]
[root@localhost Masaüstü]# echo $?
0
```

test komutu ile if operatörü birlikte aşağıdaki örnekteki gibi kullanılabilir:

```
#!/bin/bash         #calistir
echo "0 ile 20 arasinda bir sayi secin"
read sec
if [ $sec -lt 10 ]
   then
        echo "Seçtiğiniz sayi : $sec"
   else
        echo "Sectiğiniz sayi belirtilen aralıktan değil."
fi
```

## 7. case Kalıbı

Birkaç alternatif arasından seçim yapmak için kullanılan bir komut olan `case`, bir eşleştirme gördüğü anda belirli bir komut kümesini işleme sokar. case yapısı `case` komutu ile başlar, eşleştirilecek olan anahtar sözcük yazılır ve seçenekler alt alta, her seçeneğe ait olan komutlarla birlikte belirtilir. Tüm yapı `esac` komutu ile son bulur.

```
case anahtar-sozcuk in
        secenek1)
                komutlar
                ;;
        secenek2)
                komutlar
                ;;
        *)
                komutlar
                ;;
esac
```

Seçenekler arasında özel karakterler *(*, [, ], ? gibi)* kullanılabilir. Hiçbir eşleme yapılmadığı zaman *) seçeneği değerlendirilecek ve buna bağlı olan komutlar işletilecektir. * kullanımı isteğe bağlıdır.

**Örnek:** Girilen bir sayıya bağlı olarak ekranı temizleme, sistemdeki kullanıcıları gösterme ve dizindeki dosyaları gösterme işlemlerini gerçekleştiren bir program yazınız.

```
#!/bin/bash

clear
echo    "1. ekrani temizle"
echo    "2. sistemdekileri goruntule"
echo    "3. dizindeki dosyalari goster"

echo -n "Secenegi giriniz : "
read secenek

case $secenek in
        1)
```

```
          clear
          ;;
        2)
          w
          ;;
        3)
          ls -al
          ;;
        *)
          echo Hatali secenek
esac
```

## 8. Döngüler

Diğer hemen tüm programlama dillerinin en büyük gücü olan döngü işlemlerine kabuk altında da izin veriliyor. Burada programcı tarafından en çok kullanılan 2 döngü tipi anlatılacaktır: `while` ve `for`. `while` komutu her döngüde bir denetleme mekanizmasını harekete geçirirken `for` döngüsü bir listenin elemanlarını sırayla seçer.

### a. while-do Döngüsü

Döngü bloğu `while` anahtar kelimesiyle başlar, ardından gelen koşul sağlandığı sürece döngü işletilir. Önce koşulun sağlanıp sağlanmadığına bakılır. Döngüden çıkabilmek için mutlaka döngü içindeki koşul ifadesinin değerini yanlış yapacak bir durum oluşmalıdır, aksi halde sonsuz döngü oluşur.

```
while kosul ifadesi
   do
        komutlar
   done
```

`if` komutuyla birlikte kullanılan `test` komutu, `while` döngüsünde koşul ifadesi olarak da yer alabilir. Aşağıda 1'den 100'e kadar sayan ve ekrana basan bir döngü görülüyor.

```
#!/bin/bash
deger=0
while [ $deger -lt 100 ]
   do
        deger=$((deger+1))
        echo $deger
   done
```

Yukarıda kullanılan (( ve )) karakterleri arasına matematiksel bir işlem getirilebilir. Bu özellik bash kabuğuna özgüdür.

### b. for-do döngüsü

Bir liste dahilindeki tüm değerlere sırayla erişimi sağlar. for komutundan sonra yeralan liste sırayla kullanılır ve herbirisi için döngü çalıştırılır. Listenin sonuna gelindiğinde ise döngüden çıkılır.

```
for degisken1 in deger1 deger2 ... degerX
   do
        komutlar
   done
```

Aşağıdaki örnek bu döngüyü kullanarak ekrana bir dizi kelime yazıyor. Döngü boyunca akasya, elma ve visne kelimeleri "agac" değişkenine kopyalanıyor ve her döngüde bu değişkenin içerdiği bilgiler ekrana yazılıyor.

```
for agac in akasya elma visne
   do
        echo $agac
   done
```

for-do döngüsü, dosya isimleri üzerinde yapılan işlemlerde de büyük kolaylıklar sağlar. Bunun için özel karakterlerden yararlanmak da olasıdır. Örnek olarak * karakteri o anki çalışma dizini içindeki tüm dosyaları seçer.

```
for a in * ; do
        file $a
        done
```

## 9. Diziler

Bir kabuk programı yazılırken benzer özellikteki değişkenler diziler olarak tutulabilir. Diziler aşağıdaki gibi ifade edilir;

dizi =(değişken_0 değişken_1 …)

Örneğin;

Aylar=(Ocak Subat Mart nisan Mayis Haziran Temmuz Agustos Eylul Ekim Kasim Aralik )

Dizilerin indisi 0'dan başlar. ${# dizi[*]} ifadesiyle dizinin eleman sayısına ulaşılabilir.

```
#!/bin/bash

dizi=(Ocak Subat Mart nisan Mayis Haziran Temmuz Agustos Eylul Ekim Kasim
Aralik )

echo "Bir değer giriniz:"

read deger

echo "cevap: ${dizi[$deger-1]}"
```

## 10. Fonksiyonlar

Fonksiyonlar kullanılmadan önce tanımlanmalıdır. Bash programlamada iki çeşit fonksiyon tanımı yapılabilir;

```
function fonksiyon_adi {
    komutlar
}
```

ya da;

```
fonksiyon_adi () {
    komutlar
}
```

Örnek;

```
#!/bin/bash

function merhaba() {

echo "merhaba dünya"

}

merhaba
```

Fonksiyon tanımlarken dikkat edeceğimiz bir nokta da şudur; normal dillerde fonksiyon içinde tanımlanan değişken sadece o fonksiyon içerisinde geçerli olurdu. Halbuki bash'de fonksiyon içinde tanımlanan değişken tüm program boyunca aktif olur. Bunu engellemek yani fonksiyon içinde tanımlanan değişkenin geçerliliğini fonksiyon ile sınırlamak istersek local değişken tanımı kullanırız. [4] Örneğin;

Local_degisken="bu degisken yereldir"

## *Referanslar*

[1] http://www.belgeler.org/lis/archive-tlkg-lis-6.6.html

[2] http://www.programlama.com/sys/c2html/view.php?DocID=800

[3] http://www.e-ogretmen.org/site/2009041048/linux/linux-kabuk-programlama-sh-shell-bash.html

[4] http://www.enderunix.org/docs/linuxileprogramlama.html

## *Diğer faydalı Linkler:*

http://cayfer.bilkent.edu.tr/~cayfer/kku/141-189.pdf

http://www.ceturk.com/muhendislik/kabuk-programlama-bash-i.html

http://www.seyhan.biz/index.php/tag/linux/

http://wiki.linux-sevenler.org/index.php/Kabuk_Programlama

http://yucelkilic.com/?p=228

http://www.unix.com/shell-programming-scripting/

http://www.tldp.org/LDP/abs/html/index.html