

bash Kabuđu ve Kabuk Programları

07

- Komut Satırının Yorumlanması ve Parametreler
- Kabuk Deđişkenleri
(ya da Ortam Deđişkenleri)
- Programları Arka Planda Çalıřtırmak
- Ön Planda Çalıřan Programları
Arka Plana Atmak
- Kabuk Programlama
 - İlk Kabuk Programı Örneđi
 - İkinci Kabuk Programı Örneđi

LINUX işletim sistemi, kullanıcıların verdikleri komutları çözümlenmek ve bu komutları yerine getirecek programları başlatmak için **kabuk** (*shell*) programlarını kullanır. Bir başka deyişle, kabuk programları, kullanıcılarla bilgisayar arasındaki komut arabirimidir. Aslında, bu tip komut arabirimleri; yani “**komut yorumlayıcıları**” (*command interpreter*), tüm işletim sistemlerinde kullanılmaktadır; örneđin MS-DOS işletim sisteminde bu görevi COMMAND.COM üstlenmiş durumdadır.

LINUX işletim sisteminde, kullanıcıların birden fazla kabuk programı arasından seçim yapma ve beğendikleri komut yorumlayıcısını kullanma hakları vardır. Hatta aynı anda birden fazla kabuk programı bile kullanabilirler.

Bu kitapta LINUX işletim sisteminin en popüler kabuk programı olan **bash** kabuk programını anlatacađız. Ayrıntılarda önemli farklar olsa bile **bash** bilen birisi **csh**, **tcsh**, **sh**, **ksh** gibi diđer kabuk programlarını çok çabuk kavrayacaktır.

LINUX kullanımında deneyim kazanıp sistemin hakkını vermeye başladığınızda; özellikle sistem yöneticisi olma yolunda ilerledikçe, kabukların öne-

mi hızla ortaya çıkacaktır. Gerek sisteminize ileri düzey komutlar verirken, gerekse bir takım işleri otomatiğe bağlamaya başladığınızda kabukların yararlı özelliklerini kullanmaya ve daha önemlisi kabuk komutlarını kullanarak kendi komut dizilerinizi (*shell script*) yazmaya başlayacaksınız.

Bu aşamada, bu bölümün özellikle “**kabuk programlama**” ile ilgili kısımları size karışık gelebilir. Hele programcılık deneyiminiz yoksa bu bölümleri hiç üzülmeyen atlayabilirsiniz. LINUX kullanmak için ille de programcı olmak gerekmez.

Komut Satırının Yorumlanması ve Parametreler

bash kabuk programı komut almaya hazır olduğunu ekranda

```
[cayfer@pusula cayfer] $
```

gibi bir karakter dizisiyle (**hazır işareti**: *prompt*) belirtir.

```
cp eski-dosya yeni-dosya
```

gibi bir komut verdiğinizde, kabuk programı, “**cp**” harflerini çalıştırmak istediğiniz programın adı olarak; “**eski-dosya**” ve “**yeni-dosya**” sözcüklerini ise bu **cp** programının iki parametresi olarak kabul edecektir.

Kabuğun yapacağı bir sonraki iş, çalıştırmak istediğiniz bu **cp** programının saklandığı disk dosyasını bulmak olacaktır. Bu arama işinin temelinde, sizin için tanımlanmış olan **PATH** kabuk ortam değişkeninin o andaki değeri yatmaktadır. Bu değişkenlerin değerleri tipik olarak

```
/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:/home/cayfer/bin
```

benzeri bir karakter dizisi olacaktır.

Kendi **PATH** değişkeninizi görmek isterseniz “**echo \$PATH**” komutunu kullanabilirsiniz.

bash programı, **PATH** değişkeninin değeri içinde, “:” karakteriyle birbirlerinden ayrılmış olan dizinleri sırayla tarayıp bu dizinlerin içinde, sizin çalış-

tırma yetkiniz olan “**cp**” isimli bir dosya arayacaktır. Arama, dizin isimlerinin **PATH** değişkeninde veriliş sırasına göre yapılacaktır.

Örneğimize göre, **bash** programı, **cp** isimli dosyayı önce **/usr/local/bin** dizininde, orada bulamazsa **/bin** dizininde; orada da bulamazsa **/usr/bin**; olmazsa **/usr/X11R6/bin** dizininde, gene bulamazsa **/home/cayfer/bin** dizininde arayacaktır. Kabuk programınız söz konusu dosyayı bu dizinlerden hiçbirinde bulamazsa,

bash: cp: command not found

diye, komutu tanıyamadığına ilişkin bir hata mesajı vererek yeniden komut bekleme durumuna dönecektir.

Eğer **cp** program dosyası bu dizinlerden birinde bulunursa, bu dosyanın erişim yetkileri kontrol edilecektir. **cayfer**'in bu programı çalıştırmaya yetkisi varsa **cp** programı kabuk tarafından belleğe yüklenecek ve çalıştırılacaktır. Varsa, komut satırında verilen parametreler çözümlenip (çözümlemeden ne kasdettiğimizi biraz sonra açıklayacağız) **cp** programına aktarılacaktır.

Yüklenen program çalışmaya başladığında kontrol artık **cp** programına geçmiştir. Parametrelerin doğru sırada ve sayıda verilip verilmediğini her program kendisi kontrol eder ve gerekirse uygun hata veya uyarı mesajları üreterek, kullanıcıyı uyarır.

Bir komut verildiğinde o komuta ilişkin hangi program dosyasının belleğe yükleneceğini merak ettiğinizde

which komut

komutunu kullanabilirsiniz. “Böyle bir şeyi neden merak edeyim ki?” diyorsanız bilgisayarınızda aynı isme sahip iki program dosyası olabileceğine dikkatinizi çekeriz.

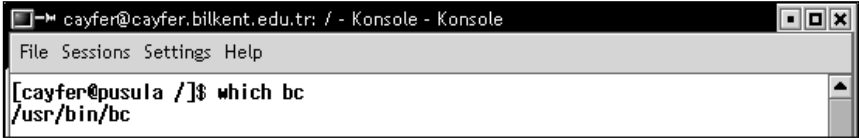
Örneğin kendiniz C dili ile “**bc**” diye bir program yazabilir ve kişisel dizinize yerleştirebilirsiniz. Programınızı güzelce hatasız derledikten sonra çalıştırmak için “**bc**” komutunu verdiğinizde garip şeyler olduğunu gözleyebilirsiniz. İlk aklınıza gelen “*acaba doğru programı mı çalıştırıyorum?*” olmalıdır.

Kim Korkar LINUX'tan?

Bu soruya yanıt bulmak için:

which bc

komutunu verdiğinizde göreceksiniz ki sizin yazdığınız **bc** programı değil, **/usr/bin/bc** çalışıyor. Çünkü **bash** kabuğu çalıştıracakı komuta ilişkin program dosyasını **PATH** değişkenindeki dizinlerde ararken **/usr/bin** dizinindeki **bc** dosyası sizin kişisel dizininizdeki **bc** dosyasından daha önce bulunuyor.



```
caufer@caufer.bilkent.edu.tr: / - Konsole - Konsole
File Sessions Settings Help
[caufer@pusula /]$ which bc
/usr/bin/bc
```

Böyle bir durumda ya kendi programınızı yerini açıkça belirterek; yani komut olarak

/home/caufer/bc veya
./bc

girerek (“**./bc**”, bu dizindeki **bc**” anlamında) çalıştırmalısınız; ya da **PATH** değişkeninizi kendi kişisel dizininiz daha önde olacak şekilde değiştirmelisiniz.

Kopyalama komutunu, çalışma dizinindeki tüm dosyaları

/disk2/home2/ayfer dizinine kopyalayacak şekilde

cp * /disk2/home2/ayfer

verdiğiniz varsayalım ve **bash** kabuğunun neler yapacağına bir göz atalım. Bu komutu gören **bash**, komut adı olan **cp** sözcüğünü bulduktan sonra, bu komutun parametrelerini bulup çıkarmaya çalışacaktır. Komut satırını tararken ***** karakterine rastlayınca, “tüm dosyalar” anlamına gelen bu işaret yerine, çalışma dizininde yer alan tüm dosya ve dizinlerin isimlerini yan yana gelecek şekilde yerleştirecektir.

Yani komut satırı,

cp abc dosya1 dosya2 xyz x123 /disk2/home2/ayfer

satırına benzer bir şekle dönüştürülecektir. (Çalışma dizininde sadece **abc**, **dosya1**, **dosya2**, **xyz** ve **x123** dosyalarının olduğunu varsayarak.) Bu dönü-

şümü ekranda gözleyemezsiniz; ancak bu tip dönüşümlerin olduğunu bilmeniz ve komutları verirken bu dönüşümleri dikkate almanız çok önemlidir.

Bazı durumlarda kabuk programlarının komutlarındaki parametreleri yorumlayıp açmaya çalışmasını istemezsiniz. Böyle bir gereksinim duyarsanız (ki **find** komutunu kullanırken duyacaksınız); kabuk programının kurcalamasını istemediğiniz parametreleri tırnak içinde yazmalısınız. Eğer kabuk programının irdelemeden komuta aktarmasını istediğiniz özel karakter tek bir karakterden oluşuyorsa, o karakteri tırnak içine almak yerine, önüne bir “\” (back slash) yerleştirebilirsiniz.



Bu arada, kabuk tarafından çalıştırılan programların sıfıncı parametrelerinin de bulunduğunu söylemeden geçemeyeceğiz. Bir program çalıştırıldığında, sıfıncı parametresi, programın kendi adıdır. Böylece, her program, hangi isimle kullanıldığını bilebilmektedir. Bu özelliğe tipik örnek **gzip** ve **gunzip** komutlarıdır. Bu iki komuta ait program dosyaları aslında tıpatıp aynı dosyadır. Aslında **gzip** isimli dosya gerçekten bu isimle diskte yer alırken, **gunzip** sadece bu dosyaya bir bağlantı da (*link*) olabilir. Yani **gzip** komutunu da verseniz, **gunzip** komutunu da verseniz aynı program dosyası belleğe yüklenip çalıştırılır. Bir dosyayı sıkıştırması gerektiğini mi yoksa sıkıştırılmış bir dosyayı açması mı gerektiğini anlamak için program sıfıncı parametreye, yani hangi isimle çalıştırıldığına bakarak karar verir.

Kabuk Değişkenleri (ya da Ortam Değişkenleri)

bash kabuk programları içinde çeşitli değişkenler tanımlamanız mümkündür. Hatta bazı standart değişkenler zaten öntanımlıdır. Öntanımlı kabuk değişkenleri arasında en önemlileri şunlardır:

Bazı Önemli bash Kabuk Değişkenleri	
PATH	Bir komut verildiğinde, komut programını oluşturan dosyanın aranacağı dizinler listesini içeren değişkendir.
HOME	Kullanıcının kişisel dizininin adını içeren değişkendir. Sisteme bağlandığınızda kabuk programı tarafından otomatik olarak yaratılır.

TERM	Kullandığınız terminalin tipini belirleyen deęiş-kendir. X altında çalışırken genellikle “ xterm ” deęerini içerir. Windows bir makinadan telnet ile bağlanıldığında ise “ vt100 ” deęeri verilmeli-dir; daha doğrusu önerilir.
DISPLAY	X altında çalışırken grafik ekran görüntülerinin gönderileceęi X sunucusunun adresini ve ekran numarasını içermelidir. Kendi bilgisayarınızda çalışırken “ :0.0 ” deęerini içerir. Bir başka bil-gisayarda çalıştıracağınız X uygulamasının gö-rüntüsünü kendi ekranınıza almak istediğinizde o dięer bilgisayara bağlantı kurmak için kullandığınız terminal ekranında çalışmakta olan ka-buk programının DISPLAY deęişkeninde “ 192.168.10.32:0 ” gibi bir deęer olmalıdır. Buradaki 192.168.10.32 yerine önünde oturu-duğunuz grafik ekranın takılı olduęu bilgisaya-rın IP numarası veya adı gelmelidir.
HISTSIZE	Kabuk programınızın, en son verilen kaç ko-mutu saklayacağını gösteren deęeri içerir. Ka-buk programları, her kullanıcının kullandığı son HISTSIZE komutu kişisel dizinlerdeki .bash_history isimli dosyalarda saklar.
SHELL	O anda çalışmakta olan kabuk programının yeri ve adıdır. (/bin/bash gibi.)
LD_LIBRARY_PATH	Programların kullandıkları, paylaşılan kütüpha-nelerin aranacağı dizinleri sırasıyla gösteren bir karakter dizisi içerir. Kullanım mantığı PATH de-ęişkenine çok benzer; yalnızca aranan program deęil, kütüphanedir.

Herhangi bir anda, kabuğunuzda tanımlı olan deęişkenleri ve/veya deęerle-rini merak ederseniz,

env

komutunu kullanabilirsiniz.

```

cayfer@cayfer.bilkent.edu.tr: / - Konsole - Konsole
File Sessions Settings Help
[cayfer@pusula ~]$ env
PWD=/
ORACLE_SID=LOJMAN
XAUTHORITY=/home/cayfer/.Xauthority
LC_MESSAGES=en_US
HOSTNAME=cayfer.bilkent.edu.tr
LD_LIBRARY_PATH=/usr/local/lib
NLSPATH=/usr/share/locale/%l/%N
LESSKEY=/etc/.less
LANGUAGE=en_US:en
LESSOPEN=|/usr/bin/lesspipe.sh %s
HISTIGNORE=[ ]*:&:bg:fg
LESS=-MH
BROWSER=kfmclient openProfile webbrowsing
USER=cayfer
HISTCONTROL=ignoredups
MACHTYPE=i586-mandrake-linux-gnu
KDE_MULTHEAD=false
HELP_BROWSER=kfmclient openProfile webbrowsing
MAIL=/var/spool/mail/cayfer
INPUTRC=/etc/inputrc
OLDPWD=/
BASH_ENV=/home/cayfer/.bashrc

```

Kabuk değişkenleri, standart isimli birtakım değişkenlerle sınırlı değildir. Kullandığınız uygulama programları, çalışma ortamını tanımlamak için özel değişkenlerin tanımlanmasını ve özel değerler verilmesini gerektirebilir. Örneğin Oracle istemcileri **ORACLE_HOME** ve **ORACLE_SID** isimli iki kabuk değişkeni tanımlanmış olmasını ister.

Bir nedenle bir kabuk değişkeni tanımlamak ya da tanımlı bir değişkenin adını değiştirmek gerektiğinde; örneğin **DISPLAY** değişkenine “**192.168.10.33:0**” değerini vermeniz gerektiğinde

```
export DISPLAY=192.168.10.33:0
```

komutunu kullanmalısınız.

Komut satırından istediğiniz gibi kabuk değişkeni tanımlayabilir ya da değerlerini değiştirebilirsiniz; ancak bu değişiklik kalıcı olmaz. Kabuk programı kapatıldığında ya da öldürüldüğünde tanımladığınız değişken de kaybolur gider. Ayrıca bu şekilde yapılan bir tanım, yalnızca tanımın yapıldığı kabuk için geçerlidir. Yani, bir anda aynı bilgisayara bağlı olan üç telnet pencereniz varsa, bunlardan birinde vereceğiniz **export** komutu yalnızca o terminal penceresinde çalışan kabuk için geçerli olacaktır. Tüm pencereleriniz de geçerli ve kalıcı bir kabuk değişkeni tanımlamak istiyorsanız, bu işi ya-

pan **export** komutunu kişisel dizininizdeki **.bashrc** dosyasına eklemelisiniz. Bu tanımın tüm kullanıcılar için geçerli olmasını istiyorsanız, aynı eklemeyi **/etc/bashrc** dosyasına da yapmalısınız (nokta yok).

Programları Arka Planda Çalıştırmak

Diyelim ki, çok büyük bir disk dosyasındaki (söz gelimi 200 Mbyte) müşteri kayıtlarını alfabetik sırada dizmek istiyorsunuz. Bu işin, kullandığınız bilgisayar sisteminde yarım saat süreceğini varsayalım. Eğer tek iş düzeninde çalışan bir işletim sistemi kullanıyor olsaydınız, **sort** sıralama komutunu verdikten sonra yemeğe çıkabilir veya köpeğinizi dolaştırmaya götürebilirdiniz; çünkü sıralama bitinceye kadar bilgisayarınızdan bir başka amaçla yararlanmanız söz konusu olamazdı. Oysa LINUX işletim sisteminde, sıralamayı arka planda bir iş olarak başlattıktan sonra, ön planda başka işler yapmanız mümkündür. Bunu yapabilmek için tek yapmanız gereken, arka planda yapılmasını istediğiniz işi başlatan komutun sonuna bir “&” işareti eklemektir.

```
sort musteri_dosyasi > sıralı_dosya &
```

Diyeceksiniz ki “arka planda çalıştırmaya ne gerek var, yeni bir telnet penceresi açar, uzun işi orada başlatırım ve normal çalışmamda kullandığım pencereye geri dönerim!” Doğru! Yapabilirsiniz. Mis gibi de çalışır. Geri planda iş çalıştırmak için komut sonuna “&” işareti koyma fikri, yeni bir pencere açmaya üşenenler içindir...

Elbette ki her iş bu şekilde arka planda çalıştırılmaya uygun değildir. Örneğin, bir editör programı gibi, kullanıcının sürekli olarak klavyeden bilgi girmesini gerektiren programlar arka planda çalıştırılsa bile, sürekli ilgi istedikleri için bu tip bir çalışma anlamlı olmaz. Oysa yukarıdaki sıralama örneğimizde, sıralama süresince kullanıcıdan herhangi bir bilgi istenmeyecektir. Sıralama programı arka planda sessizce çalışıp işini bitirecektir.

Bazı programlar kullanıcıdan bir bilgi istememekle birlikte, sürekli olarak ekrana yaptıkları işin gelişmesini açıklayan bilgiler dökerler. Bu tip bir programı arka planda çalışmak üzere başlattığınızda, sürekli olarak ekrana gelen bilgiler yüzünden ön planda başka bir iş yapmanıza pek olanak kalmaz. Örneğin, genellikle bir dizindeki tüm dosya ve alt dizinleri tek bir dosya içinde toplamak (paketlemek) için kullanılan **tar** komutunu,


```
tar -cvf hepsi.tar /home/ayfer &
```

şeklinde verirseniz (bu komutla ilgili detaylı bilgiyi daha ileride vereceğiz; şimdilik komutun ne yaptığı ve parametrelerinin ne olduğu üzerinde durmayınız), program arka planda işini yapacaktır, ama bir yandan da paketlediği dosyaların isimlerini ekrana listeleyecektir. Bu şekilde her saniye yeni bir satır listelenen bir ekranda başka bir iş yapmak pek kolay olmayacaktır.

Ancak aynı komutu,

```
tar -cvf hepsi.tar /home/ayfer > /tmp/tarmesajlari &
```

şeklinde verirseniz, ekrana gelmesi gereken tüm mesajlar, /tmp dizinindeki **tarmesajlari** isimli bir dosyaya yönlendirilmiş olacaktır. İş bittikten sonra **tarmesajlari** dosyasına bakarak paketleme işinin başarıyla bitip bitmediğini ve kopyalanan dosyaların listesini görebilirsiniz.

Geri planda başlattığınız işlerin hata mesajı üretmeleri durumunda mesajlar **STDERR**'e gidecektir. Çalıştığınız ekrana zırt pırt hata mesajı gelmesini istemiyorsanız programı başlatırken hata mesajlarını; daha doğrusu **STDERR**'i bir dosyaya ya da dipsiz kuyuya yönlendirebilirsiniz.

```
cp -r /home/cayfer/tmp/* /yedik 2> /tmp/hatalar
cp -r /home/cayfer/tmp/* /yedik 2> /dev/null
```

Dikkat ederseniz, **STDERR**'i yönlendirirken “2>” karakterleri kullanılıyor. Oysa **STDOUT** yönlendirilirken yalnızca “>” kullanılıyordu.



Ön Planda Çalışan Programları Arka Plana Atmak

Bazı durumlarda, başlattığınız bir programın ne kadar süreyle çalışacağını önceden kestiremezsiniz. İşin uzun süreceğini ve sessiz çalışan bir iş olduğunu sonradan fark edersiniz ya da işin bu özelliklerini bilseniz bile, boş bulunup komut satırının sonuna “&” koymadan Enter tuşuna basıverirsiniz. Örneğin, bir dizindeki tüm dosyaları bir başka diske ya da dizine kopyalama komutunu,

Kim Korkar LINUX'tan?

```
cp -r /home/ayfer /disk2/home2
```

şeklinde verdiğiniz ve programı ön planda başlattığınızı varsayalım. Başlattıktan birkaç saniye (ya da birkaç dakika) sonra işin uzun süreceğini fark ettiniz ve “*Tüh! Keşke arka planda başlatsaydım!*” dediniz. Eğer kabuk programı olarak **bash** kullanıyorsanız sorun değil...

Klavyenizden,

```
^z (Ctrl-Z)
```

tuşuna basarsanız (Control tuşu basılıyken Z tuşuna da basarsanız) ekranda,

Suspended.

mesajını görürsünüz. Bu mesaj, o sırada ön planda çalışan işinizin geçici olarak askıya alınarak durdurulduğunu göstermektedir. Ancak, buradaki “durduruldu” ifadesi, işinizin tamamlanmadan kesildiği anlamında değildir. Buradaki durdurma, müzik kaseti çalan teyplerdeki Pause düğmesinin görevine benzeyen bir durdurmadır. İşiniz çalışmaya ara vermiş ve devam edebilmek için sizden bir komut bekler durumdadır.

Bu noktada,

```
bg (Background)
```

komutu verirseniz, işiniz arka planda çalışmaya devam edecektir. Ancak, o anda ekranda bir de,

```
[1] cp ... &
```

mesajı görünecektir. Bu mesajın anlamı kısaca programınızın geri plana çekildiğini ve geri planda çalışan işleriniz arasındaki numarasının 1 olduğunu. Geri plandaki 1 numaralı işi tekrar ön plana almak isterseniz,

```
fg %1 (Foreground)
```

komutunu verebilirsiniz. (Eğer arka plana atılmış tek bir işiniz varsa, “%” işareti ve ardındaki numarayı girmeniz gerekmez.)

Arka planda çalışmak üzere başlatılacak, ya da sonradan arka plana atılacak işlerin sayısı ile ilgili herhangi bir sınırlama yoktur. Ancak, arka plan ya da ön plan olsun, çalışan her işin bilgisayarın performansından bir pay alacağını unutmamalısınız.

Bazen, arka planda başlattığınız ya da sonradan arka plana attığınız işlerin hesabını şaşırtabilirsiniz. Böyle bir durumda,

jobs

komutunu verirseniz, arka plana atılmış işlerin bir listesini alırsınız.

Kabuk Programlama

Bu bölümde amacımız, okuyuculara kabuk programlamayı öğretmek değil, sadece bu kavramın nasıl bir şey olduğu konusunda fikir vermektir. Aslında oldukça karmaşık bir iş olan ve deneyim isteyen kabuk programlama, programcılık deneyimi olmayan LINUX kullanıcılarının pek ilgisini çekmemekle birlikte programcılık temeli olan okuyuculara oldukça ilginç gelebilir.

LINUX kabukları (**bash**, **csh**, **tsch** gibi) aslında oldukça gelişmiş birer programlama dilini çözümlenebilecek yeteneğe sahiptirler. Genel amaçlı işler için pek kullanışlı olmamakla birlikte ileri düzeydeki kullanıcıların ve sistem yöneticilerinin en değerli araçlarından biridir.

csh ve **bash** kabuk programlama dilleri birbirlerinden oldukça farklıdır. Bu kitapta vereceğimiz örnekler yalnızca **bash** kabuğu için olacaktır.

Her ne kadar LINUX kullanmak için **bash** ya da bir başka kabukla programlama yapmaya gerek olmasa da LINUX/UNIX sistem yöneticisi olmayı düşünenlerin bu işlerden biraz anlaması çok yararlı olacaktır. Kabuk programlama konusunda daha fazla ayrıntıya girmek isteyen okuyucular için internet'te "*bash programlama*", "*bash programming*" anahtar sözcükleriyle bir tarama yapmalarını öneririz.

Şimdi isterseniz birkaç örnek kabuk programına göz atalım:

İlk Kabuk Programı Örneği

İlk örnek çok da anlamlı bir iş yapmamakla birlikte kabuk programlarının neye benzediğini göstermek açısından tipik bir **bash** kodudur.

Öncelikle aşağıdaki LINUX komutlarını **vi** editörünü kullanarak “**ornek1.sh**” isimli bir dosyaya kaydediniz. (Aslında **vi** editörünü kullanmak zorunda değilsiniz elbette ama kabuk programlamayla ilgilendiğimize göre sistem yöneticisi olma yolunda ilerliyorsunuz demektir; bu durumda da **vi** bilmeden olmaz. Neyse, şimdilik istediğiniz editörü kullanabilirsiniz.)

```
#!/bin/bash
COUNT=1
while [ $COUNT -le 100 ]
do
ls | wc -l
sleep 2
COUNT=`expr $COUNT + 1`
done
```

Yukardaki **bash** kabuk programı iki saniyede bir toplam 100 kez “**ls**” komutunu çalıştırıp bu komutun ürettiği satırları sayacaktır.

Şimdi bu kısa bash programının satırlarına bir göz gezdirelim:

#!/bin/bash: **bash** kabuk programlarında açıklama satırları (*comment*) “**#**” ile başlar. Bu satır ilk bakışta bir açıklama satırı gibi görünse de aslında çok özel bir anlamı vardır. Bir kabuk programının ilk satırı olması; ayrıca ilk iki karakterinin “**#!**” olması, bu kabuk program dosyasındaki satırların **/bin/bash** programıyla yorumlanması gerektiğini belirtmektedir. Bir kabuk programının başında, programın hangi kabukla çalıştırılacağını belirtmesi oldukça önemlidir. Bu satır olmadığında, kabuk programı yalnızca **bash** kabuğu ile çalışan kullanıcılar tarafından başarıyla çalıştırılabilir. Oysa bu satır sayesinde, kullanıcının komutu verdiği sırada kullandığı kabuk ne olursa olsun, önce **/bin/bash** kabuğu başlatılacak, program da bu kabuk içinde çalıştırılacaktır. Program bittiğinde bu program için başlatılmış kabuk da bitecek ve kullanıcı kendi kabuğuna geri dönecektir.

COUNT=1: **COUNT** isimli bir değişkene 1 değeri atanıyor. Kabuk değişkenlerinin isimleri genellikle büyük harflerle yazılır. Bu kesin bir kural olmamak-

la birlikte kullanılacak değişkenlerin isimlerinin LINUX komutlarıyla karışmaması için yararlı bir gelenektir.

while [\$COUNT -le 100]: \$COUNT değişkeninin değeri 100'den küçük veya eşit olduğu sürece tekrarlanacak bir döngü başlıyor.

do: while kalıbının bir parçasıdır. Döngü içindeki komut satırları “**do-done**” deyimleri arasına yazılır.

ls | wc -l: “ls” komutu ve “wc -l” komutları birlikte çalıştırılıp ls komutunun çıktıları wc komutuna girdi olarak yönlendirilip satırlar sayılıyor.

sleep 2: İki saniye beklemek için sleep komutu.

COUNT=`expr \$COUNT + 1`: \$COUNT değişkeninin değerini bir arttıran komut.

done: while döngüsündeki deyim grubunun sonunu belirliyor.

ornek1.sh isimli dosyadaki bu **bash** komutlarını bir program gibi çalıştırmak için sisteme bu dosyanın “**çalıştırılabilir**” program dosyası olduğunu belirtmek gerekir. Bu komutu kimlerin kullanabileceğine bağlı olarak

chmod a+x ornek1.sh

veya

chmod u+x ornek1.sh

veya

chmod 755 ornek1.sh

gibi bir komutla dosyanın çalıştırma yetkilerini düzenleyebilirsiniz.

Son olarak da komutu çalıştırmak için:

./ornek1.sh

komutunu verebilirsiniz. Eğer **PATH** değişkeninizde “çalışma dizini” anlamında “.” yoksa, komutun başına “bu dizindeki” anlamına gelen “./” işaretini koymanız; yani komutu “**./ornek1.sh**” şeklinde yazmanız gerekecektir.

İkinci Kabuk Programı Örneği

Bu örneğimiz ileri düzey bir **bash** programı olacak. Amacımız **bash** programlamanın tüm detaylarını öğretmek olmadığı için bu programın ayrıntılarına girmeyeceğiz. Programcılık deneyimi olup bash programlamayı ayrıntılı olarak öğrenmek isteyen kullanıcıların başka kaynaklara yönelmesi gerekecektir.

```
#!/bin/bash
# lsx : Dizin yapısını hiyerarşik olarak görüntüler.

ara () {
    # Fonksiyon tanımı
    for DIZIN in `echo *`
    do
        if [ -d $DIZIN ] ; then
            W_DUZEY=0
            while [ $W_DUZEY != $DUZEY ]
            do
                echo -n "| "
                W_DUZEY=`expr $W_DUZEY + 1`
            done
            if [ -L $DIZIN ] ; then
                echo "+---$DIZIN" `ls -l $DIZIN | sed 's/^.*'$DIZIN' //'`
            else
                echo "+---$DIZIN"
                if cd $DIZIN ; then
                    DUZEY=`expr $DUZEY + 1`
                    ara
                    DIR_SAYISI=`expr $DIR_SAYISI + 1`
                fi
            fi
        fi
    done
    cd ..
    if [ $DUZEY ] ; then
        DEVAM=1
    fi
    DUZEY=`expr $DUZEY - 1`
}
```

```

#
# Başlıyoruz... (Ana program)
#
CALISMA_DIZINI=`pwd`
if [ $# = 0 ] ; then
    cd `pwd`
else
    cd $1
fi

echo "Başlangıç dizini : `pwd`"
DEVAM=0
DUZEY=0
DIR_SAYISI=0
W_DUZEY=0
while [ $DEVAM != 1 ]
do
    ara
done
echo "Toplam $DIR_SAYISI dizin bulundu."
#
# Program başlatıldığındaki çalışma dizinine geri dön
cd $CALISMA_DIZINI

```

Yukardaki programı dikkatle incerseniz C, PASCAL gibi genel amaçlı programlama dillerindeki komut yapılarının eşdeğerlerini, hatta özyinelemeli (*recursive*) fonksiyonların dahi kullanılabilirdiğini göreceksiniz. Hatırlarsanız, **bash** kabuğunun aslında çok gelişmiş bir programlama aracı olduğunu daha önce de vurgulamıştık.

Şimdi bu **bash** programını bir UNIX komutu gibi kullanabilmek için yapmanız gerekenleri bir gözden geçirelim:

Öncelikle **bash** kodunu içeren dosyanın “çalıştırılabilir” bir dosya olması gerekir:

```

chmod 755  lsx          veya
chmod a+x  lsx

```

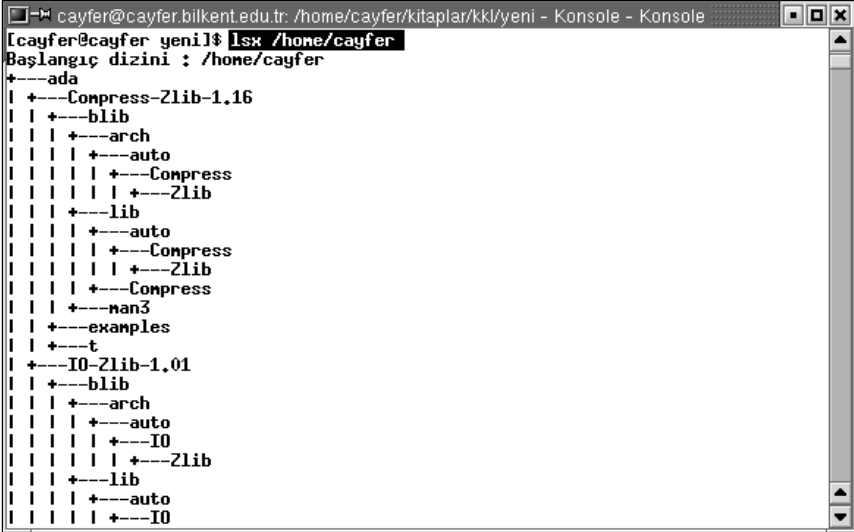
gibi komutlar bir dosyayı herkes tarafından çalıştırılabilir hale getirilebilir.

Kim Korkar LINUX'tan?

Dosyaya herkes kolayca erişebilsin diye dosya, kodu yazan programcının kişisel dizininden, **/usr/local/bin** dizinine kopyalanmalıdır. **/usr/local/bin** dizinine dosya kopyalayabilmek için root kullanıcı kimliğine bürünmek gerekecektir:

```
su -  
cp /home/cayfer/lxx /usr/local/bin
```

Artık **PATH** değişkeninde **/usr/local/bin** olan tüm kullanıcılar rahatlıkla yeni **lxx** komutunuzu kullanabilecektir.



```
cayfer@ cayfer.bilkent.edu.tr: /home/cayfer/kitaplar/kkl/yeni - Konsolle - Konsolle  
[cayfer@cayfer yeni]# lsx /home/cayfer  
Başlangıç dizini : /home/cayfer  
+---ada  
| +---Compress-Zlib-1.16  
| | +---bilib  
| | | +---arch  
| | | | +---auto  
| | | | | +---Compress  
| | | | | | +---Zlib  
| | | | | +---lib  
| | | | | +---auto  
| | | | | +---Compress  
| | | | | | +---Zlib  
| | | | | +---Compress  
| | | | | +---nan3  
| | | | | +---examples  
| | | | | +---t  
| +---IO-Zlib-1.01  
| | +---bilib  
| | | +---arch  
| | | | +---auto  
| | | | | +---IO  
| | | | | | +---Zlib  
| | | | | +---lib  
| | | | | +---auto  
| | | | | +---IO
```