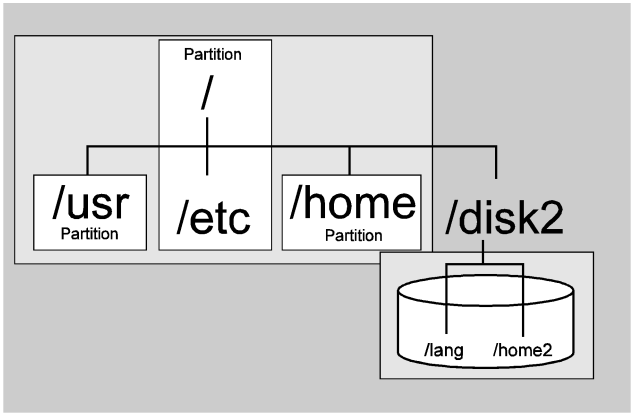
Şimdi sıkı durun: UNIX kullanıcılarının, disklerin ne şekilde ayrılmış olduğundan; hatta bilgisayarda kaç disk sürücü bulunduğundan haberi olması bile gerekmemektedir. UNIX'de tüm diskler ve disk parçaları (*partition*'lar), root (/) dizinin altında birer alt dizin olarak yer alacaktır.

Şematik olarak göstermek gerekirse :



Her disk ve disk parçası üzerinde diğerlerinden bağımsız bir **dosya sistemi** (**file system**) bulunmalıdır. Bu sistemler, diskler (ya da disk parçaları) üzerinde, formatlama işleminden sonra **mkfs** komutu ile yaratılmaktadır. (Merak etmeyin; bu işin yapılmasından siz değil; sistem yöneticiniz sorumludur.)

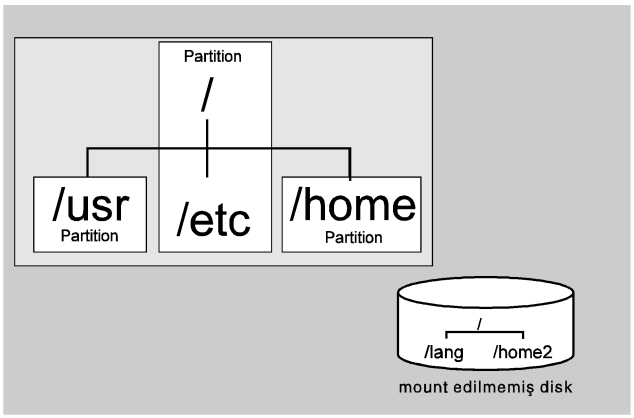
Genellikle, **boot** diskinizin (bilgisayar açıldığında UNIX işletim sisteminin yüklendiği disk) ilk parçası size root (**/**) dizini olarak görünür. Diğer disk ve disk parçalarıysa bu dizinin altındaki alt dizinler olarak görünür. UNIX geleneğine göre **boot** diskleri an az 3 parçaya bölünür. İlk parça **/**, ikinci parça **/usr**, üçüncü parça ise **/home** dizini olarak isimlendirilir. Aslında pek yeri değil ama sanırım biraz daha ayrıntılı açıklama yararlı olacak.

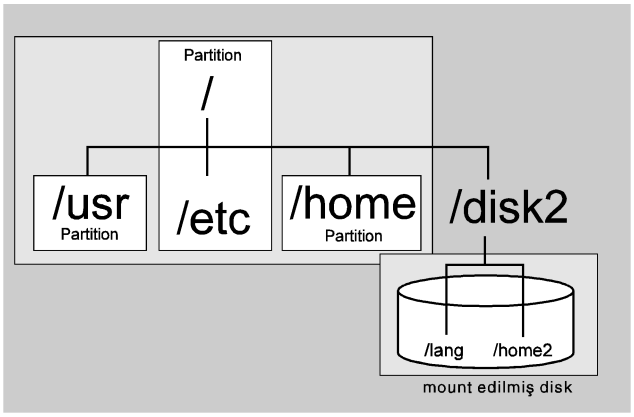
/ dizini, bilgisayarın açılabilmesi için gerekli olan dosyaların ve alt dizinlerin yer aldığı dizin; **/usr** dizini, tüm kullanıcıların ortak olarak kullanacağı çeşitli derleyici ve servis programlarının yer aldığı dizin; **/home** diziniyse, adından da anlaşılacağı gibi kullanıcıların kendilerine özgü dosyalarını yerleştirecekleri **home** dizinlerinin yer aldığı dizindir. Bu yerleştirme tarzı UNIX geleneğinin bir parçasıdır. Aynen uygulanması gerekmese de, genellikle tüm UNIX sistemlerinde diskler bu veya buna çok benzeyen bir şekilde düzenlenir. Bu düzenlemenin yararlarını daha ilerideki bölümlerde (özellikle sistem yöneticilerini ilgilendiren konulara gelince) anlatacağım.

Üzerinde bir dosya sistemi olan bir disk birimine veya parçasına, okuma veya yazma amacıyla ulaşabilmeniz için, o dosya yapısının, **/** dosya yapınızda bir alt dizine **mount** edilmiş olması gerekmektedir. (**/** dizini, bilgisayarın açılması sırasında otomatik olarak **mount** edilmektedir. Eğer bu **/** dizini, bilgisayarın açılması aşamasında **mount** edilemezse, o bilgisayar zaten açılamaz; bu durumda mutlaka teknik desteğe gereksiniminiz vardır. Diğer disk veya disk

parçalarının otomatik olarak **mount** edilmesi için gerekli işlemlerse, sistem yöneticiniz tarafından yapılmalıdır.

UNIX'deki dosya-dizin yapılarını ters duran bir ağaca benzetirsek, **mount** etme işlemini, bir ağacı, bir başka ağacın dallarından birine iliştirmek (monte etmek) gibi düşünebilirsiniz.





Şimdi isterseniz, kullandığınız bilgisayarda kaç disk ve/veya disk parçası olduğuna ve bunların hangi dizinlere **mount** edildiğine bir bakalım. Bu iş için lütfen terminalinizden şu komutu veriniz :

```
% mount
```

Tipik olarak şöyle bir liste almalısınız

```
% mount
/dev/sd0a on / rw 4.2
/dev/sd0g on /usr rw 4.2
/dev/sd0h on /home rw 4.2
%
```

(Bu örnek, SUNOS 4.1.3 UNIX işletim sistemiyle çalışan bir iş istasyonundan alınmıştır. Sizin kullandığınız bilgisayarda alacağınız liste bununla aynı olmayabilir).

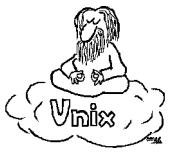
Bu listeden, bilgisayarımızda sadece bir disk olduğunu (sadece **/dev/sd0** serisi bulunduğundan anlaşılıyor) ve bu diskin en az üç parçaya ayrıldığını (parça isimleri a, g ve h) veya sadece üç parçasının o anda **mount** edilmiş durumda olduğunu (**a** parçası **/** olarak, **g** parçası **/usr** dizini olarak ve nihayet **h** parçası da **/home** dizini olarak) anlıyoruz. **mount** komutunun verdiği listedeki satırlarda yer alan **rw** harfleri, söz konusu disk parçalarının oku-yaz ***(read-write)*** olarak kullanıma sunulduğunu (tabii ki, kullanıcıların yetkilerinin izin verdiği ölçüde) belirtmektedir. 4.2 sayısıysa, SUNOS 4.1.3 işletim sistemine özgü bir dosya sistemi sürüm kod numarasıdır *(File System Version).*

mount komutu hakkında daha detaylı bilgiyi sistem yönetimi ile ilgili bölümlerde bulabilirsiniz. UNIX işletim sisteminin bir çok türevinde **mount** komutunu parametrelerle birlikte kullanabilmeniz için süper kullanıcı yetkilerine sahip olmanız gerekecektir; yani eğer süper kullanıcı (**root**) değilseniz, zaten, **mount** komutunu yalnızca parametresiz olarak kullanmanıza izin verilecektir.

Bir UNIX bilgisayarı açıldığında, otomatik olarak **mount** edilmesi istenen diskler ve **mount** edilecekleri dizinler **/etc/fstab** (BSD UNIX) veya **/etc/vfstab** (SVR4 UNIX) dosyalarında tanımlanır. Bu dosyalara sadece **root** kullanıcının yazma yetkisi vardır; bu nedenle bu dosyalara korkmadan bakabilirsiniz. (**more /etc/fstab** gibi bir komut işe yarayabilir; ne dersiniz?)

Disket sürücüler ve CD-ROM sürücüleri de küçük birer disk sürücü olarak düşünülebilirler; bu nedenle kullanılabilmeleri için önce **mount** edilmeleri gerekir. Ancak; hem disketler, hem CD'ler, takılıp çıkarılabilir birimler olduklarından, bilgisayar açılırken otomatik olarak **mount** edilmezler. Normal olarak, bir disket veya CD'yi **mount** etmek için yeterli yetkiniz olmayacağından, bu tip birimlerin **mount** edilmesi konusunda sistem yöneticisinden yardım istemelisiniz. **mount** edilecek birimin, **mount** işleminden sonra hangi dizin altında görünmesi gerektiğine siz karar verebilir ve bu dizini siz yaratabilirsiniz. Bazı sistem yöneticileri, normal kullanıcılar tarafından çalıştırılabilen ve disket/CD **mount** işlemini yapan komutlar yaratırlar. Sizin çalıştığınız sistemde de böyle bir olanak olup olmadığını araştırınız. (Hayat Bilgisi kitabı gibi oldu, değil mi?)

İşi biten disket ve CD'ler **unmount** edilmelidir; yani, bu birimlere takılı medyalar üzerindeki dosya sistemlerinin, **root** dosya sistemiyle bağlantısı kesilmelidir. Aynı **mount** komutu gibi **unmount** komutu için de yöneticinizin yardımına gereksiniminiz olabilir..



Haaaaa. Bu arada.... Sistem yöneticinizle iyi geçinin. Zaman zaman kendisinin ne kadar inanılmaz yeteneklerle donanmış olduğunu; bu kadar çok şeyi bilebildiğine göre dehşetli zeki olduğunu kendisine hatırlatmayı ihmal etmeyin. Aslında bu özelliklerini kendisi çok iyi biliyordur; ama gene de hatırlatılmasından hoşlanacaktır. Arada bir ona hediyeler alın; özellikle sizin için geç saatlere kadar çalışacaksa yemek zamanında onun için bir pizza ve büyük kola getirmeyi sakın ihmal etmeyin. Yalnız dikkatli olun; UNIX guruları, içinde yeşil malzeme olan pizza yemezler (UNIX geleneği)!

Süreçler *(Processes)*

UNIX işletim sisteminin **çok kullanıcılı** ve **çok işli** bir işletim sistemi olduğunu şimdiye kadar bir kaç kez vurgulamıştım. Burada bir daha açıklamak gerekirse; UNIX işletim sisteminin denetimindeki bir bilgisayar hem aynı anda birden fazla kullanıcıya hizmet edebilir, hem de her kullanıcının aynı anda birden fazla işi yapmasına olanak sağlar. UNIX, kendi işlerini de bir sürü programı aynı anda çalıştırarak yapar. Örneğin, kullanılmayan terminallerin açılıp açılmadığını kontrol eden **getty** (bazı UNIX'lerde **init**) programları, kullanıcıların birbirlerine gönderdikleri mesajları gözleyen ve gelen-giden mesajları uygun posta kutularına yönlendiren **mail server** programı, bilgisayar ağı üzerinden gelen istekleri değerlendiren **inetd** programı, belirli aralıklarla disklere yapılan kayıt işlemlerinin fiziksel olarak disklere kaydedilmesini *(flushing disk buffers)* sağlayan **update** programı gibi... (Tipik bir UNIX bilgisayarında, kullanıcı programları dışında 30-40 sistem programı sürekli çalışıyor durumdadır.)

Bir UNIX bilgisayarında, belirli bir anda, merkezi işlem birimini (ya da birimlerini) ve belleği paylaşarak, birlikte çalışan programlara genel anlamda **PROCESS** (**süreç**) adı verilir. Süreçlerin Merkezi İşlem Birimi (MİB) zamanını paylaşmaları UNIX tarafından koordine edilir (İşletim Sistemlerine ilişkin İngilizce terminolojide : *Process Scheduling* işlevi). MİB paylaşımına ilişkin önemli bir terim de **'zaman dilimi'** *(time slice)* kavramıdır. Her süreç, MİB'ni belirli ve kısa bir süre için (tipik olarak 10 - 100 milisaniye) sürekli olarak kullanabilir. Zaman dilimini dolduran süreçler beklemeye alınıp, MİB sırada bekleyen bir başka sürece tahsis edilir. Bu şekilde tüm süreçler aynı anda çalışıyormuş gibi bir etki elde edilir. Bu süreçlerin birden fazla kullanıcıya ait olmaları durumunda da, MİB kullanıcılar arasında paylaştırılmış olur. UNIX'in çok kullanıcılı olma özelliğinin altında yatan temel mekanizma budur

Herhangi bir anda, bilgisayarda çalışan süreçlerin neler olduğunu görmek isterseniz kullanacağınız komut

```
% ps -axl                    Berkeley UNIX için process status
% ps -efl                    SVR4 UNIX için process status
```

olacaktır. Aslında, bu komutun gerek BSD (Berkeley), gerekse SVR4 UNIX için daha bir çok parametresi olabilmektedir. Bu parametreleri merak eden kullanıcılar, **man** komutu yardımıyla kullandıkları UNIX'in **ps** komutunun detaylarını öğrenebilirler.

Komutu parametresiz kullanırsanız, yanlızca kendinize ait süreçlerin bir listesini alırsınız. (Alacağınız bu liste BSD veya SVR4 UNIX'ler için biraz farklı olacaktır, fakat içerdikleri bilgi açısından eşdeğer sayılabilirler; bu nedenle sadece BSD UNIX'den örnekler vereceğim).

```
% ps

  PID TT STAT  TIME COMMAND
 1210 co IW    0:01 sunview
 1234 p0 S     0:43 shelltool
 1226 p0 R     2:46 shelltool
 1456 co R     0:04 ps
 1605 co S     0:01 -bin/csh (csh)
. . .
. . .
%
```

Bu listedeki önemli bilgiler şunlardır:

PID — *(Process ID)* UNIX'de, her sürecin birer tanıtım numarası vardır. Aynı numaraya sahip iki süreç olamaz.

TT — (*Teletype* : Çok eskilerden kalan bir alışkanlık) Sürecin hangi terminalden başlatıldığı (genellikle **co** : *console*, **ttya** : a isimli seri arabirim, **p0** bilgisayar ağı üzerinden bağlanmış bir ekran). TT bilgisi kullandığınız donanımın özelliklerine göre değişebilir.

STAT — *(Status)* Sürecin bulunduğu duruma ilişkin bir kod.
R: *Runnable* : Çalışabilir durumda, sırasını bekliyor
S: *Sleeping* : Uyuyor
Z: *Zombie* : Bu süreç ile ilgili tüm diğer süreçler bitmiş veya ölmüş; bununda bitmiş olması gerekirdi ama bir nedenle **ölememiş**. **ps** listesinde hala görünüyor olması zararsızdır.

TIME — Sürecin ne kadar zamandır çalıştığını gösterir

COMMAND — Süreci başlatan komut satırı

Süreçler hakkında daha detaylı bilgi isterseniz :

```
% ps -l                                    (process status -long list)
```

Sistemdeki tüm süreçler hakkında bilgi isterseniz :

```
% ps -ax                                   (process status  -all, extended)
% ps -ef
```

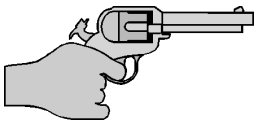
Sistemdeki tüm süreçler hakkında detaylı bilgi isterseniz :

```
% ps -axl                        (process status -all, extended, long)
% ps -efl
```

ps komutu, sistem yöneticileri için (***sysad***, ***sysadmin*** : *system administrator*) için çok önemlidir. Sistemde neler olup bittiğini, kullanıcıların ne gibi programlar kullanmakta olduklarını, sisteme kullanıcıların nerelerden eriştiğini, hep bu komut yardımı ile gözlerler. Ayrıca, sistemin çalışmasında bir gariplik olduğu zaman hemen bu komutla bilgisayarda çalışmakta olan süreçlerin bir listesini alırlar.

ps komutu, zaman zaman normal kullanıcılar için de çok önemli olur. İşte size hemen bir senaryo...

Süreç Öldürme *(Killing Processes)*



Diyelim ki başlattığınız bir iş kontrolden çıktı ve istediğiniz ya da beklediğiniz gibi davranmıyor. Doğal olarak bu işi hemen kesmek istiyorsunuz. İlk denemeniz gereken Ctrl-C tuşu. Olmazsa Ctrl-D tuşu... (Fazladan basacağınız Ctrl-D **logout** edilmenize neden olabilir.) Gene olmadı diyelim. **BİLGİSAYARI ELEKTRİK ANAHTARINDAN KAPATMAYI VEYA RESET DÜĐMESİNE BASMAYI AKLINIZDAN DAHİ GEÇİRMEMELİSİNİZ!**

Böyle bir durumda, eğer yapabiliyorsanız, ekranınızdaki başka bir pencereden veya bir başka kullanıcı terminalinden :

- Uygun bir **ps** komutuyla (**ps** veya **ps -axl** veya **ps -efl**) çalışmakta olan süreçlerin bir listesini alın.
- Bu listeye bakarak, sorun çıkaran sürecin numarasını öğrenin ve

```
% kill nnn                                    (process kill)
```

(Burada **nnn**, öldürülmek istenen sürecin numarasıdır).

- Eğer sorun yaratan süreci bu komutla öldüremezseniz

```
% kill -9 nnn
```

komutunu deneyiniz. (-9 seçeneği *'koşulsuz öldürme'* isteğinizi belirtir.)

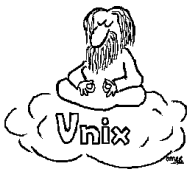
```
% ps
  PID TT STAT  TIME COMMAND
1234 p0 S     0:43 shelltool
1266 p0 R     2:46 problemli prog
1456 co R     0:04 ps
1605 co S     0:01 -bin/csh (csh)
% kill 1266                              olmazsa
% kill -9 1266
```

Süreci hala öldüremiyorsanız, kabuk programınızı öldürmeyi deneyiniz. Hala direniyorsa sistemin **USULÜNE UYGUN OLARAK** kapatılmasını sağlayınız. Eğer **root** yetkilerine sahip olabiliyorsanız, bu işi kendiniz de yapabilirsiniz; ancak sistemde başka kullanıcılar olabileceğini unutmayıp, bu kullanıcılara bir mesaj gönderip (**write** ve **wall** komutları), onlara makul bir süre tanıyıp; ancak ondan sonra **shutdown** komutunu kullanarak sistemi kapatınız. Sistemleri kapatma yöntemlerini daha sonraki bölümlerde anlatacağım.

Link Kavramı ve ln Komutu

Şimdi biraz mistik bir konudan söz edeceğim. UNIX işletim sistemi altında **bazı dosyalar aslında bulundukları yerde olmayabilirler.** Evet, yanlış okumadınız! Diskin üzerinde yer alan bazı dosyalar aslında orada olmayabilir; hatta bir dosyanın sistemde tek bir kopyası olmasına rağmen, bu dosya birden fazla dizinde; üstelik farklı isimlerle yer alabilir. Kavraması ve kullanması zor bir kavram fakat bir kez mecbur kalıp da kullandığınızda hoşunuza gideceğine emin olabilirsiniz.

Sanırım en iyisi bir örnekle anlatmak : 



Farzedin ki bir UNIX sisteminin yöneticisisiniz. Sizden, bilgisayara **matlab** isimli yeni bir uygulama programı yüklemenizi istediler. Ancak, uygulama programının bir gereği olarak, program paketine ilişkin dosyaların **/usr/local** dizininin altında açılacak bir dizinde yer alması gerekiyor. Eh! Olabilir. Ancak, bir sorun var! **/usr** diskinde, yeni programa ilişkin dosyalar için yeterli boş yer yok; ve silebileceğiniz gereksiz dosyalar da yok!

Mistik **ln** kavramını kullanarak bu işi UNIX'in şanına yaraşır bir yöntemle çözebilirsiniz. Disklerin birinde; örneğin **/home** dizininin bulunduğu disk bölümünde *(partition)*, yeni yükleyeceğiniz program için bir dizin yaratınız :

```
# mkdir /home/matlab
```

Sonra, bu dizini, **/usr/local** altında yer alıyormuş gibi gösterebilmek için

```
# ln -s /home/matlab  /usr/local/matlab
```

komutunu veriniz.

Böylece, gerçekte **/home** altında yer alan **matlab** dizini, aynı zamanda **/usr/local** altında da varmış gibi olacaktır. Bu dizini kullanırken isterseniz **/home/matlab**; isterseniz **/usr/local/matlab** dizin adreslerini kullanabilirsiniz. Bir başka deyişle, dosyalarının /usr/local altında bulunmasını isteyen matlab yazılımını kandırmış olursunuz.

link kavramının çok işe yarayabileceği, bir öncekine benzeyen bir senaryo daha anlatabilirim. Diyelim ki elinizde **mhsb1995** isimli bir dosya var ve muhasebe departmanının kullandığı muhasebe programı bu dosyayı mutlaka bu isimde görmek istiyor. Öte yandan yeni satın aldığınız bir mali analiz programı, aynı muhasebe verilerini **acct95** adıyla görmek istiyor.

Söz konusu dosyanın adı **mhsb1995** olduğu zaman muhasebe departmanının sorunu yok ama siz mali analiz programını çalıştıramıyorsunuz. Analiz çalışmaları için dosyanın adını değiştirseniz, siz çalışabiliyorsunuz ama bu sefer muhasebe departmanındaki program kullanılamıyor. Dosyanın adını **mhsb1995** olarak tutup, kendi analiz çalışmalarınız için **acct95** adlı bir kopyasını çıkardığınızda ve siz bu kopya üzerinde çalıştığınızda problem kısmen çözülüyor ama çok kullanıcılı ortamda siz analizler üzerinde çalışırken öte taraftan muhasebe personeli yeni kayıtlar girip sizin analizlerinizin eskimiş kayıtlar üzerinde yapılmasına neden oluyorlar. İşte böyle bir durumda **link** kullanımı sizi kurtaracaktır.

```
# ln ./mhsb1995 ./acct95
```

Bu komutla **mhsb1995** dosyasını **acct95** isimli bir dosyaya bağladığınızda (aslında sadece tek bir asıl kopya var; o da **mhsb1995**. **acct95** isimli bir dosya aslında yok sadece diğer dosyanın bir başka adı.) Bu sayede **mhsb1995** dosyasında yapılan her değişiklik **acct95** diye tanınan dosyada da aynen gözlenebilecektir. İşin bir başka yaralı tarafı da; **acct95** isimli dosyanın diskte hiç yer kaplamayacak olmasıdır.

Bu örnekler arasında, dikkatinizi çekmiş olduğunu umduğum bir fark var. İlk örnekte (matlab), **ln** komutunda **-s** diye bir parametre kullandım; oysa ikinci muhasebe örneğinde kullanmadım!

- Eğer, **ln** komutuyla birbirlerine bağlanacak olan dosya sistemi elemenları birer dizinse; **-s** parametresini kullanmak zorundasınız.
- Eğer, **ln** komutuyla birbirlerine bağlanacak olan dosya sistemi elemanları birer dosyaysa ve farklı disk parçalarında (bir başka deyişle; farklı dosya sistemleri altında) yer alıyorlarsa, gene **-s** parametresini kullanmak zorundasınız.
- **ln** komutuyla, bir dizini ve bir dosyayı birbirlerine bağlayamazsınız. Bağlanacak olan elemanların ikisi de dizin; ya da ikiside dosya olmalıdır.

Aynı dosya sisteminde yer alan ve birbirine bağlı olan dosyalardan birini silmeniz diğerini etkilemez. Asıl dosyayı silseniz bile, UNIX, bağlantıyı farkedip dosyayı diskten gerçekten silmeyecektir. UNIX, her dosya için bağlantıları sayar ve her silme işleminde bağlantı sayısını bir azaltır. Gerçek silme işi bu bağlantı sayısı sıfırlanınca yapılır.

Farklı dosya sistemlerinde yer alan bağlantılar için, bu bağlantı sayma işine güvenmeyiniz. Farklı dosya sisteminde bağlantısı olan bir dosyayı silerseniz başınız derde girer. Asıl dosya silinir ve diğer sistemde, gerçekte var olmayan bir dosyayı gösteren bir bağlantınız kalır.

Bir dosyanın gerçekten var olan bir dosya mı, yoksa sadece bir bağlantı mı (**link**) olduğunu anlamak için **ls** komutunu **-l** seçeneği ile kullanmanız gerekir. İçinde bağlantılı dosyalar bulunan bir dizinde **ls -l** komutunu vererek, alacağınız listede bağlantılı dosyaları ve hangi dosyaya bağlantılı olduklarını açıkça görebilirsiniz.

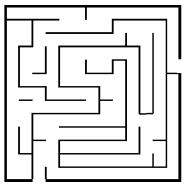
```
/home/ayfer % cd /
/ % ls -l
total 3166

lrwxrwxrwx  1 root        7 Jan  12 12:09 bin -> /usr/bin
-r--r--r--  1 root   110912 Jan  12 12:11 boot
drwxr-sr-x  2 bin      7680 Jan  12 12:23 dev
drwxr-sr-x  7 bin      1536 Jan  15 08:45 etc
drwxr-sr-x  4 root      512 Feb   1 11:56 export
drwxr-xr-x  5 root      512 Mar  23 09:03 home
-rwxr-xr-x  1 root   239783 Feb  09 13:34 kadb
lrwxrwxrwx  1 root        7 Mar  01 18:23 lib -> /usr/lib
drwxr-xr-x  2 root     8192 Jun  15 23:09 lost+found
drwxr-sr-x  2 bin       512 Mar  01 20:09 mnt
drwxr-sr-x  2 bin       512 Mar  09 08:59 sbin
lrwxrwxrwx  1 root       13 Jan  24 07:45 sys -> /usr/kvm/sys
drwxrwsrwt  2 bin       512 Feb  24 09:56 tmp
drwxr-xr-x 20 root      512 Nov  23 16:08 usr
drwxr-xr-x 11 root      512 Nov  23 16:11 var
-rwxr-xr-x  1 root  1101191 Jan  11 09:35 vmunix
/ %
```

Bu örnek listeye göre, aslında **/bin** diye bir dizin bulunmamakta, bu isimde bir bağlantının **/usr/bin** dizinine yapılmış olduğu anlaşılmaktadır.

Dikkat ederseniz, **ls -l** komutunun verdiği listede, gerçek bir dosya (dizin) değil de, bağlantı olan dosyalara (dizinlere) ait satırların başında bir **l** harfi bulunmaktadır.

İpin ucunu kaçırmayacağınıza eminseniz, bağlantılara bağlantı yapabilirsiniz.

Önemli UNIX Komutları



Günümüzün tipik UNIX bilgisayarlarında, GigaByte düzeyinde diskler bulunmaktadır. Bu kadar büyük disklerde de doğal olarak çok sayıda dizin ve binlerce dosya yer almaktadır. Zaman zaman adının bir kısmını hatırlayabildiğiniz; bulunduğu diziniyse bir türlü hatırlayamadığınız dosyalar olacaktır. Tek tek bütün dizinlere girip **ls** komutuyla bu dosya ya da dosyaları aramak pek akıllıca bir yöntem değildir. Böyle bir durumda kullanacağınız komut **find** dır.

```
find  baslama-dizini  kriter[ler] [-exec komut ";"]
```

find komutuyla yapabileceğiniz aramalarda tek kriter dosya adı değildir. Bu komutla

a) erişim yetkileri belirli bir kalıpta olan,
b) belirli özelliklere sahip,
c) belirli bir kullanıcıya ait,
d) belirli bir boydan büyük ya da küçük,
e) belirli bir tarihten veya saatten bu yana değişmemiş, erişilmemiş

dosyaları veya dizinleri bulabilirsiniz.

Üstelik verdiğiniz arama kriterlerine uyan dosyalar üzerinde uygulanacak UNIX komutlarını da **find** komutuna parametre olarak verebilirsiniz.

başlama-dizini

Arama işlemi, **find** komutunun bu ilk parametresinde belirtilen dizinden başlar ve varsa bu dizinin alt dizinleri de arama ağacına dahil edilir.

Eğer arama işleminin, bilgisayarınıza bağlı ve **mount** edilmiş tüm disklerinde yapılmasını istiyorsanız, bu ilk parametre olarak **/** sembolünü kullanınız.

Bilgisayarınızın CD-ROM sürücüsü varsa ve bu sürücüye bir CD takılıysa ve bu CD **mount** edilmiş durumdaysa ve arama, **/** dizininin hiyerarşisi boyunca yapılırsa, CD-ROM sürücüsünü de kapsayacaktır. CD'lerin kapasitelerinin büyüklüğü ve erişim hızlarının düşüklüğünden dolayı bu arama uzun sürecektir. Aynı mantıkla, bilgisayar ağı üzerinden başka bilgisayarların diskleri de sizin dosya sisteminize **mount** edilmiş durumdaysa, o diskler de arama ağacına girecektir. Zaman kaybına yol açmamak için, gerekmedikçe aramayı **/** dizininden başlatmamanızı öneririm.

kriter[ler] Aranan dosya ve dosyaların ortak özelliklerini tanımlayan kriterlerdir

Bir kaç örnek vermek gerekirse :

-name isim	adı "isim" olan dosyalar (farklı dizinlerde aynı isme sahip dosyalar olabilir)
-name "abc*"	adı "abc" ile başlayan dosyalar
-name "a*data"	adı "a" ile başlayan ve adının sonunda "data" olan dosyalar
-name "[a-k]95"	adı a95, b95, ..., j95 veya k95 olan dosyalar

Dikkatinizi çektiyse, **-name** kriteri kullanıldığında, dosya adını verirken, dosya adını tam olarak yazıyorsak tırnak (") kullanmıyoruz; oysa ***** karakterini içeren bir kalıp kullanıyorsak ***(wildcard)*** bu kalıbı tırnak (") içinde yazıyoruz.

Bunun nedeni şu :

Bir komut verdiğinizde, bu komut önce kabuk programınız tarafından irdelenir. Bu irdeleme sırasında rastlanan ***** karakterleri dosya adı kalıpları olarak kabul edilip, bu kalıba uyan dosya isimleriyle değiştirilmeye çalışılır. Oysa, kalıplara uyan dosya isimlerinin kabuk programı tarafından değil, **find** programı tarafından bulunması gerekmektedir. Kabuk programlarının irdeleme sırasında karşılaşacakları ***** karakterlerine dokunmamaları için, kalıp tanımları tırnak içine alınır.

`-user ayfer`	sahibinin adı **ayfer** olan dosyalar
`-group yonetim`	sahibi **yonetim** grubuna dahil olan dosyalar
`-perm 755`	erişim yetki düzeyi **755** olan dosyalar

`-newer dosya1`	**dosya1** isimli dosyadan daha sonraki bir saat ya da tarihte değişikliğe uğramış olan dosyalar
`-size 10`	diskte kapladığı alan **10 blok** olan dosyalar (1 blok = 512 Byte)
`-size +100`	diskte kapladığı alan **100 bloktan büyük** olan dosyalar (51 KByte'dan büyük dosyalar)
`-size -45`	diskte kapladığı alan **45 bloktan küçük** olan dosyalar
`-ctime 3`	Tam 3 gün önce değişikliğe uğramış olan dosyalar
`-ctime +8`	8 günden daha uzun bir süre önce değişikliğe uğramış olan dosyalar
`-ctime -8`	8 günden daha kısa bir süre önce değişikliğe uğramış olan dosyalar
`-mtime 3`	Tam 3 gün önce değişikliğe uğramış olan dosyalar
`-mtime +8`	8 günden daha uzun bir süre önce değişikliğe uğramış olan dosyalar
`-mtime -8`	8 günden daha kısa bir süre önce değişikliğe uğramış olan dosyalar
`-atime -3`	3 günden daha kısa bir süre içinde bir şekilde erişilmiş olan dosyalar

-ctime ve **-mtime** parametrelerinin her ikisi de dosyanın değişikliğe uğramasıyla ilgili süreleri kontrol eder; ancak aralarında küçük bir fark vardır. **-mtime**, dosyanın içeriğinde bir değişiklik yapılıp yapılmadığına; **-ctime** ise dosyanın içeriği yanısıra özelliklerinin de değişip değişmediğini kontrol eder. Örneğin, sahibi değişen bir dosya **-mtime** tarafından farkedilmezken **-ctime** tarafından dikkate alınır.

`-atime 3`	Tam 3 gün önce bir şekilde erişilmiş dosyalar
`-atime +8`	8 günden daha uzun bir süre önce erişilmiş olan dosyalar
`-atime -8`	8 günden daha kısa bir süre önce erişilmiş olan dosyalar
`-type f`	Basit birer 'dosya' olan dosyalar

```
-type d              Dizinler
```

Bu arama kriterlerini bir arada kullanabilirsiniz; örneğin, sahibi **hakman** adlı kullanıcı olan ve son 40 gündür kullanılmamış dosyaları bulmak isterseniz, kullanmanız gereken **find** komutu

```
% find /home -user hakman -atime +40 -print
```

olmalıdır.

Komutun sonundaki **-print** parametresini kullanmayı unutursanız, **find** programının verdiğiniz kriterlere uygun dosya bulup bulmadığını öğrenemezsiniz. Bulunan dosyaların isimlerinin listelenmesi için bu parametreyi kullanmak şarttır. İlk bakışta anlamsızmış gibi geldiğini biliyorum. Eğer bulunan dosyaların adını görmek istemiyorsanız, **find** komutunu neden kullanasınız ki? Bu sorunun yanıtı şöyle : **find** komutunu bir kabuk programı içinde çalıştırıyorsanız ve sizin için verdiğiniz kriterlere uygun dosya bulunup bulunmadığını bilmek yetiyorsa (hangi dosyalar olduğunu görmeniz gerekmiyorsa) aramanın başarılı olup olmadığını belirten bir sistem değişkeninin değerine bakmanız yeterli olacaktır.*(condition code veya completion code).*

Şimdi, sık kullanılan **find** formları için bir kaç örnek vereyim :

```
% find /home/ayfer -name onemli.dosya -print
```

/home/ayfer dizininden başlayarak bu dizinde ve alt dizinlerinde **onemli.dosya** isimli dosyaları arar ve bulduklarının adını standart çıktıya (ekrana) listeler.

```
% find / -name core -exec /bin/rm {} ";"
```

/ dizininden başlayarak tüm disklerde **core** isimli dosyaları arar ve bulduklarını siler.

find komutunu **-exec** parametresiyle birlikte kullanırken sondaki **";"** parametresini UNUTMAMALISINIZ. Bu **";"** karakter dizisinin gerekliliği tamamen **find** programının yazılışından kaynaklanmaktadır.

Bu komut, sistem yönetiminden sorumlu olanların oldukça sık kullanacakları bir komuttur. UNIX, çeşitli programların kullanımı sırasında bir sistem problemi olduğunda "**core dumped**" mesajıyla birlikte, belleği **core** isimli bir dosyaya kopyalar. Bu **core** dosyaları, problemin nedenini bulmasına yardımcı olmak amacıyla yaratılır. Bu dosyaları irdeleyerek problemin nedenini bulmak pek kolay

olmadığından, genellikle bu dosyalar içeriklerine bakılmaksızın silinebilirler. Zaman içinde biriken **core** dosyaları diskte oldukça önemli yer harcadıklarından, rastladıkça bu dosyaları silmenizi öneririm.

```
% find /home -user hasan -exec /bin/rm {} ";"
```

/home dizininden başlayarak **hasan** isimli kullanıcıya ait dosyaları arar ve bulduklarını siler.

Sisteme erişim hakları iptal edilen kullanıcılara ait dosyaları tek harekette silmek için kullanılabilir.

```
% find /home -name "*.tmp" -exec /bin/rm {} ";"
```

/home dizininden başlayarak adı ***.tmp** kalıbına uyan dosyaları arar ve bulduklarını siler.

```
% find /home -type d -name [tmp, temp] -print
```

/home dizininden başlayarak adı **tmp** veya **temp** olan dizinleri bulur ve listeler.

find komutuyla birlikte kullanılan kriterleri çeşitli mantık operatörleriyle birleştirebilirsiniz. Bunlar

```
-a  :  "ve"
-o  :  "veya"
\!  :  "de□il"
```

operatörleridir.

Örnekler :

```
% find /home -name "*.tmp" -a -size +100 -print
```

adı ***.tmp** kalıbına uyan **ve** büyüklüğü 100 bloktan fazla olan dosyaları bulur. (1 blok = 512 byte)

```
% find /home/ayfer \! -user ayfer -print
```

ayfer isimli kullanıcının home dizininde yer alan ama **ayfer**'e ait **olmayan** dosyaları bulur.

Bu örnekteki "**değil**" anlamında kullanılan **\!** operatöründeki **** işareti ardından gelen **!** işaretinin özel bir anlamı olduğunu ve kabuk programı (sh veya csh) tarafından yorumlanmaya çalışılmaması gerektiğini belirtmek için kullanılmaktadır.

Hatırlarsanız, daha önceki bölümlerden birinde, UNIX işletim sisteminde kendi komutlarınızı yaratabileceğinizden bahsetmiştim. Sanırım bu uygulamaya bir örnek vermek için uygun bir noktadayız.

find komutu oldukça yetenekli ve seçenekli bir komut; ama bunun karşılığında da yazması oldukça uzun. Dosyaları sadece adlarıyla arayan daha kısa bir UNIX komutu yaratmaya ne dersiniz?

Önce **vi** editörünü kullanarak home dizininizde **ff** isimli ve içinde aşağıdaki satırlar bulunan bir dosya yaratınız. **(% vi ~/ff)**

```
#!/bin/sh
case $# in
  1) find . -name "$1" -print;;
  2) find "$1" -name "$2" -print;;
  *) echo "Error... Usage : ff [path] name"
     echo "             ff [path]  \"name*\""
     echo "             ff [path]  \"*name\""
esac
```

ff program dosyasını oluşturan bu satırların anlamları üzerinde şimdilik durmayınız.

Daha sonra,

```
% chmod a+x ~/ff
```

komutuyla, bu dosyanın erişim yetki kalıbını, tüm kullanıcılar tarafından çalıştırılabilen bir komut dosyası olacak şekilde değiştiriniz.

Bu komut, aksi belirtilmedikçe, aramalara bulunduğunuz dizinden başlar. Eğer tek parametreyle çalıştırılırsa, bu parametreyi bir dosya adı olarak kabul edip, bulunduğunuz dizinde ve alt dizinlerinde bu dosyayı arayacaktır. Eğer iki parametreyle başlatılırsa, birinci parametre aramanın başlayacağı dizin, ikinci parametreyse aranacak dosyanın adı kabul edilecektir. Eğer dosya adı içinde * kullanmak istiyorsanız *'li ifadeyi çift tırnak içine almayı unutmayınız.

Örnekler :

```
% ff aranan.veri.dosyasi
% ff /home/ugur prog.c
% ff ~ file001.dat
% ff "*dat"
% ff /cdrom "openwin*"
```

Yeni yarattığınız **ff** komutunu verdiğinizde, komut programının bulunamadığına ilişkin bir mesaj alıyorsanız, **path** değişkeninizde **home** dizininiz olmayabilir. Çalıştırmak istediğiniz programı oluşturan dosyanın çalışma dizininizde bulunması yetmez. Bir programın çalıştırılabilmesi için

i) ya yeri tam olarak komutta belirtilmelidir (**~/ff** gibi)
ii) ya da program dosyasının bulunduğu dizin, **path** değişkeninde tanımlanmış olmalıdır.

path ile ilgili bir sorun olmamasına rağman 'komut bulunamadı' *(ff : Command not found.)* mesajını alıyorsanız; **chmod** komutuyla, **ff** programının **"çalıştırılabilir"** *(executable)* bir dosya olduğunu belirtmeyi unutmuş olabilirsiniz.

Bir olasılık ta; **ff** komut dosyasını girerken bir hata yapmış olabileceğinizdir.

Arama - Tarama

find komutuyla; dosyaları, adları ve sahipleri gibi özelliklerine göre taramayı öğrendiniz. Peki.... Dosyaların içinde kayıtlı verilere göre aramaları nasıl yapcaksınız? Örneğin, içerdiği kayıtlar arasında **ayfer** sözcüğü geçen dosyaları bulmak istediğinizde hangi komutu kullanmalısınız?

```
% grep [-ilnc] patern dosya(lar)      general purpose regular
                                       expression search program
```

Hemen bir kaç örnek...

İçinde yaklaşık 20,000 satır bulunan **/etc/termcap** dosyasında (terminal karakteristikleri tanıtım dosyası) "wyse50 marka terminallerle ilgili bir tanım var mı?" diye merak ettiğinizde

```
% grep wyse50 /etc/termcap
```

komutunu kullanabilirsiniz. Eğer bu dosyanın içinde **wyse50** sözcüğü geçiyorsa, bu satırlar standart çıktı birimine (ekrana) listelenecektir.

wyse50 sözcüğünün büyük harflerle yazılmış olma olasılığı varsa

```
% grep -i wyse50 /etc/termcap
```

formunu denemelisiniz. (**-i** : *ignore case*; büyük-küçük harf ayrımı yapılmasın)

Bulunan satırların satır numaralarını da görmek isterseniz

```
% grep -ni wyse50 /etc/termcap
```

formunu kullanabilirsiniz. (-n : *numbered*)

Bulunduğunuz dizinde, adı **mektup** ile başlayan dosyalar arasında bir veya birkaç tanesinin içinde **ayfer** sözcüğünün bulunduğunu biliyorsunuz ama hangileri olduğunu hatırlayamıyorsunuz! İşte çözüm :

```
% grep ayfer mektup1 mektup2 mektup3 ...
veya
% grep ayfer mektup*
```

grep komutu, arama işini birden fazla dosya üzerinde yaptığı zaman, kullanıcıya kolaylık olması için; bulunan satırları ekrana listelerken, her satırın başına, satırın bulunduğu dosyanın adını ekler.

Eğer, bulunan satırlar için yalnızca dosya adlarını görmek istiyorsanız,

```
% grep -l ayfer mektup*
```

formunu kullanmalısınız.

Adı **mektup**'la başlayan dosyalarda **ayfer** sözcüğünün kaç defa geçtiğini öğrenmek isterseniz

```
% grep -c ayfer mektup*
```

komutunu kullanılabilirsiniz.

grep komutu (ve onun biraz geliştirilmişleri olan **egrep** ve **fgrep**), UNIX işletim sisteminin en çok kullanılan komutlarındandır. Bu komutun daha yararlı kullanımlarına ilişkin örneklere devam etmeden önce çok önemli bir UNIX kavramından daha söz etmek istiyorum: **PIPE**.

pipe kavramını anlatırken kullanacağım örnekler arasında **grep** komutuyla ilgili olanları dikkatle incelerseniz yukarıda verilen örneklerden daha yararlı kullanımlarını öğrenmiş olacaksınız.

UNIX kullanıcılarının günlük hayatta karşılaşacağı tipik bir sorun ve bu sorunun çözümünden söz etmek istiyorum : Sorun (ya da soru) şu :

home dizinimin altında yer alan bir takım dizinlerde bir takım dosyaların içinde "piper" sözcüğü geçiyor. Bu dosyaların hangileri olduğunu

```
grep piper *
```

komutuyla bulabileceğimi biliyorum; ama home dizinimin altında o kadar çok alt-dizin var ki! Her birine teker teker geçip aynı grep komutunu tekrarlamak istemiyorum. Bu arama işini hem home dizinimde, hem de onun alt dizinlerinde tek komutla yapabilir miyim?

Elbette yapabilirsiniz! UNIX'de, sadece standart UNIX komutları kullanarak, hiç program yazmadan, veri tabanı sistemi bile geliştirebilirsiniz!

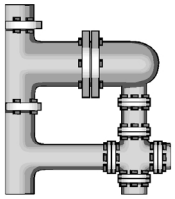
Bu küçük problemin çözümü şu iki komut :

```
cd /home/ayfer
grep piper`find . -print`
```

tırnak işaretlerine dikkat! ASCII kodu desimal 96 olan tırnak işaretidir. Burada ' veya " kullanamazsınız.

Komutun çalışma sistemi aslında basit. Kabuk programınız, komut satırında ` işaretleri arasında bir başka komut görünce önce onu çalıştıracaktır (**find . -print** komutu). Bu programın standart çıktıya gönderdiği listeyi de **grep** komutunun sonuna ekleyecek; **grep** komutunu ondan sonra çalıştıracaktır. Bir başka deyişle, **grep** komutuna, içinde **piper** sözcüğü aranacak dosyaların listesini klavyeden yazmak yerine, bu işi **find** komutuna yaptırmış olacaksınız. Zarif, değil mi?

UNIX PIPE Kavramı



pipe (boru) kavramı, daha önce açıklamış olduğum 'Giriş/Çıkış Yönlendirme' kavramıyla kolayca karıştırılan, bu yüzden dikkatle ele alınması gerek bir kavramdır. Kısaca bir tekrarlamak gerekirse; '**çıkış yönlendirme (>)**', çalıştırılan bir programın, standart çıktı birimine yazacağı satırların bir dosyaya yönlendirme işlemidir. Aynı mantıkla, verilerini standart giriş biriminden okuyan programlar için '**giriş yönlendirme (<)**'; verilerin bir dosyadan okunmasını sağlayan işlemdir.

Piping işlemiyse, gene bir çeşit yönlendirmedir; ancak şu farkla ki, bir programın standart çıktısı, bir başka programa standart girdi olarak yönlendirilir.

Pipe kurmak için, aynı komut satırında en az iki program birden başlatmalı ve bu iki programa ilişkin komutların arasına

|

karakterini yerleştirmeniz gerekir.

Şimdi **grep** komutu ve ***pipe*** kavramının birlikte kullanımına bir kaç örnek vereyim :

```
% grep ayfer mektup* | more
```

Bu komut, daha doğrusu komut ikilisinin anlamı şu :

grep ve **more** programlarını aynı anda başlat.
Adı '**mektup**'la başlayan dosyalar içinde **ayfer** sözcüğünü ara, bulduğun satırları **more** programına gönder, **more** programı bu satırları alsın ve kendi görev tanımı doğrultusunda işlesin. (Yani sayfa sayfa listelesin).

```
% ps -ax | grep in.named
```

ps ve **grep** programlarını aynı anda başlat.. **ps** programının oldukça uzun olabilecek çıktısını **grep** programına girdi olarak gönder.

grep kendisine gönderilen satırlar arasında, içinde i**n.named** sözcüğü geçenleri bulsun ve sadece ilgilendiğimiz bu satırları listelesin. Böylece **grep** programı bir filtre gibi kullanılmış olacaktır.

Şimdide **sıkı** bir *pipe* örneği...

□ tuşunun solundaki tırnak işareti...

```
% echo Sistemde `who | wc -l` kullan□c□ var
```

Bu komut satırında bir kaç kademeli bir işlem istenmektedir. İlk olarak **who** programı çalıştırılacaktır. Aynı anda **wc** programı da çalıştırılacak ve **who** programının çıktısı standart girişindeki satır, kelime ve karakterleri sayan **wc** (-l seçeneği yanlızca satırların sayılmasını sağlıyor) programına gönderilecektir. **wc** programının çıktısıysa (**who** komutunun listelediği satırların sayısı) tırnaklar arasına yerleştirilerek elde edilen; örneğin üç kullanıcı varsa, '**Sistemde 3 kullanıcı var**' dizisi de **echo** programına girdi olarak transfer edilir. **echo** programıysa parametrelerini aynen ekrana gönderir. Bu örnekteki komutu, **home** dizinizdeki **.login** veya **.cshrc** dosyasına eklerseniz, sisteme her **login** edişinizde, sistemde siz dahil, kaç kişinin çalıştığını öğrenmiş olursunuz.

Yazıcı Kullanımı

MS-DOS işletim sistemi ile çalışan kişisel bilgisayarlarda yazıcı kullanımı oldukça kolaydır. Bilgisayarı yalnızca siz kullandığınız için, diğer tüm kaynaklar gibi yazıcı da sadece sizin kullanımınıza tahsis edilmiş durumdadır. Canınız istediği zaman programınıza 'yaz' komutunu verir ve yazıcının başına geçip, çıktılarınızın kağıda aktarılmasını beklersiniz.

UNIX dünyasında durum farklı... Kullanıcılar, bilgisayarın tüm kaynakları gibi yazıcısını da başkalarıyla paylaşmak zorundalar. Yazıcıya gönderilen her döküm, MS-DOS'da olduğu gibi anında basılmaya başlamayabilir; çünkü o anda yazıcı bir başka kullanıcının bir çıktısını döküyor olabilir. Kağıda bir kaç satır bir kullanıcıdan, bir kaç satır da başka kullanıcıdan döküm yapmak pek sağlıklı olmayacağı için, tüm çok kullanıcılı işletim sistemlerinde olduğu gibi, UNIX'te de; yazıcı dökümlerinin sıraya konmasını sağlayan **SPOOLING** ***(Shared Peripheral Operation Online)*** işlemi uygulanmaktadır.

UNIX'de, kullanıcı programlarından gelen '**yazıcıya yolla'** emirleri sanki yerine getirilmişcesine olumlu karşılanır; ancak yazıcıya gönderilmesi istenen bilgiler diskte önceden belirlenmiş bir alana kaydedilir *(**spool** area).* Zaman içinde kullanıcılardan gelen döküm istekleri UNIX **spooler** programı tarafından sıraya konur ve yazıcı boş kaldığında, diskte saklanan dökümler kağıda aktarılmak üzere sırayla yazıcıya veya yazıcılara gönderilir.

Bir başka deyişle, uygulamanız, kağıda bir döküm almanızı gerektiriyorsa ve siz bu doğrultuda yazdırma komutu verdiyseniz; isteğiniz hemen yerine getirilemeyebilir (yazıcı meşgul olabilir veya hazır olmayabilir). Ancak, bu durum size yansıtılmaz ve kağıda dökülmesini istediğiniz her şey, **spooler** tarafında diske kaydedilerek sıraya sokulur ve ilk fırsatta yazıcıya gönderilir.

UNIX altında çalışan bilgisayarlar genellikle büyükçe sistemler olduğundan, birden fazla yazıcıya sahip olabilirler. Özellikle bilgisayar ağlarında bu durumla daha da sık karşılaşılır. Bir döküm almak istediğinizde, yazıcı seçme şansınızın da olabileceğini unutmayınız.

BSD ve SVR4 UNIX türevlerinde yazıcı kullanma komutları oldukça farklı olduğundan, bu iki tip UNIX için ayrı ayrı bölümler hazırladım. Genel kültür açısından her iki bölümü de okumanızı öneririm.

BSD UNIX'de Yazıcı Kullanımı

BSD

Herhangi bir dosyayı yazıcıya göndermek istediğinizde kullanabile-ceğiniz en basit komut formu şudur :

```
% lpr  dosya_adi                                    (line printer)
```

Bu komutu verdiğinizde, **dosya_adı** isimli dosya, adı **lp** olan yazıcının (veya **PRINTER** isimli kabuk değişkeninde belirtilmiş olan isme sahip yazıcının) sırasına sokulur. Sisteminizde adı **lp** olan bir yazıcı bulunmasını sağlamak, sistem yöneticisinin görevidir.

Eğer, dosyanızı, özel bir yazıcıya göndermeniz söz konusuysa kullanacağınız komut

```
% lpr  -Pyazici_adi dosya_adi
```

Bu komutta **P** harfinin büyük P olduğuna ve yazıcı adının bu **P** harfine bitişik olarak yazıldığına dikkatinizi çekerim. (Bazı UNIX'ler **P** harfiyle yazıcı adı arasında boşluk kullanılmasına izin verir.)

```
% lpq  [-Pyazici_adi]                              (line printer queue)
```

Yazıcı için sıra bekleyen işler hakkında bilgi verir. Sıra bekleyen her dökümün bir tanıtma numarası vardır.

```
% lprm  [ nnn [mmm ...] ]                          (line printer remove)
```

Sıra bekleyen dökümler arasında tanıtma numarası **nnn** (ve **mmm** vs) olan işleri iptal eder. **nnn** verilmezse, komutu veren kullanıcıya ait olan ve o sırada dökülmekte olan ya da sıradaki ilk işi iptal edilir.

```
% lprm  kullanici_adi                              (line printer remove)
```

Sıra bekleyen işler arasında sahibi **kullanici_adi** olan dökümleri iptal eder. **nnn** (ve **mmm** vs)

```
% lpstat  [-Pyazici_adi]                           (line printer status)
```

Yazıcının durumunu gösterir. (Hazır olup olmadığını vs.)

```
% lpr  -#n  -Plazer  dosya_adi
```

dosya_adı isimli dosyanın, **lazer** isimli yazıcıdan **n** kopyasının basılmasını sağlar.

```
% lpr  -m  onemli
```

onemli isimli dosyanın, basılmak üzere **lp** isimli yazıcıya gönderilmesini ve basım tamamlandığında, komutu veren kullanıcıya bir mesaj *(mail)* gönderilmesini sağlar.

```
% lpr  -r  onemsiz
```

onemsiz isimli dosyanın, basılmak üzere **lp** isimli yazıcıya gönderilmesini ve dosyanın, ilgili yazıcının sırasına alınmasından hemen sonra diskten silinmesini sağlar.

```
% lpr  dosya1 dosya2 ...
```

Birden fazla dosyanın tek komutla yazıcı sırasına gönderilmesini sağlar.

```
% sort < sirasiz | lpr
```

lpr komutu ve **pipe** kavramının birlikte kullanılışına bir örnek... Bu örnekte, **sirasiz** isimli dosya **sort** programıyla sıraya dizilmekte ve sıralanmış hali doğrudan yazıcıya gönderilmektedir.

Örneklerini verdiğim çeşitli **lpr** seçeneklerini birleştirebileceğinizi ayrıca belirtmeme sanırım gerek yok. Örneğin;

```
% lpr  -rmPepson onemsiz
```

Kullanıcısı olduğunuz bilgisayar sistemine bağlı olan yazıcıların özelliklerini ve isimlerini sistem yöneticisinden öğrenebilirsiniz.

SVR4 UNIX'de Yazıcı Kullanımı

Herhangi bir dosyayı yazıcıya göndermek istediğinizde kullanabileceğiniz en basit komut formu şudur :

```
% lp  dosya_adi                                    (line printer)
```

Bu komutu verdiğinizde, **dosya_adı** isimli dosya, adı **lp** olan yazıcının (veya **PRINTER** isimli kabuk değişkeninde belirtilmiş olan isme sahip yazıcının) sırasına sokulur. Sisteminizde adı **lp** olan bir yazıcı bulunmasını sağlamak, sistem yöneticisinin görevidir.

Eğer, dosyanızı, özel bir yazıcıya göndermeniz söz konusuysa kullanacağınız komut

```
% lp  -dyazici_adi dosya_adi
```

Bu komutta **d** harfinin küçük d olduğuna ve yazıcı adının bu **d** harfine bitişik olarak yazıldığına dikkatinizi çekerim.

```
% lpstat  [-a]                                    line printer status
```

Yazıcının durumunu gösterir. (Hazır olup olmadığını vs.) **-a** seçeneği tüm yazıcıların durumunu gösterir. Durum raporlarında, yazıcılar için sıra bekleyen işler ve tanıtım numaraları da listelenir.

```
% cancel nnn [mmm ...]
```

Sıra bekleyen dökümler arasında tanıtma numarası **nnn** (ve **mmm** vs) olan işleri iptal eder.

```
% cancel -u ugur
```

Sıra bekleyen dökümler arasında **ugur** isimli kullanıcıya ait olan döküm işlerini iptal eder.

```
% lp -nk -dlazer  dosya_adi
```

dosya_adı isimli dosyanın, **lazer** isimli yazıcıdan **k** kopyasının basılmasını sağlar.

```
% lp  -m   onemli
```

onemli isimli dosyanın, basılmak üzere **lp** isimli yazıcıya gönderilmesini ve basım tamamlandığında, komutu veren kullanıcıya bir mesaj *(mail)* gönderilmesini sağlar.

```
% lp  dosya1 dosya2 ...
```

Birden fazla dosyanın tek komutla yazıcı sırasına gönderilmesini sağlar.

```
% sort < sirasiz | lp
```

lp komutu ve **pipe** kavramının birlikte kullanılışına bir örnek... Bu örnekte, **sirasiz** isimli dosya **sort** programıyla sıraya dizilmekte ve sıralanmış hali doğrudan yazıcıya gönderilmektedir.

Örneklerini verdiğim çeşitli **lp** seçeneklerini birleştirebileceğinizi ayrıca belirtmeme sanırım gerek yok. Örneğin;

```
% lp  -mdepson onemsiz
```

Kullanıcısı olduğunuz bilgisayar sistemine bağlı olan yazıcıların özelliklerini ve isimlerini sistem yöneticisinden öğrenebilirsiniz.

SVR4 UNIX'lerde yazıcı yönetimi ile ilgili olan bir kaç komut daha vardır. Bu komutların görev ve yetenekleri kitabın sınırlarını çok aştığı için; sadece meraklı kullanıcılar için bu komutların isimlerini verip geçeceğim. Bu komutlar hakkında daha fazla bilgi almak için **man** komutunu kullanabilir veya UNIX dökumantas-yonuna başvurabilirsiniz. Yazıcı yönetimine ilişkin diğer SVR4 UNIX komutları :

```
accept, lpadmin, disable, enable,
   lpmove, pr, reject, lpsched
```

Kabuklar - C Shell ve Shell

Komut Satırının Yorumlanması ve Parametreler

UNIX İşletim sistemi, kullanıcıların verdikleri komutları çözümlemek ve bu komutları yerine getirecek programları başlatmak için kabuk *(shell)* programlarını kullanır. Bir başka deyişle, kabuk programları, kullanıcılarla bilgisayar arasındaki yazılım arabirimidir. Aslında, bu tip komut yorumlayıcıları *(command interpreter),* tüm işletim sistemlerinde kullanılmaktadır; örneğin MS-DOS işletim sisteminde bu görevi COMMAND.COM üstlenmiş durumdadır.

UNIX 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 kullanılabilirler. Daha fazla detaya girmeden, genel olarak bir kabuk programının ne işler yaptığını bir örnekle açıklamaya çalışacağım. Bu örneğimizle ilgili bir kaç tane de varsayımımız olacak; şöyleki :

- Kullanıcı C-Shell kabuk programını kullanıyor olsun,
- Kullanıcının adı **ayfer** ve komutu verdiği anda kendi çalışma
- dizini **/home/ayfer** olsun,

Başarılı bir **login**'den sonra, UNIX, komut beklediğini,

```
abc:/home/ayfer % _    veya sadece    % _
```

hazır işaretiyle *(prompt)* belli edecektir. (Eğer C-shell yerine **sh** kabuk programı kullanılıyor olsaydı, **%** işareti yerine **$** işareti görünüyordü).

Kullanıcı klavyesinden; örneğin;

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

komutunu verdiğinde, kabuk programı, **cp** harflerini kullanıcının çalıştırmak istediği programın adı olarak; **eski-dosya** ve **yeni-dosya** kelimeleriniyse bu **cp** programının iki parametresi olarak kabul edecektir.

cp	eski-dosya	yeni-dosya
(Komut)	(1. parametre)	(2. parametre)

Bir sonraki iş, kullanıcının çalıştırmak istediği bu **cp** programının saklandığı disk dosyasını bulmak olacaktır. Bu arama işinin temelinde, kullanıcımız için

tanımlanmış olan **PATH** ve/veya **path** kabuk değişkeninin o andaki değeri yatmaktadır. Bu değişkenlerin değerleri

```
PATH = /bin:/usr/bin:/usr/local/bin:~ayfer/bin:.
     veya
path= ( /bin /usr/bin /usr/local/bin ~ayfer/bin .)
```

benzeri bir karakter dizisi olacaktır ve bu değerler, kullanıcının **home** dizininde yer alan **.cshrc** ve/veya **.login** dosyalarında tanımlanmış olmalıdır. Şimdilik, bu dosyaların, sizin için, sistem yönetici tarafından hazırlanmış olduğunu kabul edebilirsiniz.

csh programı, "**:**" işaretleriyle (ya da boşluk karakterleriyle) birbirlerinden ayrılmış olan dizinlerde; **cp** isimli bir dosya arayacaktır. Arama, dizin isimlerinin veriliş sırasına göre yapılacaktır. Örneğimize göre, **csh** programı, **cp** isimli dosyayı önce **/bin** dizininde; orada bulamazsa **/usr/bin** dizininde; orada da bulamazsa **/usr/local/bin**; olmazsa **ayfer** adlı kullanıcının **home** dizininin altındaki **bin** dizininde (**~ayfer/bin**); o da olmazsa o andaki çalışma dizininde (**.**) arayacaktır. Söz konusu dosyayı bu dizinlerden hiç birinde bulamazsa

```
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 edilir; **ayfer**'in bu programı çalıştırmaya yetkisi varsa (*execute* yetkisi) **cp** programı kabuk tarafından belleğe yüklenir ve çalıştırılır. Komut satırında verilen parametrelerse, gerekirse çözümlenip, **cp** programına aktarılır.

Artık kontrol, **cp** programına geçmiştir. Bu programın mantığına göre son parametre, kopyalamanın yapılacağı dosya ya da dizin adını, önceki parametrelerse buraya kopyalanacak dosyaların isimleri olmalıdır. Bir başka deyişle, **cp** komutunun en az iki parametresi bulunmalıdır. kabuk programı bu detayları bilemeyeceği için, bu tip mantık kontrolleri komut programı tarafından yapılmalıdır. Parametrelerin doğru sırada ve sayıda verilip verilmediğini her program kendisi kontrol eder ve gerekirse uygun hata veya uyarı mesajları üreterek, kullanıcıyı uyarır.

Şimdi ortalığı biraz karıştıralım....

Kullanıcımız

```
cp  *  /disk2/home2/ayfer
```

komutunu vermiş olsun. Bu komutla kullanıcının yapmak istediği iş, çalışma dizinindeki tüm dosyaları (*****), **/disk2/home2/ayfer** dizinine kopyalamak... Bu komutu gören **csh**, komut adı olan **cp** sözcüğünü bulduktan sonra, bu komutun parametrelerini bulup çıkarmaya çalışacaktır. Komut satırını tararken

(parsing) ***** karakterine rastlayınca, **csh**, "tüm dosyalar" anlamına gelen ***** yerine, çalışma dizininde yer alan dosyaların isimlerini yanyana gelecek şekilde yerleştirecektir.

Yani, komut satırı

```
cp abc dosya1 dosya2 xyz x123 muhasebe.dat /disk2/home/ayfer
```

şekline dönüştürülecektir (çalışma dizininde sadece **abc**, **dosya1**, **dosya2**, **xyz**, **x123** ve **muhasebe.dat** dosyalarının yer aldığı varsayımıyla). Bu dönüşümü ekranda gözleyemezsiniz; ancak bu tip dönüşümlerin olduğunu bilmeniz ve komutları verirken bu dönüşümleri dikkate almanız çok önemlidir.

Bazı komutların doğru çalışması için, komut satırlarının, kabuk programları tarafından dönüştürülmeden komut programlarına aktarılması gerekmektedir. Bu gerekliliği açıklayan en iyi örnek **find** komutudur.

Hatırlarsanız, **find** komutuna ilişkin verdiğim örneklerden biri, adı ***.tmp** kalıbına uyan ve büyüklüğü 100 bloktan fazla (51200 byte'dan fazla) olan dosyaları bulup listelemeye yönelikti.

```
find /home -name "*.tmp" -a -size +100 -print
```

Bu örnekte **"*.tmp"** yazarken kullanılan **"** işaretleri **çok çok önemlidir**. Kabuk programı, tırnak içinde yer alan komut bölümlerini çözümlemeye çalışmayacaktır. Komut satırında tırnak içinde yer alan bölümler, hiç bir değişikliğe uğramadan, ilgili programa parametre olarak iletilecektir. Şimdi, yukarıdaki **find** örneğindeki komutu tırnak işaretlerini kullanmadan yazdığımızı farzedelim....

```
find /home -name *.tmp -a -size +100 -print
```

hatalı

Bu komutu gören kabuk programı, **find** sözcüğünü program adı olarak değerlendirip, bu programın parametrelerini saptamak amacıyla satırı taramaya devam edecektir. **/home** birinci; **-name** ise ikinci parametre olarak çözümlenecektir. Buraya kadar sorun yok.... Ancak ***.tmp** kalıbına rastlandığında, çalışma dizininde yer alan ve adı bu kalıba uyan dosyaların isimleri komut satırına üçüncü, dördüncü, beşinci vs parametre olarak yerleştirilecektir (tabii çalışma dizininde adı bu kalıba uyan dosyalar varsa). Diyelimki, bu komutu verdiğimizde, çalışma dizinimizde şu dosyalar bulunmaktaydı :

```
a           dosya1      dosya2
a.tmp       dosya1.tmp  dosya3
```

kabuk programı tarafından çözümlenen ve dönüştürülen komut satırı

```
find /home -name a.tmp dosya1.tmp -a -size +100 -print
```

olacaktır.

BU ŞEKİLDE ÇÖZÜMLENMİŞ KOMUT BİRKAÇ NEDENLE HATALIDIR.

Birincisi; **find** komutunun mantığına göre arama sadece adı **a.tmp** olan dosyalar için yapılacaktır; oysa biz adı ***.tmp** kalıbına uyan tüm dosyaları aramak istiyoruz.

İkincisi; dosya1.tmp parametresi tanımlayıcı işaretsiz (-name, -size gibi) kalmıştır. (Nitekim, komutu verdiğinizde **find : missing conjunction** mesajı alırsınız).

Bu hataların olmaması için, kabuk programının, komut satırımızla oynamamasını ve ***.tmp** parametresini, **find** programına AYNEN göndermesini sağlamamız gerekmektedir. İşte, " tırnak karakterleri burada işe yaramaktadır.

KABUK PROGRAMLARI, " TIRNAK KARAKTERLERİ ARASINDA YER ALAN KOMUT PARÇALARINI ÇÖZÜMLEMEYE ÇALIŞMAZ VE PROGRAMA OLDUĐU GİBİ İLETİR:

Komutumuzu

```
find /home -name "*.tmp" -a -size +100 -print
```

doğru

olarak verince, **find** komutunun birinci parametresi **/home**, ikinci parametresi **-name**, üçüncüsü ***.tmp**, dördüncüsü **-a**, beşincisi **-size**, vs. olarak kabul edilecek ve **find** programı bu parametre yapısıyla çalıştırılacaktı. Yani, ***.tmp** kalıbı kabuk tarafından değil, **find** programı tarafından yorumlanacak ve komut istediğimiz şekilde çalışacaktır.

Eğer, kabuk programının irdelemeden komuta aktarmasını istediğiniz özel karakter tek bir karakterden oluşuyorsa, o karakteri tırnak içine almak yerine, önüne bir **** *(back slash)* yerleştirebilirsiniz. Bir başka deyişle

`"*.tmp"` ile `\*.tmp` ve
`\!` ile `"!"` eşdeğerdir.

Bu arada, kabuk tarafından çalıştırılan programların **sıfırıncı parametrelerinin** de bulunduğunu söylemeden geçemeyeceğim. Bir program çalıştırıldığında, sıfırıncı parametresi, programın kendi adıdır. Böylece, her program, hangi isimle kullanıldığını bilebilmektedir. Bu özelliğe tipik örnek **compress** ve **uncompress** komutlarıdır. Bu iki komut aslında tek bir program dosyasıdır. **compress** isimli dosya gerçekten bu isimle diskte yer alırken, **uncompress** sadece bu dosyaya bir bağlantıdır *(link)*.

```
% which compress
/usr/ucb/compress

% ls -lF /usr/ucb/compress /usr/ucb/uncompress
-rwxr-xr-x  1 root  23783 ... compress*
```

```
lrwxr-xr-x  1 root  23783 ... uncompress --> ./compress
```

which komutu da nereden çıktı diyorsunuz... Çok önemli bir komut değil... Parametresi olarak belirtilen komut verilmiş olsaydı, hangi dosyanın çalıştırılacağını bildirir. Bir diğer deyişle, parametresi olan komutu **PATH** ve/veya **path** değişkenlerine göre disk(ler)de arar ve ilk bulduğunun (bulursa) yerini bildirir.

compress program dosyasını **compress** adıyla kullanırsanız, dosya sıkıştırma işlemi; **uncompress** adıyla çalıştırırsanız, daha önce sıkıştırılmış olan bir dosyayı açma işlemi yapılmasını sağlarsınız.

Kabuk Değişkenleri

Kullandığınız kabuk programı içinde çeşitli değişkenler tanımlamanız mümkündür; hatta bazı standart değişkenler zaten tanımlıdır. Kabuk değişkenleri arasında en önemlileri

Bourne Shell (sh)	C Shell (csh)	Görevi
PATH	path	Bir komut verildiğinde, komut programını oluşturan dosyanın aranacağı dizinler listesini belirleyen değişken.
HOME	HOME	Kullanıcının **home** dizinin adını içeren değişken. **login** ettiğinizde kabuk programı tarafından otomatik olarak yaratılır.
MAIL	mail	Size elektronik posta (e-mail) gelip gelmediğini anlamak için kontrol edilecek dosyaların listesi.
PS1 Tanımlanmazsa **$** kabul edilir	prompt Tanımlanmazsa **%** kabul edilir	Sistem hazır işaretini tanımlayan değişken.

TERM	TERM	Kullandığınız terminalin tipini belirleyen değişkendir. Eğer **vi** editörünü kullanmak istediğinizde ekranınıza garip işaretler yanısıra satırlar da garip bir düzensizlik içinde çıkıyorsa, büyük olasılıkla **TERM** değişkeniniz hatalı bir değere sahiptir ya da hiç tanımlı değildir. Bu değişkenin kullanımı, **/etc/termcap** dosyasındaki binlerce değişik terminal tipinin tanımlarıyla yakından ilgilidir. **TERM** değişkeninize vermeniz gereken değer için sistem yöneticinize danışınız.
	history	Sadece **csh** kabuğunda anlamlıdır. Bu değişkenin değeri, klavyeden gireceğiniz komutlardan son kaç tanesinin saklanacağını ve **!** ile birlikte tekrar kullanılabileceğini belirtir. Tipik değeri 30-50 arasındadır.

Örnekler	
Bourne Shell (sh)	**C Shell (csh)**

`PATH=/bin:/usr/local/bin:.`	`set path=( /bin /usr/local/bin . )`
	veya
	`setenv PATH /bin:/usr/local/bin:.`

`MAIL=/usr/mail/ayfer`	`setenv mail /usr/mail/ayfer`
`PS1="ayfer@abc $ "`	`set prompt="ayfer@abc % "`
`TERM=vt100`	`setenv TERM vt100`
	`setenv history 100`

Bazı UNIX'lerdeki **sh** uyarlamalarında, bir kabuk değişkenine değer verdikten sonra değişkeni **export** komutu ile geçerli kılmanız gerekebilir. Şöyle ki :

```
PATH=/bin:/usr/local/bin:.      komutundan hemen sonra
export PATH                     komutunu vermeniz gerekebilir.
```

Herhangi bir anda, tanımlı olan kabuk değişkenlerini ve/veya değerlerinin ne olduğunu merak ederseniz

```
% set                           sh ve csh için
% env                           sh için
% setenv                        csh için
```

komutlarını kullanabilirsiniz

Kabuk değişkenleri, standart isimli bir takım değişkenlerle sınırlı değildir. Kullandığınız uygulama programları, çalışma ortamını tanımlamak için bir takım değişkenlerin tanımlanmasını ve özel değerler verilmesini gerektirebilir. Örneğin X Windows uygulamaları, kullanılacak ekranın bağlı bulunduğu bilgisayarın adının **DISPLAY** isimli bir değişkende tanıtılmasını gerektirecektir.

C Shell'e Özgü Özellikler

UNIX dünyasında kabuk programı olarak **csh** kullanımı gittikçe yaygınlaşmaktadır. Bu nedenle kitabın bundan sonraki kısımlarında kullanıcıların kabuk programı olarak C Shell kullandıklarını varsayacağım. Eğer bu kitapta bundan sonra anlatılacak konulardaki örnekleri denemek istiyorsanız, **csh** kabuk programını kullanıyor olmalısınız. Hangi kabuk programını kullandığınızı bilmiyorsanız, sisteme login ettiğinizde sizin için başlatılacak olan kabuk programınızı değiştirmeniz gerekiyorsa sistem yöneticinizden yardım isteyiniz. Kullandığınız kabuk **tcsh** ise bir değişiklik yapmanız gerekmeyecektir.

csh'in bazı önemli özelliklerini örneklerle açıklamak istiyorum :

Hatalı Komutları Düzeltme

Diyelimki uzun bir UNIX komutunu yanlış yazdınız....

```
cp /home/hakman/.Xdesksetdefualts /home/ayfer
```

*(**.Xdesksetdefaults** olmalıydı...)*

ve doğal olarak **.Xsetdefualts** diye bir dosya bulunamadığına dair bir hata mesajı aldınız. Eğer **csh** kullanıyorsanız, bu karışık satırı baştan bir kez daha yazmak yerine

```
% ^fual^faul^
```

yazıp ENTER tuşuna basmanız yeterli olacaktır. ("Bir önceki komuttaki **fual** karakter dizisini **faul** karakter dizisi ile değiştir ve komutu tekrarla" anlamında...)

Son Komutu Tekrarlama

Diyelimki MS-DOS işletim sisteminden alışkanlıkla

```
% cp /home/hakman/.Xdesksetdefaults
```

yazdınız. UNIX kurallarına göre kopyalamanın nereye yapılacağını da belirtmiş olmanız gerekirdi. **csh** kullanıyorsanız, böyle bir durumda, tüm komutu tekrarlamak yerine

```
% !!  /home/ayfer
```

yazmanız yeterli olacaktır. Böylece; bir önceki komutunuz aynen tekrarlanacaktır; ancak sonuna **/home/ayfer** eklenmiş olarak...

Eski Bir Komutu Tekrarlama

Eğer **history** isimli kabuk değişkeniniz tanımlıysa, bu değişkenin değeri kadar sayıda UNIX komutu kabuk tarafından bir ara bellekte saklanacaktır. Her hangi bir anda, eski komutlarınızı

```
% history
```

komutuyla listeyebilirsiniz.

```
% history
.....
7  23:12  ls -l
8  23:13  cat  /etc/printcap
9  23:13  cp /usr/bin/....   .....
```

Bu eski komutlardan birini tekrar çalıştırmak isterseniz, bir ünlem işaretinin ardından o komutun listedeki sıra numarasını girmeniz yeterli olacaktır.

```
% !8                                  8 numaralı komutu tekrarlama
```

İlk Birkaç Harfini Hatırladığınız Eski Bir Komutu Tekrarlama

Her hangi bir anda, ilk harfini (ya da birkaç harfini) hatırladığınız eski bir komutu tekrarlamak isterseniz

`% !c`	***c** harfiyle başlayan son komutu tekrarla*
`% !ca`	***ca** harfleriyle başlayan son komutu tekrarla*

Kendi Gereksinimlerinize Göre Özel Komut Yaratma

UNIX kullanıcılarının çok sık tekrarladıkları bazı uzun komutları, daha kısa ve kolay yazılan komutlarla değiştirmeleri mümkündür. Bu iş **csh**'in **alias** komutu ile yapılır.

Örneğin, **history** komutunu her seferinde uzun uzun yazmaktansa,

`% alias h history`	*artık **history** yerine **h** kullanabilirsiniz*

ls listelerinde isimlerin çalıştırılabilir program olup olmadığını ***** işaretiyle, dizinlerinse **/** işaretiyle belirlenmesi için kullanılan **-F** seçeneğinin standart hale getirilmesi için :

`% alias ls "ls -F"`	*yeni tanımda birden fazla sözcük olduğu için tırnak kullanmak gerekir*

Eğer **more** komutunu sık sık hatalı yazıyorsanız :

```
% alias mroe more
```

MS-DOS alışkanlıklarınızdan vaz geçemiyorsanız :

```
% alias dir "ls -F"
% alias copy "cp -i"
% alias del "rm -i"
% alias ren mv
% alias edit vi
```

Herhangi bir anda geçerli olan **alias**'ları listelemek için

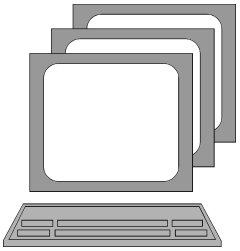
```
% alias
```

Hazır işaretinizin her zaman çalışma dizininizi göstermesi için

```
% alias cd 'cd \!*;set prompt="`hostname`:$cwd> "'
```

biraz çetrefilli ama çalışır... Bana güvenin..

Programları Arka Planda Çalıştırma



Diyelim ki, çok büyük bir disk dosyasındaki (söz gelimi 42 Mbyte) müşteri kayıtlarını alfabetik sıraya dizmek istiyorsunuz. Bu iş için kullandığınız bilgisayar sisteminde yarım saat süreceğini varsayalım. Eğer tek iş düzeninde çalışan bir işletim sistemi kullanıyor olsaydınız (MS-DOS gibi), sıralama komutunu verdikten sonra (**sort**) yemeğe çıkabilir veya köpeğinizi dolaştırmaya götürebilirdiniz; çünkü sıralama bitinceye kadar bilgisayarınızdan bir başka amaçla yararlanmanız söz konusu olamazdı. Oysa, UNIX 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 eklemekten ibarettir.

`% sort musteri-dosyasi &`	***&,*** *işi arka planda yürütmek istediğinizi belirtiyor*

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

Bazı programlar, kulanıcıdan 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 teybe yedekleme yapmak için kullanılan **tar** komutunu

```
% tar -cvf /dev/rst1 /home/ayfer &
```

şeklinde verirseniz (bu komutla ilgili detaylı bilgiyi daha ileride vereceğim; şimdilik komutun ne yaptığı ve parametrelerinin ne olduğu üzerinde durmayınız), program arka planda teybe yedekleme yapacaktır, ama bir

yandan da kopyalamayı tamamladığı bütün dosyaların isimlerini ekrana listeleyecektir. Böyle her saniye yeni bir satır gelen ekranda başka bir iş yapmak pek kolay olmayacaktır.

Ancak aynı komutu

```
% tar -cvf /dev/rst1 /home/ayfer > tarmesajlari &
```

şeklinde verirseniz ekrana gelmesi gereken tüm mesajlar, çalışma dizininizde **tarmesajlari** isimli bir dosyaya yönlendirilmiş olur. İş bittikten sonra **tarmesajlari** dosyasına bakarak teybe kopyalama işinin başarıyla bitip bitmediğini ve kopyalanan dosyaların listesini görebilirsiniz.

Ön Planda Çalışan Programları Arka Plana Atma

Sadece C Shell'de geçerlidir

Bazı durumlarda, başlattığınız bir programın ne kadar süreyle çalışacağını önceden kestiremezsiniz. İşin uzun süreceğini ve sessiz çalışan bir iş olduğunu sonradan farkedersiniz; 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 çekme komutunu

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

şeklinde verdiğinizi ve programı ön planda çalıştırdığınızı varsayalım. Başlattıktan bir kaç saniye (ya da birkaç dakika) sonra işin uzun süreceğini farkettiniz ve **''Tüh! Keşke arka planda başlatsaydım!!''** dediniz. Eğer kabuk programı olarak **csh** kullanıyorsanız sorun değil...

Klavyenizden

```
^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 alındığını göstermektedir; ancak, buradaki **durdurma** kelimesi, işinizin tamamlanmadan kesildiği anlamında kullanılmamaktadır. 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 işniz arka planda çalışmaya devam edecektir. Ancak, o anda ekranda bir de

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

mesajı görünecektir. Bu mesajın kısaca anlamı şudur :

Programınız (komutunuz) arka planda çalışır duruma alındı...
Bu şekilde arka plana atılan işler arasında sıra numarası 1 oldu.

Bu işi tekrar ön plana almak isterseniz

```
% fg  %1                                                Foreground
```

komutunu verebilirsiniz. (Eğer birden fazla arka plana atılmış işiniz varsa, **%** işaretinden sonra o işin numarasını yazmayı unutmamalısınız.)

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.

Bazan, arka planda başlattığınız ya da sonradan arka plana attığınız işlerin hesabını şaşırabilirsiniz. Böyle bir durumda

```
% jobs
```

komutunu verirseniz, arka plana atılmış işlerin bir listesini alırsınız.

Benzeri bir listeyi

```
% ps                                                    process status
```

komutuyla da alırsınız. Ancak **ps** komutunun görevleri biraz daha farklı olabilmektedir. **ps** komutu, parametresiz olarak verildiğinde, kullanıcı olarak sizinle ilgili olarak başlatılmış olan işlerin listesini verir. UNIX işletim sisteminde, siz tek bir iş yaparken (hatta hiç program çalıştırmazken bile) sizinle ilgili birkaç iş, UNIX **kabuk** programı tarafından çalıştırılmaktadır (bu arada kabuk programının kendisi de çalışmaya devam etmektedir.) Hele X Windows, Motif, OpenWindows gibi grafik kullanıcı arabirimleri *(GUI : Graphical User Interface)* kullanıyorsanız; sizinle ilgili olarak onlarca iş başlatılmış olduğunu göreceksiniz. **jobs** komutu sizin tarafınızdan arka plana atılmış işleri; **ps** komutuysa, daha geniş kapsamlı olarak sistemdeki işleri ve bu işlerle ilgili çalışma istatistiklerini listeler.

UNIX'de SÜREÇ (process) Kavramı - Tekrar Bir Göz Atış

Süreçler, UNIX işletim sisteminde çok iyi anlaşılması gereken bir kavramdır. Bu nedenle, bu konuyu bir kez daha tekrarlamak istiyorum. Okuyucunun eski bir bölümü tekrar okumasını istemektense, burada tekrarlamanın daha sempatik ve yararlı olduğunu düşünüyorum. Üstelik, okuyucunun UNIX'e daha yatkın bir duruma geldiğini dikkate alarak biraz daha ayrıntılı olarak anlatma ve örnekleme olanağı bulmuş olacağım. Tekrarlamaya gerek görmeyen okuyucular bu bölümü atlayabilirler.

UNIX, tasarımından kaynaklanan nedenlerle, çeşitli problemlerin çözümü için yazılan programların, problemi parçalayıp, her bir parçanın ayrı bir (ya da birkaç) programdan oluşacak şekilde yazılmasına olanak sağlamaktadır . Pek açık olmadı, farkındayım. İsterseniz örneklerle biraz daha açmaya çalışayım.

UNIX'i yazan insanların bakış açısıyla düşünelim... İşletim sistemi aynı anda şu temel işlerle ilgilenmek zorunda : (bu temel işler herhangi bir sıraya göre verilmemiştir)

1) Kullanıcı programlarının veya bu programları oluşturan süreçlerin bir seferinde ve kesintisiz olarak 200 milisaniyeden fazla MİB kullanmalarını önlemek için işletim sisteminin bir modülü sürekli olarak zamanı izlemeli.

2) Hangi kullanıcının hangi terminalden ne zaman sisteme gireceği belli olmadığından, işletim sisteminin bir modülü, devamlı olarak terminallerin bağlandığı arabirimleri gözlemeli, sisteme **login** etmek isteyen kimse varsa, ona hizmet edecek programları başlatmalı.

3) Hangi kullanıcının ne zaman hangi yazıcıya bir şeyler göndereceği bilinemediğinden ve ayrıca yazıcıların herhangi bir anda, ne durumda olacakları kestirilemeyeceğinden dolayı bir (ya da birkaç) program sürekli olarak yazıcılarla ilgilenmeli.

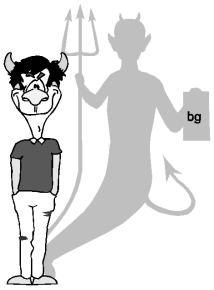
4) Elektronik posta *(mail)* keza...

5) Hele hele bilgisayar bir bilgisayar ağına bağlıysa durum daha da yürekler acısı... İzlenmesi gereken çevre birimleri yetmezmiş gibi bir de sağdan soldan gelen erişim isteklerine yanıt vermek gereklidir.

6) vs. vs.

UNIX işletim sistemini tasarlayanlar, bu problemlerin çözümüne doğru önemli bir kavram geliştirmişler : **SÜREÇ** (*PROCESS*).

Bu tasarıma göre, yapılması gerek tüm işler için ayrı ve bağımsız çalışabilen programlar yazılır, bu bağımsız programlar gerektiğinde birbirleriyle mesaj alışverişinde bulunabilirler ve sıradan kullanıcı programları gibi bilgisayarların MİB, bellek gibi kaynaklarını paylaşırlar.



Bazı süreçlere **daemon** (sözlük anlamıyla : kötü ruh, iblis, iblisin uşağı) adı verilir. Neden bu adın seçildiğini bilmiyorum ama bu tip süreçlerin ortak özellikleri şunlar:

- Arka planda tamamen sessizce çalışırlar,
- Hata mesajları olsa bile bunu genellikle ekrana değil, kendilerine ait bir disk dosyasına yazarlar *(Log File),*
- Klavyeden müdahale gerektirmezler,
- İsimleri genellikle **d** harfiyle biter.

ngi bir anda sistemde çalışmakta olan süreçlerin listesini almak için **ps** komutu kullanılır. BSD veya System V UNIX için küçük farklar gösteren bu komutun en çok kullanılan kalıpları ve bu kalıplar için örnekleri izleyen sayfalarda bulacaksınız.

BSD **BERKELEY UNIX (BSD)**

`ps`	*Kullanıcıyla ilgili süreçleri listeler*
`ps -a`	*Tüm kullanıcılarla ilgili süreçleri listeler*
`ps -ax`	*Tüm kullanıcılar ve sistemle ilgili süreçleri listeler*
`ps -axl`	*l (le harfi) seçeneği ayrıntılı liste verilmesini sağlar*

SVR4 **AT&T UNIX (System V)**

`ps`	*Kullanıcıyla ilgili süreçleri listeler*
`ps -a`	*Tüm kullanıcılarla ilgili süreçleri listeler*
`ps -ae`	*Tüm kullanıcılar ve sistemle ilgili süreçleri listeler*
`ps -ael`	*l (le harfi) seçeneği ayrıntılı liste verilmesini sağlar*

BSD UNIX'de ps komutu kullanımına bir kaç örnek....

```
abc:/home/ayfer> ps
  PID TT STAT  TIME COMMAND
29011 a  R     0:00 ps
abc:/home/ayfer>
```

```
abc:/home/ayfer> ps -x
  PID TT STAT  TIME COMMAND
28731 a  S     0:00 -tcsh (tcsh)
29010 a  R     0:00 ps -x
```

```
abc:/home/ayfer> ps -axl
       F UID   PID  PPID CP PRI NI  SZ  RSS WCHAN  STAT  TT   TIME COMMAND
   80003   0     0     0  0 -25  0   0    0 runout   D    ?   1:20 swapper
20088000   0     1     0  0   5  0  52    0 child    IW   ?   0:08 /sbin/init -
   80003   0     2     0  0 -24  0   0    0 child    D    ?   0:04 pagedaemon
   88000   0    58     1  0   1  0  68    0 select   IW   ?   0:58 portmap
   88000   0    63     1  0   1  0 224    0 select   IW   ?  14:39 ypserv
   88000   3    65     1  0   1  0  36    0 select   IW   ?    0:01 ypbind
   88000   0    67     1  1   1  0  40    0 select   IW   ?   0:00 rpc.ypupdate
   88000   0    69     1  0   1  0  40    0 select   IW   ?   0:00 keyserv
   88001   0   114     1  0   1  0  40   64 select   I    ?   1:09 in.routed
   88000   0   117     1  0   1  0 324    0 select   IW   ?   2:02 in.named
   88001   0   120     1  0   1  0  16    0 nfs_dnlc I    ?   0:00  (biod)
   88000   0   134     1  0   1  0  60    0 select   IW   ?   0:16 syslogd
   88000   0   148     1  0   1  0 108    0 select   IW   ?   1:09 rpc.mountd -
   88000   0   153     1  1   1  0  52    0 select   IW   ?   0:00 rpc.statd
   88001   0   154   149  0   1  0  28    0 socket   I    ?  23:57  (nfsd)
   88000   0   161     1  0   1  0  84    0 select   IW   ?   0:00 rpc.lockd
   88000   0   167     1  0   1  0  80    0 select   IW   ?   0:06 /usr/etc/rpc
   80201   0   182     1  0  15  0  12    4 kernelma S    ? 109:22 update
  488000   0   185     1  0   1  0  56    0 Heapbase IW   ?   0:00 cron
   88000   0   191     1  0   1  0  48    0 select   IW   ?   0:47 inetd
   88000   0   194     1  0   1  0  52    0 select   IW   ?   0:00 /usr/lib/lpd
20488020 560  4969     1  0   1  0 132    0 socket   IW   ?   0:04 ncftp ramiga
   88401   0 24725   167  0  25  0   0    0           Z    ?   0:00 <defunct>
20088000   0 27454     1  0   3  0  40    0 Heapbase IW   co  0:00 - cons8 cons
204882018700 28731     1  0  15  0 216  700 kernelma S    a   0:00 -tcsh (tcsh)
200000018700 29007 28731 24  31  0 216  468           R    a   0:00 ps -axl
abc:/home/ayfer>
```

```
abc:/home/ayfer> ps -ax
  PID TT STAT  TIME COMMAND
    0 ?  D     1:20 swapper
    1 ?  IW    0:08 /sbin/init -
    2 ?  D     0:04 pagedaemon
   58 ?  IW    0:58 portmap
   63 ?  IW   14:39 ypserv
   67 ?  IW    0:00 rpc.ypupdated
   69 ?  IW    0:00 keyserv
  114 ?  I     1:09 in.routed
  117 ?  IW    2:02 in.named
  120 ?  I     0:00  (biod)
  134 ?  IW    0:16 syslogd
  142 ?  IW    0:05 /usr/lib/sendmail -bd -q1h
  148 ?  IW    1:09 rpc.mountd -n
  149 ?  I    24:20  (nfsd)
  ...
```

Meraklı olan okuyucular için yukarıdaki örneklerde adı geçen bazı süreçlerin açıklamalarını yapmak istiyorum. Ancak bunların sadece birer örnek olduğunu, gerçek süreç listelerinin daha uzun ve/veya daha farklı olacağını unutmamalısınız.

Süreç	Görevi
`lpd`	*Line printer daemon.* Yazıcılarla ilgilenir. Yazıcı(lar)dan dökülmek üzere gönderilen bilgilerin sıraya konmasından ve yazıcıların yönetiminden sorumludur. **kill** komutunu kullanarak bu süreci öldürürseniz artık kimse yazıcılardan döküm alamaz.
`inetd`	*Internetworking daemon.* Bilgisayar ağı üzerinden gelip giden servis istekleriyle ilgilenir.
`cron`	Günün, haftanın, ayın belirli zamanlarında çalıştırılması gereken programları izler; zamanı gelen programı başlatır.
`update`	Disk tampon bellek alanlarının belirli aralıklarla disklere kaydedilmesini sağlar*(flush)*; böylece bir arıza ya da enerji kesintisi durumunda ortaya çıkabilecek bilgi kaybını en aza indirir.

syslogd	Sistemde meydana gelen önemli olayları uygun **log** dosyalarına kaydeder. (Sisteme giren/çıkan kullanıcıları, **root** kullanıcı kimliğini alan kullanıcıları, donanımla ilgili sorunları vs. kaydeder.)
swapper	Ana belleğin yetmediği durumlarda diskte ayrılmış olan **swap** alanının sanki ana belleğin bir uzantısıymış gibi kullanılmasını sağlar.
init	Kullanıcı terminallerini dinler. Sisteme girmek üzere terminalini açan bir kullanıcıya rastlarsa, onun sisteme **login** edebilmesi için gerekli hazırlıkları yapar.
in.routed	Bilgisayar ağı üzerinde, çeşitli mesajların doğru bilgisayarlara ulaşmasından sorumludur.
in.named	Bilgisayar ağı üzerinde, adresi bilinmeyen bilgisayarların yer ve adreslerinin bulunabilmesi ile ilgili protokolleri yürütmekle görevlidir.
-tcsh	İlgili olduğu kullanıcı terminali için çalışan bir kabuk programıdır.

Önceki sayfalardaki **ps** komutlarının çıktılarının oldukça karmaşık bir görünümde olduklarını kabul etmek lazım. Ancak, gözünüzü, sürecin adının gösterildiği son kolonla **PID** kolonlarına alıştırırsanız, tipik bir UNIX kullanıcısının gerek duyabileceği tüm bilgileri almış olursunuz. Aradığınız sürecin adı bu listede varsa, programınız çalışıyor demektir (aslında çalışmaktan çalışmaya fark var... UNIX açısından çalışıyor, fakat sizin istediğiniz işleri yapmıyor olabilir; o başka mesele). **PID** kolonundaki numaraysa UNIX'in sizin programınızı izlemek ve denetlemek için kullandığı **süreç tanıtım numarasıdır** *(PROCESS ID).* UNIX altında çalışan programların herbirinin özgün *(unique)* bir **PID** numarası vardır. Bu numaranın büyüklüğü veya küçüklüğü bir anlam taşımaz. Nitekim, UNIX, programlara verdiği PID numarası 65535 e ulaşınca, tekrar birden başlatmakta bir sakınca görmez. (PID numaraları her UNIX'de 65535 le sınırlı değildir; bazı uyarlamalarda bu sayı daha da büyüyebilir; ancak bunun kullanıcılar için pek önemi yoktur.)

Eğer bir süreç (ya da program) çakılıp kaldıysa; ya da sizin istediğiniz gibi davranmıyorsa o süreci **ÖLDÜREBİLİRSİNİZ.** Deyim çok tatmin edici; değil mi?

```
% kill nnn                                    (kill process )
```

Bu komut, tanıtım numarası (**PID**) olan süreci öldürmek için kullanılır. Tek bir komutla birden fazla süreci beraber öldürebilirsiniz.

`kill 154 185 117` gibi...

Eğer öldürmek istediğiniz süreç ölmemek için direniyorsa,

```
% kill -9 nnn                                 şartsız öldürme
```

formunu deneyiniz. Eğer süreç gene ölmezse daha fazla uğraşmayınız. Bazı süreçler ölemezler. Bu tip süreçlere **zombie** adı verilir. (Hoş... Değil mi ?) **Zombie** süreçler genellikle pek sistem zamanı ya da bellek harcamazlar. Bu tip süreçlerin sistemde çalışır durumda olması genellikle zararsızdır.

Öldürülen bir süreç, daha önce başka süreçler yarattıysa; yani eğer bir *ebeveyn* süreçse ***(parent process)***, büyük olasılıkla o yavru süreçler de ***(child process)*** ölecektir. Bu nedenle mecbur olmadıkça süreçleri sona erdirmek için bu öldürme yöntemini kullanmayınız.

kill komutu ile sadece kendinize ait süreçleri öldürebilirsiniz. Sisteme, ya da başka kullanıcılara ait süreçleri öldürme yetkisi sadece **root** kullanıcıya aittir.

Zaman zaman klavyenizin kilitlendiği ve sistemin hiç bir komuta tepki göstermediği durumlarla karşılaşacaksınız. **Böyle bir durumda, bilgisayarı kapatıp açmayı aklınızdan bile geçirmemelisiniz.**

- Eğer UNIX bilgisayarını bir terminalden kullanıyorsanız, terminalinizi açıp kapatmayı bir denemenizde sistem açısından herhangi bir tehlike yoktur.
- Eğer bilgisayarı sistem konsolundan kullanıyorsanız (iş istasyonlarında olduğu gibi) ekranı kapatıp açmak bir yarar sağlamaz. Bilgisayarı kapatmayı düşünmemelisiniz dahi.. Peki ne yapılmalı?

- İlk denemeniz gereken Ctrl-Q tuşu. Daha önce yanlışlıkla Ctrl-S tuşuna basmış olabilirsiniz. Ctrl-S tuşu, ekrana gelen dökümleri durdurmak için kullanılan bir komuttur. (Ctrl-Q ise **'devam et'** anlamındadır). (**XON/XOFF** seri haberleşme protokolu).

- Olmazsa Ctrl-C tuşu ile çalışan işi kesmeyi deneyin. Gene olmuyorsa Ctrl-Z tuşuyla çalışan işi askıya almayı denemelisiniz. Her iki durumda da denemeniz başarılıysa ekranınızda **sistem hazır işaretini (prompt) göreceksiniz.** Hemen **ps** komutuyla çalışan işlerin bir listesini alın. Sorun yaratan sürecin numarasını (PID) öğrenip onu öldürmeyi deneyin. Ölmüyorsa çok ısrar etmeyin. Gerek duyarsanız, sistem yöneticisinden yardım isteyin.

- Eğer klavyeniz kilitlenmişse; sisteme bir başka terminal veya varsa, bilgisayar ağındaki bir başka bilgisayar üzerinden **login** etmeyi deneyin (**telnet** veya **rlogin** komutları). Bunu başarabiliyorsanız hemen **ps** komutuyla çalışan işlerin bir listesini alın. Sorun yaratan sürecin numarasını (PID) öğrenip onu öldürmeyi deneyin. Ölmüyorsa çok ısrar etmeyin. Gerek duyarsanız sistem yöneticisinden yardım isteyin.

- Hiç bir çareniz kalmadıysa en az 5 dakika bilgisayara dokunmadan bekleyin (eğer çalışıyorsa **update** daemon'u tampon belleği boşaltsın diye) sonra bilgisayarı kapatın. Tüm bilgisayarlarda olduğu gibi en az 30 saniye kadar bekleyip tekrar açın. **Bu tip zoraki işlemlerden sonra disk kayıtlarınızda büyük çaplı kayıplara hazırlıklı olmalısınız.**

% nohup	*no hangup*

Bir UNIX bilgisayarına ulaşmak için kullandığınız terminali ya da terminal gibi davranan bir PC'yi (Terminal Emulation yazılımı çalışan bir PC) kapatırsanız veya **logout** komutu ile sistem bağlantınızı keserseniz, o terminal bağlantısıyla ilgili tüm süreçler (hem ön, hem arka plandaki süreçler) UNIX tarafından öldürülür.

Ender de olsa, bazı durumlarda sistemden çıkmanıza rağmen, başlatmış olduğunuz bir işin kesilmeden devam ettirilmesini isteyebilirsiniz. Örneğin, **Internet** üzerinden çok uzun bir dosya çekiyor ve bu kopyalama işinin siz eve gittiğinizde de devam etmesini istiyor olabilirsiniz. Böyle bir durumda ilk akla gelen ''**logout** etmeden'' terminali açık bırakarak eve gitme çözümü pek iyi bir çözüm değildir. Sizin yokluğunuzda, açık bırakmış olduğunuz terminalin başına oturan birisi, sizin kimliğinizle hoşlanmayacağınız işler yapabilir.

İşte böyle durumlarda **nohup** komutu kullanılır. **logout** ettiğinizde kesilmesini istemediğiniz bir programı başlatırken kullanmanız gereken komut satırı

```
% nohup  komut [varsa parametreleri] &
```

olmalıdır.

% tcsh	*t c-shell*

tcsh, **csh**'e göre oldukça üstün özelllikleri olan bir kabuk programıdır; ancak, şimdilik hiç bir UNIX uyarlamasında standart olarak bulunmamaktadır. Sistem yöneticinize bir danışınız; eğer sisteminizde varsa, sizin için **login** kabuğu olarak **tcsh** çalıştırılmasını sağlamasını isteyiniz. (Sistem yöneticileri; kola, kahve, piza gibi rüşvetleri kabul ederler. Para falan teklif etmeyiniz. Paranın ne olduğunu bilseler, UNIX sistem yöneticisi olmazlardı...)

tcsh'in belki de en iyi iki özelliği, aynı MS-DOS'daki DOSKEY yardımıyla olduğu gibi, yukarı aşağı tuşlarla eski komutlar arasında dolaşmanızı sağlaması ve dosya adlarının tamamını yazmadan komut yazmanıza olanak sağlamasıdır. **tcsh** hakkında daha fazla reklama gerek yok. Sisteminizde varsa nasıl olsa öğrenirsiniz; yoksa, zaten özelliklerini öğrenip gıpta etmenin bir anlamı yok.

''tcsh'i nereden bulurum ?'' diyenlere ise cevabım **''Internet'den''** olacaktır. Birçok üniversitenin bilgisayarında, herkese açık alanlarda **tcsh** ve daha bir sürü ilginç program bulabilirsiniz**. ''Internet'de neyin nesi ?''** diyorsanız, bu kitabı okumakla daha fazla vakit kaybetmemenizi öneririm.