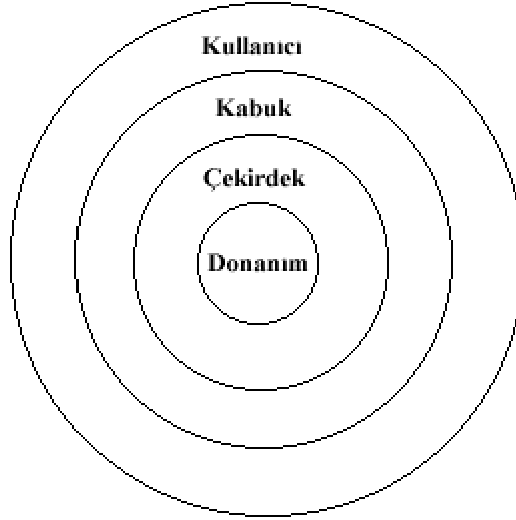
➢ Kabuk Nedir?

Kabuk kullanıcı ile çekirdek arasında bulunan bir yazılım katmanıdır. İşlevi kullanıcının yapılmasını istediği işlemleri yorumlayıp çekirdek katmana ne yapması gerektiğini anlatmaktır. Bu haliyle "Kabuk katman son kullanıcı için bir arayüzdür" denilebilir.



Kabuk bir "komut yorumlayıdır" (*command interpreter)* . Bu bağlamda sizden gelen komutları direkt olarak işletilmeye başlanmaz. Öncelikle kabuk tarafından yorumlanır ve işletimi bu aşamadan sonra gerçekleşir.

➢ Kabuk Neler Yapar?

Kabuk katman son kullanıcıya yönelik program çalıştırma, metakarakter kullanımı, girdi/çıktı yönlendirmesi, pipe işlemlerini yapar. Kabuk aynı zamanda bir programlama dili özelliği göstererek kullanıcının rutin/basit işlemlerini gerçekleştirmesine olanak sağlar.

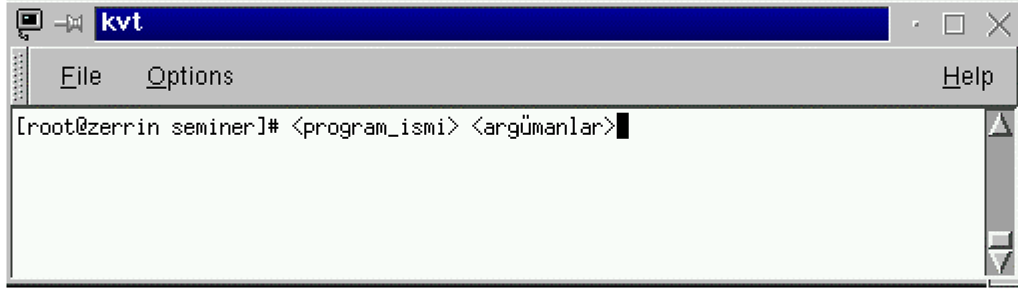
➢ Kabuk Çeşitleri

Günümüzde kullanılmakta olan çeşitli kabuklar şu şekilde sıralanabilir:

⇒ sh Bourne Shell
⇒ csh C Shell
⇒ ksh Korn Shell
⇒ bash Bourne Again Shell

Her kabuk'un kendine özgü özellikleri mevcuttur. Bir kabuk için yazılmış olan program bir diğer için çalışmayabilir. Bundan sonraki örneklerimiz "bash"e göre verilecektir.

1. Program Çalıştırma

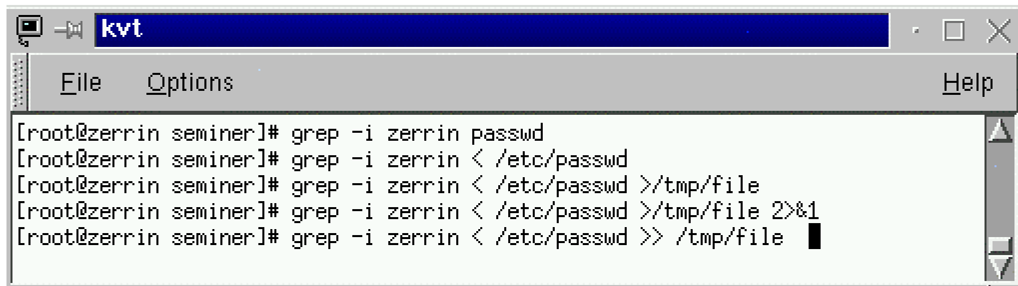


Komut satırında iken "program_ismi" ile çağırılan programa çeşitli argümanlar verilebilir. Bu argümanlar ile ne yapılacağı program içerisinde tanımlanmıştır. Genellikle '-' ile başlayan argümanlar programın işleyişi ile ilgili argümanlardır. Örneğin:

"grep" komutu kendisinden sonra verilen ve '-' karakteri ile başlamayan katarı bu argümandan sonra verilen dosya içinde bulmaya yarayan bir programdır. Yukarıdaki örnek komut "zerrin" katarını büyük küçük harf ayrımı yapmaksızın –ki bu özelliği "-i" argümanı sağlar- "/etc/passwd" dosyası içinde arar.

2. Girdi/Çıktı Yönlendirmeleri

Kabuk aracılığı ile programların girdileri/çıktıları başka kaynaklardan sağlanabilir. Şimdi bu konuyu örnekler eşliğinde inceleyelim:



Birinci örneğimizde "passwd" dosyasında "linux" katarı büyük-küçük harf ayrımı yapmadan aranır. Bu örnekte "passwd" dosyasına program (grep) ulaşır ve içeriğini KENDİ okur.

İkinci örneğimizde ise komut tek bir fark ile örnek bir'deki işin aynısını yapar. Bu komut ile "grep" dosyaya kendi ulaşmaz. Dosya içerisindeki verileri kabuk grep'e iletir.

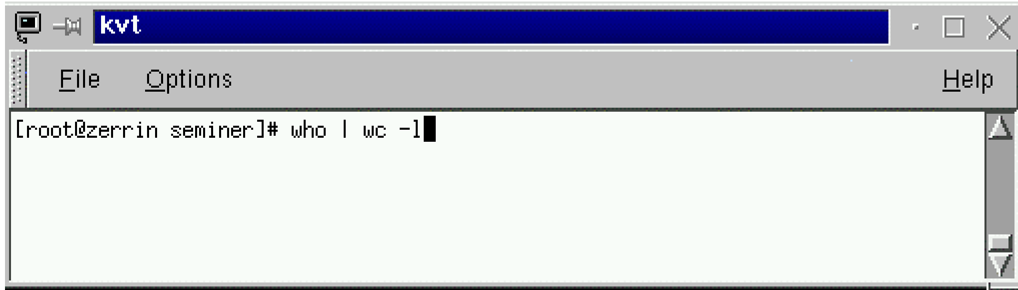
Üçüncü örneğimizde ikinci komut ile oluşan çıktımız kabuk tarafından "/tmp/file" kütüğüne yazılır. Eğer "/tmp/file" kütüğü daha önceden mevcut ise içerisindeki veri silinerek yeni veri yazılacaktır. Oluşan hata mesajları ise '2>' aracılığı ile boşluğa (hiçliğe-/dev/null) yönlendirilir. Eğer bu yönlendirme yapılmasa idi oluşacak hata mesajları ekrana yazılmaya devam edecekti.

Dördüncü örneğimizdeki fark ise '2>&1' karakter dizisinin bulunmasıdır. Bu durumda hata mesajlarıda normal mesajların yönlendirildiği yere yönlendirilir. Bu örnekte bu yer "/tmp/file" kütüğü olacaktır.

Son örneğimizde ise oluşan sonuç eğer mevcut ise dosyanın sonuna eklenecektir. Şayet dosya mevcut değil ise dosya yaratılıp içerisine sonuç yazılacaktır.

3. PIPE

Yukarıda anlatılan yöntemler ile yönlendirmeler programlar ve dosyalar arası yapılabilmektedir. Eğer yönlendirmelerin iki program arası yapılması isteniyorsa yani bir programın çıktısının diğer programa girdi olması isteniyorsa yapılacak işlem PIPE (|) olmalıdır. Örneğin;



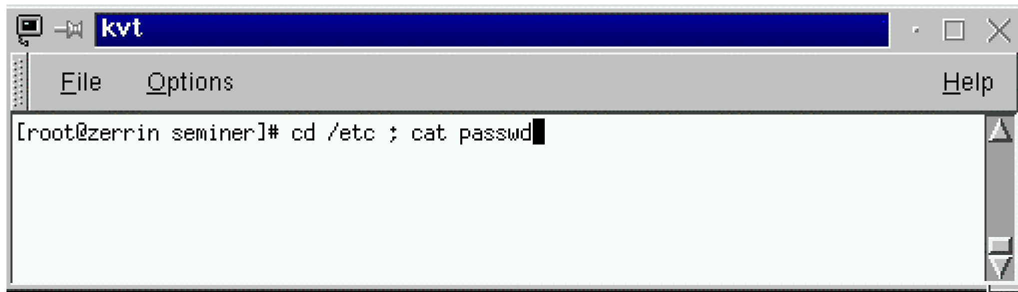
"who" komutu o anda sistemde mevcut kullanıcıları listeleyen komuttur. "wc –l" ile kendisine verilen dosyanın yada verinin kaç satırdan oluştuğunu bildirir. Bu durumda "wc –l" komutunun girdisi "who" komutunun çıktısıdır. Öyle ise bu komut bize o anda sistemde kaç kişinin bulunduğunu belirtecektir.

4. Metakarakterler

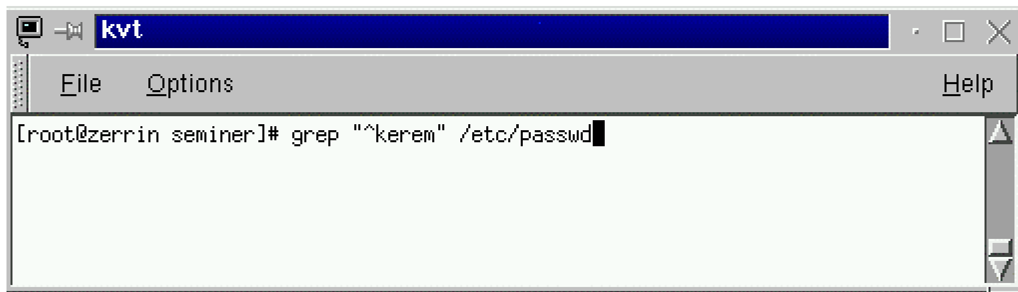
☺ *** :** Birden çok harfin yerine geçebilecek olan özel karakterdir.

☺ **? :** Tek bir harf yerine geçebilen özel karakterdir.

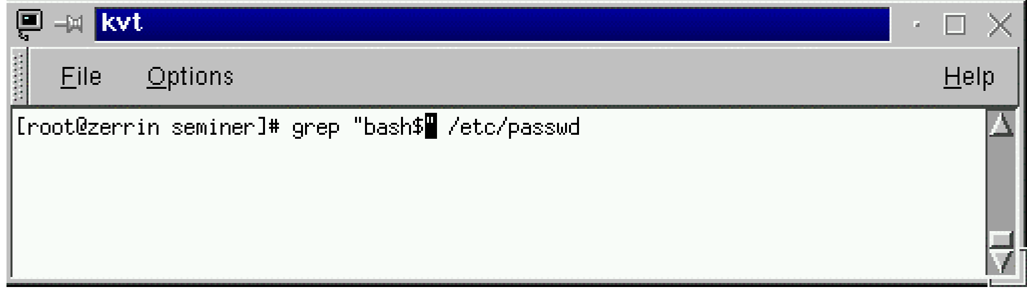
☺ **; :** Önce sol atafındaki, sonra sağ tarafındaki komutun birbirinden bağımsız ama sıra ile çalışmasını sağlayan özel karakterdir.



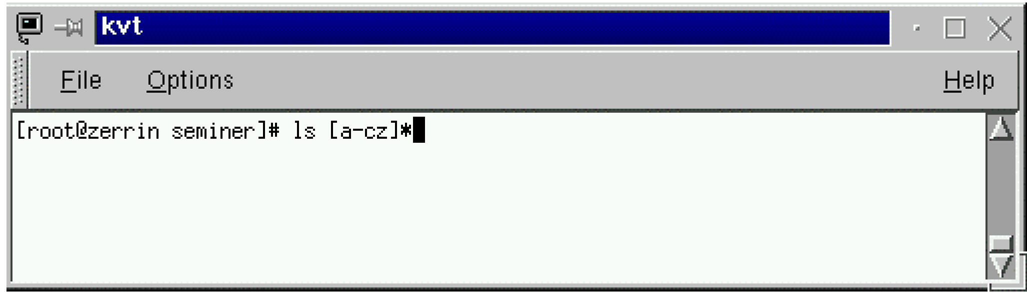
☺ **^ :** Kendisinden sonra gelen katarın satır başında bulunduğunu belirten özel karakterdir.



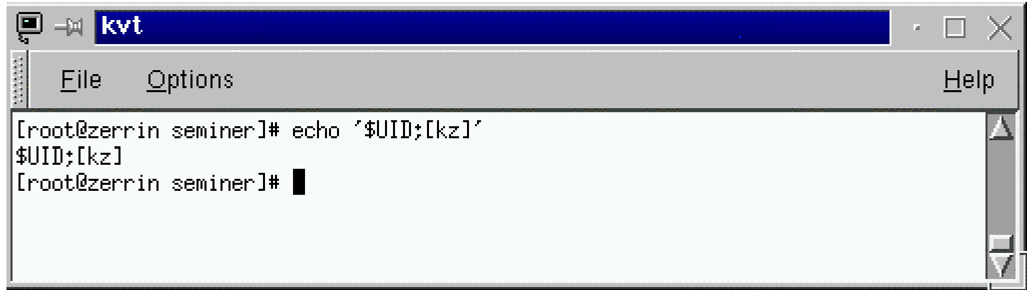
☺ **$:** İki anlamı bulunmaktadır. Eğer bir katarın sonunda bulunuyorsa belirtilen katarın satır sonunda bulunduğunu, bir katarın başında ise onun bir değişken olduğunu belirtir.



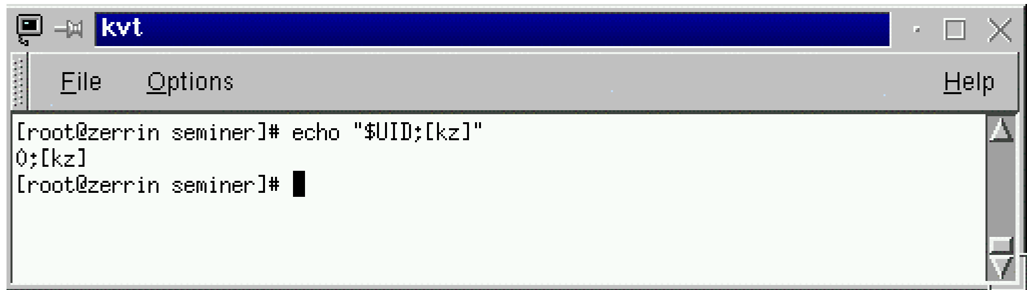
☺ **[ ] :** İki köşeli parantez arasında geçen harflerden biri anlamına gelen özel karakterdir.



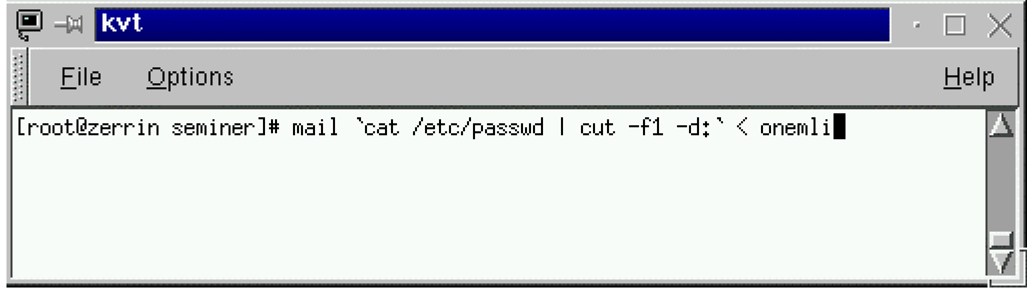
☺ **‘ (Tek Tırnak) :** İki tek tırnak arasında bulunan bütün karakterler anlamlarını yitirirler.



☺ **“ (Çift Tırnak) :** Değişkenler hariç bütün karakterler anlamlarını yitirirler.



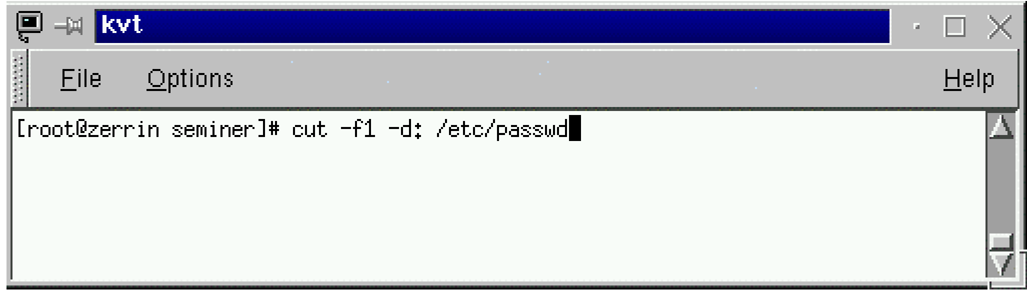
☺ **` (Eğik Tırnak) :** Kendisinden sonra gelen ve diğer ters tırnağa kadar devam eden komut olduğunu ve öncelikle bu komutun çalıştırılacağını belirtir.



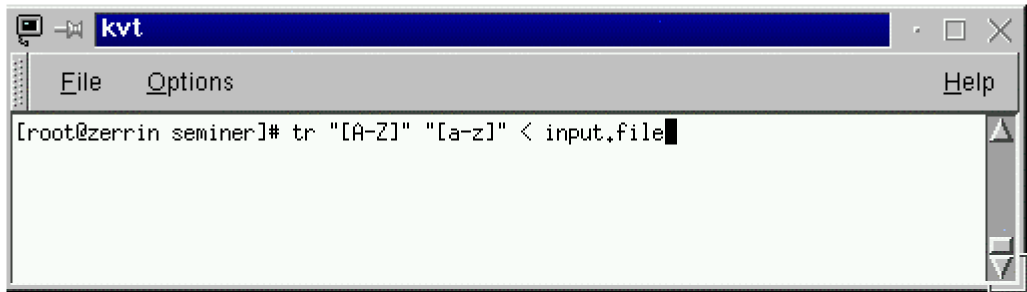
Yukarıdaki komut sistemde tanımlı olan tüm kullanıcılara “onemli” dosyasındaki mesajı gönderir.

➢ İleri Düzey Komutlar

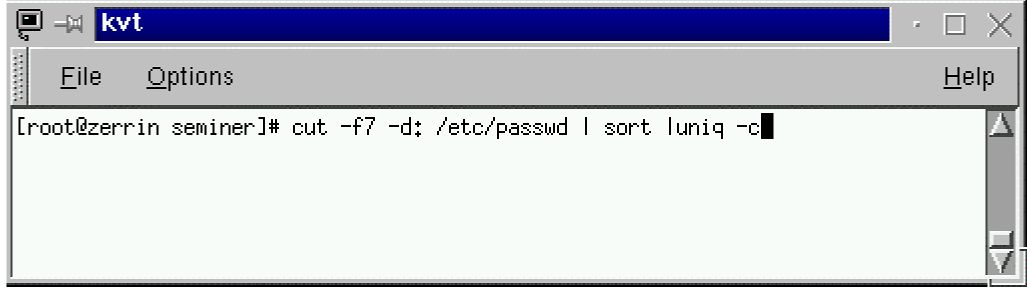
1. **Cut :** Belirli biçemdeki dosyaların belirli alanlarını almak için kullanılan bir komuttur. “-d” ile ayıraç karakteri “-f” ile istenilen bölüm numarası verilir.



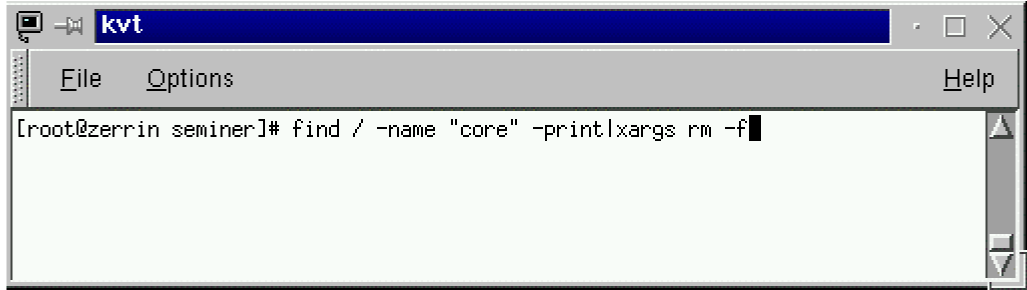
2. **Tr :** Translate’in kısa adı olan tr verilen iki harf dizileri arasında dönüşüm yapar.



3. **Grep :** Dosyaların içeriğinde arama yapmaya olanak sağlayan programdır.
4. **Sort :** Girdilerini alfabetik olarak sıralamak için kullanılan programdır.
5. **Uniq :** Birden çok defa tekrarlanan verileri teke indirgeyen, istenirse hangi veriden kaç adet olduğu sayabilen programdır.



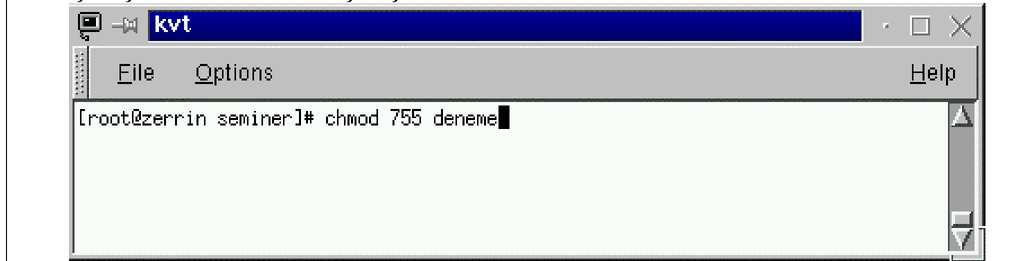
6. **Wc :** Kelime sayma işlemlerini gerçekleştiren programdır.
7. **Xargs :** Kendisine girdi için gelen verileri tek tek kendisinden sonraki programa argüman olarak veren programdır.



➢ Programlama

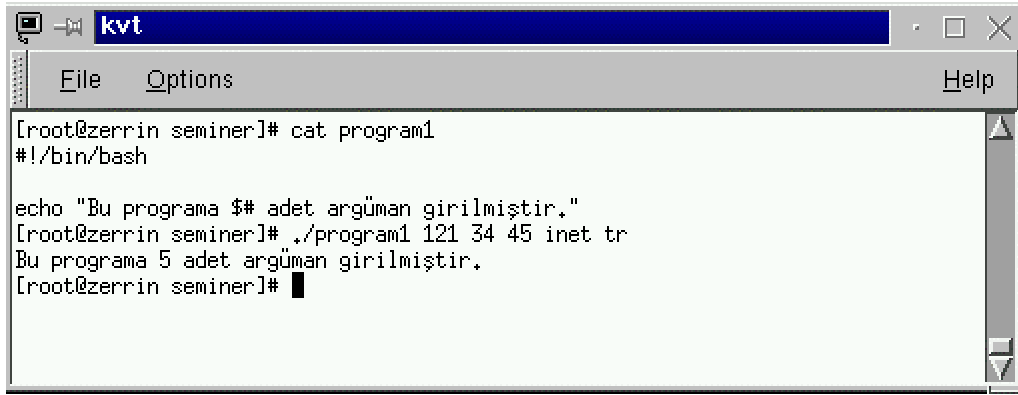
Kabuk programlama yapabilmek için ilave programlara ihtiyaç yoktur. Herhangi bir metin editörü (vi,pico,joe…) ile yazılabilir. Yalnızca uyulması gereken 2 kural bulunmaktadır.

1. Her programın başına hangi kabuk için yazıldığı “#!” karakterinden sonra belirtilmelidir.
2. Çalıştırılmadan önce çalıştırma hakkı verilmelidir.

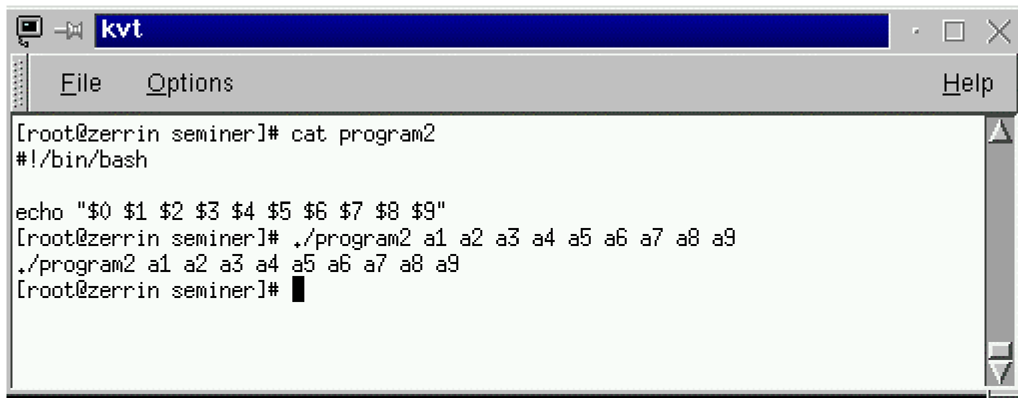


➢ Kabuk Değişkenleri

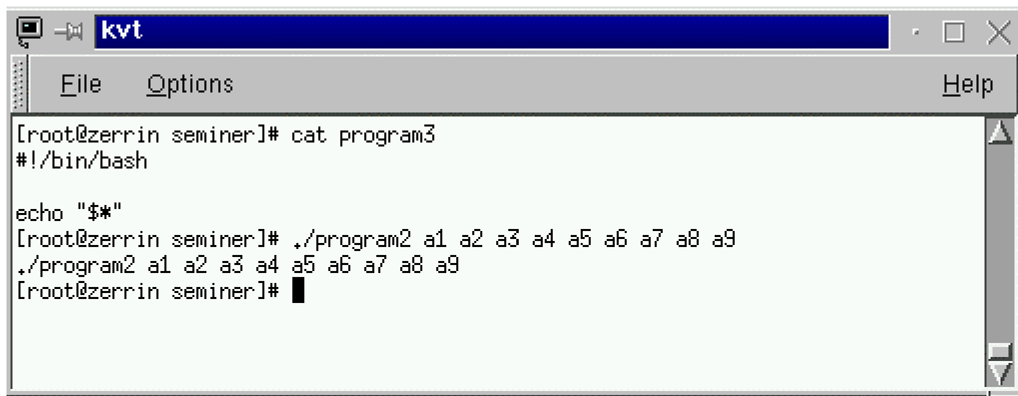
☺ **$# değişkeni :** Kabuk programına kaç adet argüman verildiği sayısını tutan değişkendir.



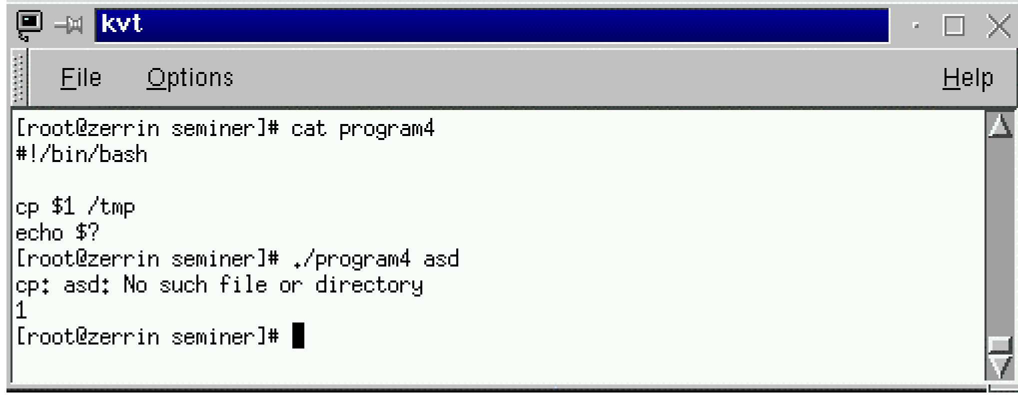
☺ **$0,$1,$2..$9 değişkenleri :** Kabuk programının tek tek argümanlarının bulunduğu değişkenlerdir.



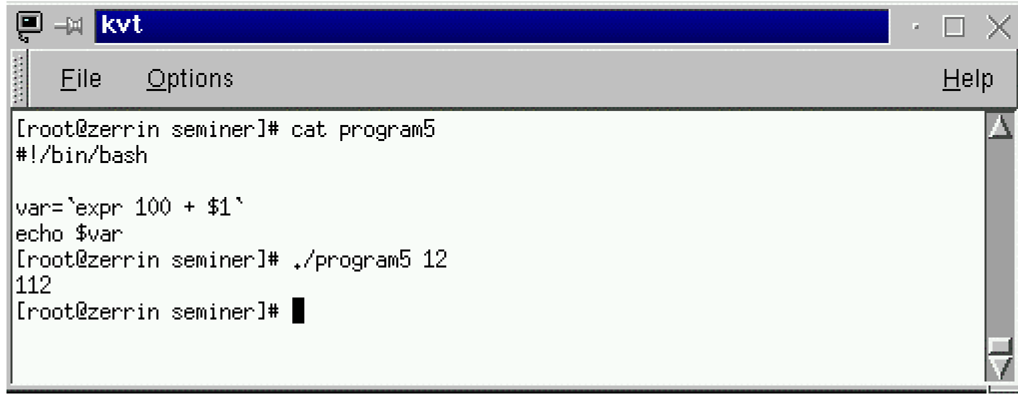
☺ **$* değişkeni :** Kabuk programına verilen tüm parametreleri birden içeren değişkendir.



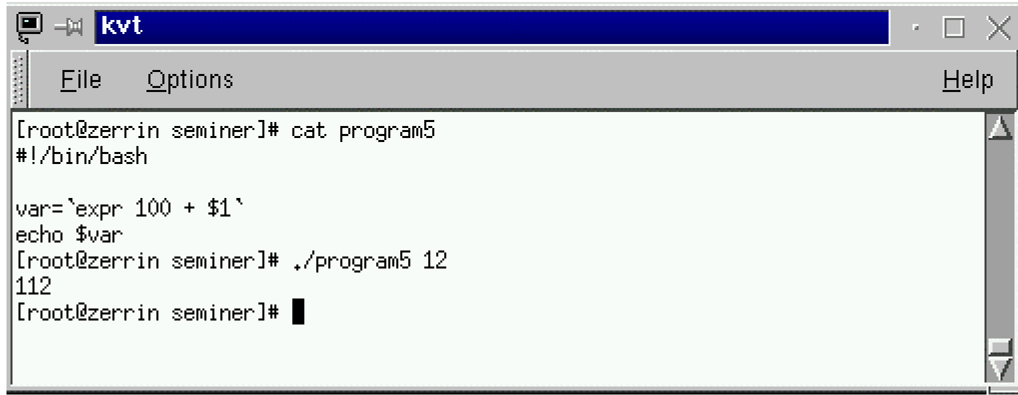
☺ **$? değişkeni :** Son gerçekleştirilen işlemin sonucunu tutan değişkendir. Eğer son işlem başarılı oldu ise değeri '0' başarısız ise '0''dan farklı bir değer alır.



☺ **$$ değişkeni :** O an çalışmakta olan kabuk programının görev numarasını barındıran değişkendir. Genelde geçici dosyalar yaratmak için kullanılır.



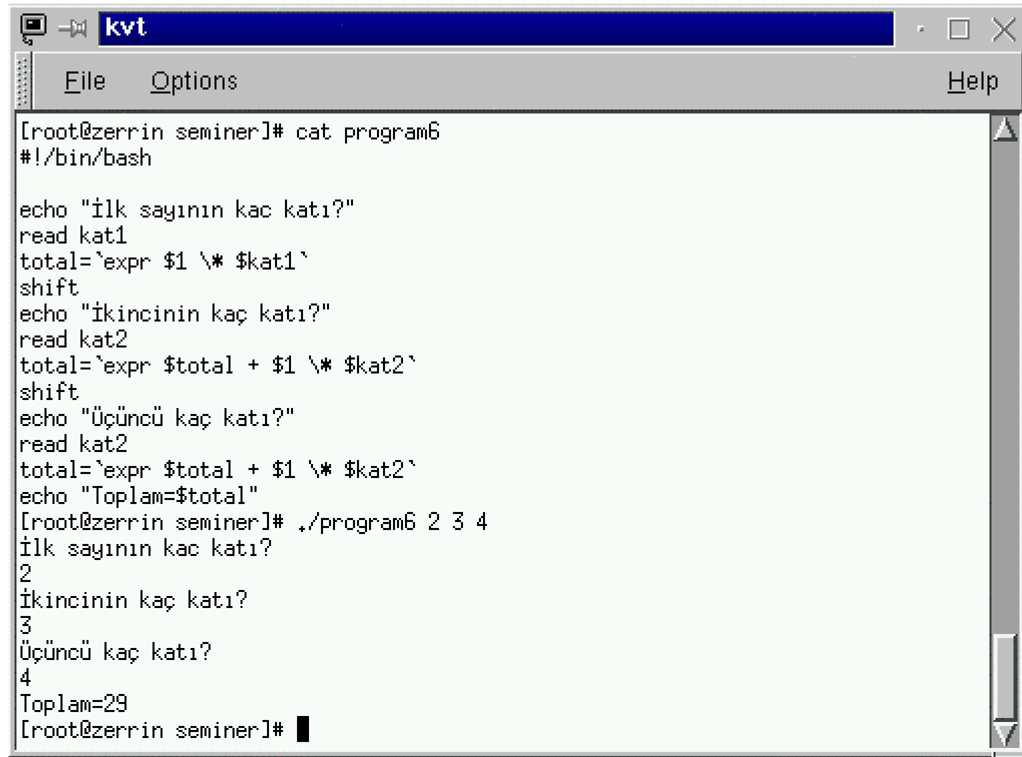
☺ **Değişken tanımlama :** Gerektiği durumlarda kullanıcı istediği zaman kullanmak üzere kendi değişkenlerini tanımlayabilir.



➢ Shift, Read, Expr komutları ve Açıklama Satırları

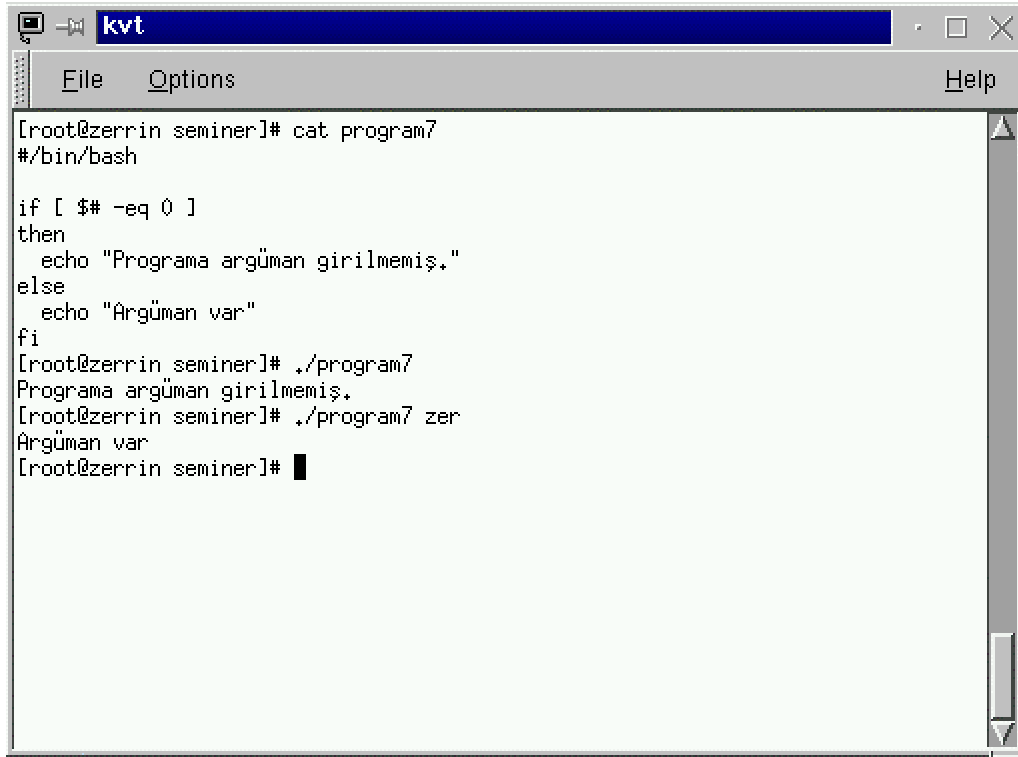
“shift” komutu programa verilen argümanların sıra ile kaydırılmasını sağlar. “read” komutu ile kullanıcıdan bilgi girilmesi istenen anlarda kullanılır. “expr” komutu ise kabuk programlarına matematiksel işlem yapmayı sağlar.

ile başlayan satırlar kabuk programlamada açıklama satırı olarak kullanılmaktadır.



➢ Test

İki türlü kullanımı olan test verilen veriler ve karşılaştırma işlemini yapar, sonuçta “doğru” yada “yalnış” değerlerinden biri döner. Kullanımlardan biri “test” yazısından sonra test katarının yazılması ile olur. Diğeri ise [ ] (köşeli parantez) ler arasında test katarının yazılması ile olur.



Test İle Kullanılan İşletmeler

a) **Tamsayı İşletmenleri :**

z1 –eq z2	z1 ve z2 birbirine eşit mi?
z1 –ne z2	z1 ve z2 birbirinden farklı mı?
z1 –qt z2	z1, z2'den büyük mü?
z1 –qe z2	z1, z2'den büyük eşit mi?
z1 –lt z2	z1, z2'den küçük mü?
z1 –le z2	z1, z2'den küçük eşit mi?

b) **Dosya İşletmenleri :**

-f dosya	Dosya mevcut mu?
-d dosya	Dosya bir dizin mi?
-s dosya	Dosya mevcut ve uzunluğu sıfırdan farklı mı?

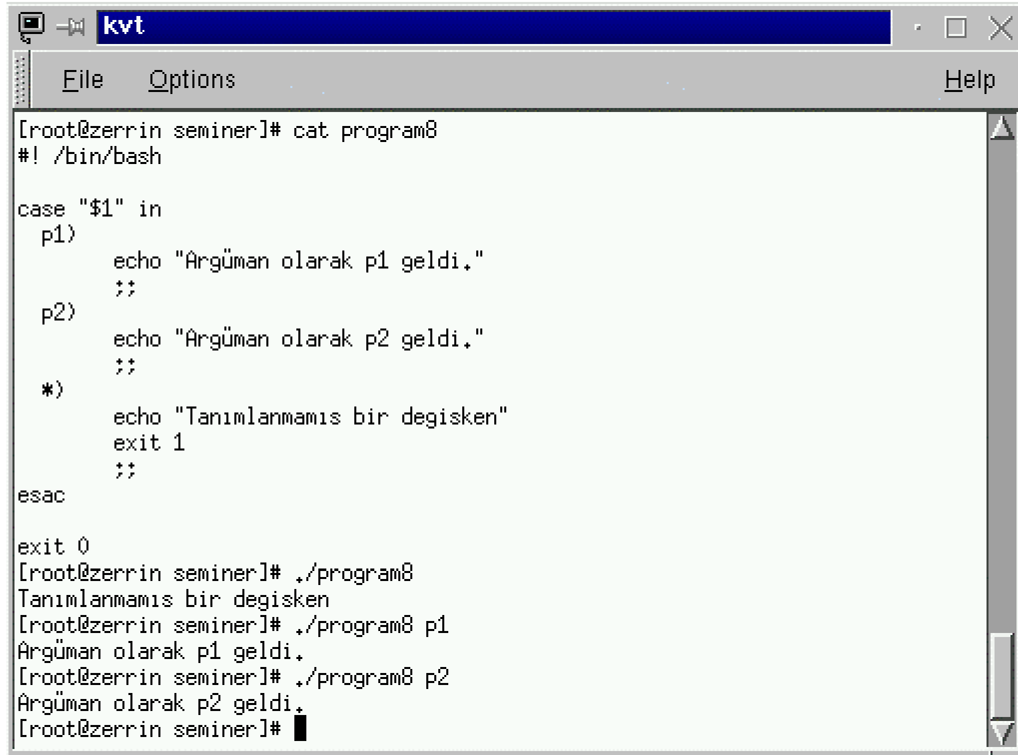
➢ Kontroller

1. if .. then .. else

if'den sonra gelen test'in sonucuna göre sapma yapan kontrol mekanizmalarıdır. Gerekli durumlarda "else" yerine "elif" kullanırak tekrar bir kontrol yapılabilir.

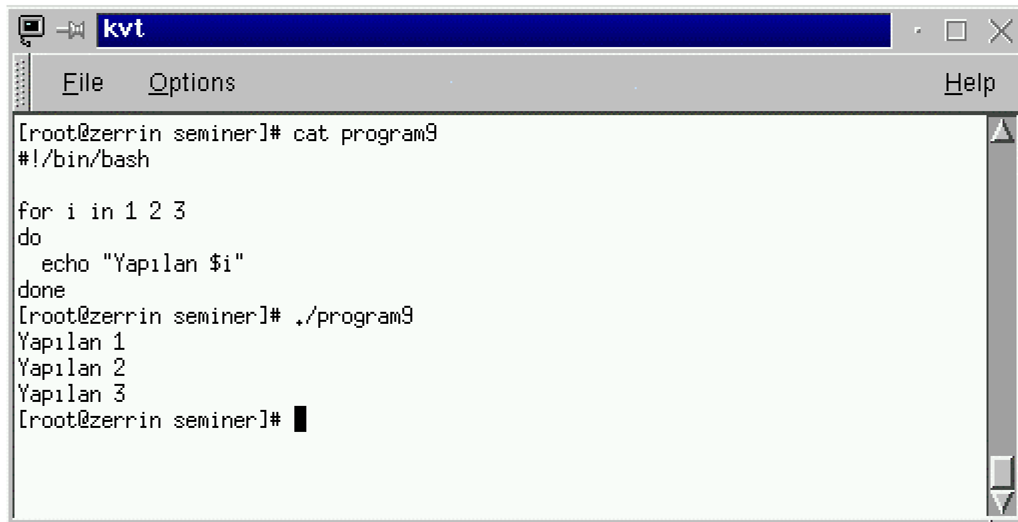
2. Case

"if .. then .. elif" yapısını daha düzenli olarak ele almaya yarayan kontrol yapısıdır.



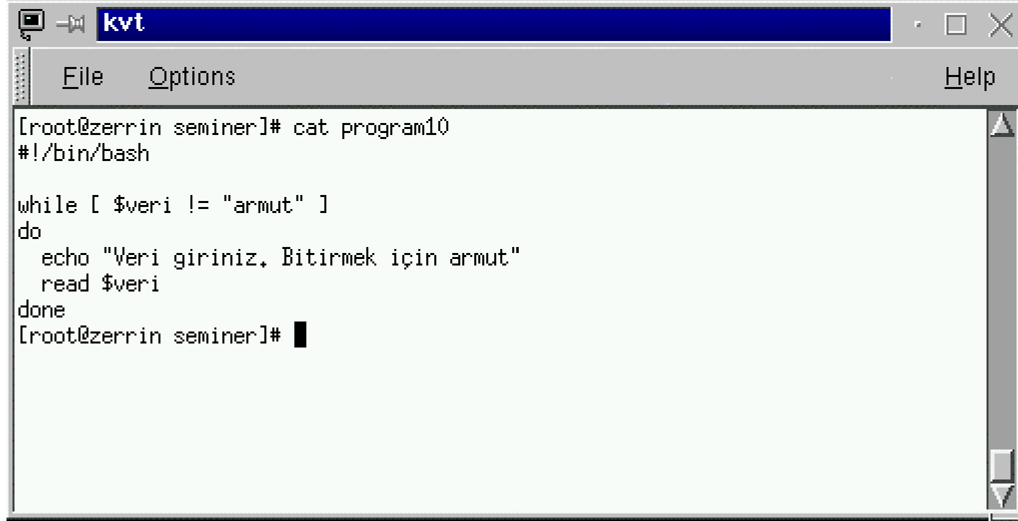
3. For

Belirli bir listede mevcut olan her eleman için yapılması istenen bir işlem varsa kullanılan yapıdır.



4. While

Bir testin doğru olduğu sürece yapılması istendiği durumlarda kullanılan kontrol yapısıdır.



5. Until

Bir testin doğru olduğu koşulda bitmesi istenen döngüler için kullanılan kontrol yapılarıdır.

6. Break

Döngülerin bitmesinin istendiği durumlarda kullanılan yapıdır. Dongü içerisinde iken break komutu kullanılırsa devam etmekte olan döngğ hemen kesilir ve döngüden sonraki komut işletilerek program işleyişine devam eder.

7. Continue

Döngülerin başladığı noktaya geri dönmesi istenildiği durumlarda kullanılan yapıdır. Döngü çalışmakta iken continue komutu kullanıldığında döngü içinde devam eden diğer komutlar işletilmez kontrol bulunan satıra geri dönülür.

➢ Örnek Programlar