GNU Bash Başvuru Kılavuzu

Bash için Başvuru Belgeleri

Yazan:
Chet Ramey
Case Western Reserve University

Yazan:
Brian Fox
Free Software Foundation

Çeviren:
Nilgün Belma Bugüner
<nilgun (at) belgeler·gen·tr>

1 Kasım 2006

Özet

Bu kitapta Bash kabuğunda bulunan özellikler kısaca açıklanmıştır (sürüm 3.2, 28 Eylül 2006).

Bash diğer kabukların özelliklerini içermekten başka onlarda bulunmayan pek çok özellik içerir. Bash'in kavramlarını alıntıladığı kabuklardan bazıları Bourne Kabuğu ('`sh`'), Korn Kabuğu ('`ksh`') ve C–kabuğudur ('`csh`' ile halefi olan '`tcsh`'). Konu başlıklarının dağılımı da, bu kabuklardan birinin konu dağılımından örnek alınmıştır.

Bu kılavuz Bash'de bulunan özelliklere kısa bir giriş olarak ele alınmalıdır. Kabuk davranışına tanımsal başvuru olarak Bash kılavuz sayfası kullanılmalıdır.

İçindekiler

Legal Notice

Yasal Uyarı

Feragatname

I. Giriş

İçindekiler

1. Bash Nedir?

Bash GNU işletim sistemi için bir kabuk ya da başka bir deyişle komut dili yorumlayıcısıdır. `Bourne-Again SHell` sözcüklerinde türetilmiş bir kısaltmadır. Bell Araştırma Laboratuarının Unix'inin yedinci sürümündeki, şu anki Unix kabuğu `sh`'ın atasının yazarı Stephen Bourne'a atfen bu isim verilmiştir.

Bash, **`sh`**'ın hemen hemen tüm özelliklerini ve Korn kabuğu olan **`ksh`** ile C kabuğu olarak bilinen **`csh`**'ın kullanışlı özelliklerini bir araya getirir. IEEE POSIX belirtiminin IEEE POSIX Kabuk ve Araçları bölümüne (IEEE Standardı 1003.1) uygun bir ürün olması amaçlanmıştır. **`sh`**'ın hem etkileşimli hem de programlama için kullanımını işlevsel olarak arttıran geliştirmeler içerir.

GNU işletim sistemi, **`csh`**'ın bir sürümü de dahil olmak üzere başka kabuklarla da teçhiz edilmişse de Bash öntanımlı kabuktur. Diğer GNU yazılımları gibi Bash'de bir çok işletim sistemine uyarlanabilir – MS–DOS, OS/2 ve Windows platformları için bağımsız olarak desteklenen sürümleri vardır.

2. Kabuk Nedir

Temelinde, bir kabuk, komutları çalıştırmaya yarayan basit bir makro işlemcisidir. Burada makro işlemcisi terimi, metinlerin ve sembollerin daha geniş ifadeler oluşturmak üzere genişletilmesi işlevini yerine getiren anlamındadır.

Bir Unix kabuğu, hem bir komut yorumlayıcısı hem de bir programlama dilidir. Bir komut yorumlayıcı olarak, GNU araçlarından zengin bir demeti barındıran bir arayüzdür. Programlama dili özellikleri ise bu araçları birarada kullanabilmeyi mümkün kılar. Komutları içeren dosyalar oluşturulabilir ve bu dosyaların kendileri birer komut haline gelebilir. Bu yeni komutlar `/bin` dizinindeki sistem komutları gibi kullanıcılara ve gruplara ortak kullandıkları işlemleri otomatikleştiren özelleştirilebilir ortamlar sağlayabilir.

Kabuklar etkileşimli ya da etkileşimsiz kullanılabilirler. Etkileşimli kipte, girdi klavyeden kabul edilirken, etkileşimsiz kipte bir dosyadan okunur.

Bir kabuk, GNU komutlarını hem eşzamanlı hem de eşzamansız çalıştırabilir. Kabuk, başka bir girdi kabul etmeden önce eşzamanlı komutların işlemlerini bitirmesini bekleyebileceği gibi, eşzamansız komutlara paralel olarak kabuğun ek komutları okumasını ve çalıştırmasını sağlayabilir. Yönlendirme yapıları ile bu komutların giriş ve çıkışlarının kolayca denetlenmesini mümkün kılar. Üstelik, komutların çalıştırıldığı ortam üzerinde de denetimi sağlar.

Kabuklar ayrıca ayrı araçlar üzerinden sağlanması sakıncalı olan ya da imkansız olan işlevselliği sağlamak üzere yerleşik komutların (yerleşikler – builtins) küçük bir demetini barındırır. Örneğin, **`cd`** (sayfa: 38), **`break`** (sayfa: 38), **`continue`** (sayfa: 39) ve **`exec`** (sayfa: 39)) komutları doğrudan kabuğun kendisini etkilediğinden kabuk dışında gerçeklenemezler. **`history`** (sayfa: 112), **`getopts`** (sayfa: 39), **`kill`** (sayfa: 85) veya **`pwd`** (sayfa: 40) yerleşikleri ise, diğerlerine karşın ayrı araçlar olarak gerçeklenebilirdi ama yerleşik komutlar olarak kullanılması daha faydalıdır. Tüm kabuk yerleşikleri kılavuzun ileri bölümlerinde anlatılmıştır.

Komutlar icra edilirken kabuğun esas gücü (ve karmaşıklığı) kabuğun gömülü programlama dilinden gelir. Diğer yüksek seviyeli programlama dilleri gibi kabuk da değişkenler, akış denetim yapıları, işlevler içerir ve ayrımlama yapabilir.

Kabuğun özellikleri programlama dilini güçlendirmekten ziyade özellikle etkileşimli kullanımı kuvvetlendirmeyi öne çıkarır. Bu etkileşim özellikleri iş denetimi, komut satırı düzenlemesi, komut geçmişi ve takma adlardır. Bu özelliklerin herbiri kılavuzun ileri bölümlerinde açıklanmıştır.

II. Terimler

Kılavuzun içinde kullanılan bazı terimler.

alan – (field)
Kabuk yorumlarından birinin sonucu olan bir metin birimidir. Yorumlama sonrası, bir komut icra edilirken oluşan alanlar komut ismi ve argümanları olarak kullanılır.

anahtar sözcük – (reserved word)
Kabukta özel anlamı olan bir sözcük. Anahtar sözcüklerin çoğunluğu `for` ve `while` gibi kabuk akış denetimi yapılarında kullanılır.

boşluk – (blank)
Bir boşluk ya da sekme (tab) karakteri.

çıkış durumu – (exit status)
Bir komutun kendini çağırana döndürdüğü değer. Değer, sekiz bit genişlikle sınırlıdır ve dolayısıyla en çok 255 değerini alabilir.

denetim işleci – (control operator)
Bir denetim işlevi gerçekleştiren bir sözcüktür. Bir satırsonu (LF) karakteri olabildiği gibi `||`, `&&`, `&`, `;`, `;;`, `|`, `(` veya `)` dizgeciklerinden biri de olabilir.

dizgecik – (token)
Kabuk tarafından tek birim olarak ele alınan bir karakter dizisi. Bir sözcük olabildiği gibi bir işleç de olabilir.

dosyaismi – (filename)
Bir dosyayı tanımlayan bir karakter dizgesidir.

dönüş durumu – (return status)
Çıkış durumu ile eşanlamlıdır.

isim – (name)
Bir harf veya altçizgi karakteri ile başlayan ve altçizgiler, harfler ve rakamlardan oluşan bir sözcük. İsimler, kabuk değişkenleri ve işlev isimleri olarak kullanılır. Ayrıca ***belirteç*** olarak da bilinir.

iş – (job)
Bir boruhattı (`|`) içeren bir süreç kümesi ve tamamı aynı süreç grubundan olarak onun alt süreci olan süreçlerin hepsi.

işleç – (operator)
Bir denetim ya da bir yönlendirme işleci. Yönlendirme işleçleri hakkında daha ayrıntılı bilgi edinmek için *Yönlendirmeler* (sayfa: 29) bölümüne bakınız.

iş denetimi – (job control)
Kullanıcıların süreçleri durdurma ve yeniden başlatmayı seçebildikleri bir mekanizma.

metakarakter – (metacharacter)
Tırnaklarla sarmalanmamış, tek başına bir anlamı olan bir karakter. Bir metakarakter bir boşluk olabildiği gibi `|`, `&`, `;`, `(`, `)`, `<` veya `>` karakterlerinden biri de olabilir.

özel yerleşik – (special builtin)

POSIX standardı tarafından özel olarak sınıflanmış bir yerleşik kabuk komutu.

POSIX
Unix'i temel alan bir açık sistem standartları ailesi. Bash, POSIX 1003.1 standardının Kabuk ve Araçlar bölümünü ile ilgilenir.

sinyal – (signal)
Sistem içinde bir eylem oluşturmak üzere çekirdek tarafından bir sürecin uyarılmasını sağlayan mekanizma.

sözcük – (word)
İşleç olmayan bir dizgecik.

süreç grubu – (process group)
Aynı süreç kimliğine (PID) sahip birbiriyle ilişkili süreçlerden oluşan küme.

süreç grubu kimliği – (process group ID)
Bir süre grubunu yaşamı süresince tanımlayan bir eşsiz belirteç.

yerleşik – (builtin)
Dosya sisteminin içinde bir yerlerde bulunan bir çalıştırılabilir program değil, kabuğun kendi içinde gerçeklenmiş komutlar.

III. Temel Kabuk Özellikleri

İçindekiler

Bash, **B**ourne-**A**gain **SH**ell sözcüklerinden oluşturulmuş bir kısaltmadır. Bourne kabuğu, ilk olarak Stephen Bourne tarafından yazılmış olan geleneksel Unix kabuğudur. Bourne kabuğunun içerdiği tüm yerleşik komutlar Bash'de de bulunmaktadır. Değerleme ve ayrımlama kuralları 'standart' Unix kabuğu için posix belirtiminden alınmıştır.

Bu kısımda, kabuk bloklarının kurgulanmasına kısaca değinilmiştir: komutlar, denetim yapıları, kabuk işlevleri, kabuk parametreleri, kabuk açılımları, isimli dosyalarla doğrudan çalışılabilmeyi sağlayan yönlendirmeler ve kabuk komutlarının çalıştırılması.

1. Kabukta Sözdizimi

Kabuk bir girdiyi okuduktan sonra bir dizi işlem yürütür. Eğer girdi bir açıklama başlangıcı içeriyorsa, kabuk açıklama sembolünü (`#`) ve satırın kalanını yoksayar. Aksi takdirde kabuk, girdiyi okur ve onu sözcüklere ve işleçlere ayırır, ayrımlama kurallarını kullanarak sözcükleri ve karakterleri anlamlandırır.

Bundan sonra bu dizgecikleri komutlar ve bağlı yapılar olarak çözümler, özel anlamları olan sözcükleri ve karakterleri kaldırır, kalanları yorumlar, gerekiyorsa girdi ya da çıktıyı yönlendirir, belirtilen komutları çalıştırır, komutların çıkış durumu oluşturmasını bekler ve bu çıkış durumunu denetim veya işlem yapılabilmesi için çıktılar.

1.1. Kabuk İşlemleri

Aşağıdaki kısa açıklamada bir komutun okunması ve çalıştırılması sırasındaki kabuk işlemleri anlatılmıştır. Temel olarak, kabuk şunları yapar:

1. Girdiyi *bir dosyadan* (sayfa: 35) veya `-c` *çağrı seçeneğinin* (sayfa: 67) bir argümanı olarak sağlanan bir dizgeden ya da kullanıcının uçbiriminden okur.
2. *Ayrımlama* (sayfa: 13) bölümünde açıklandığı gibi ayrımlama kurallarına uygun olarak girdiyi sözcüklere ve işleçlere ayırır. Bu dizgecikler metakarakterlerle ayrılır. *Takma ad yorumlaması* (sayfa: 75) da bu adımda uygulanır.
3. Dizgecikleri basit ve birleşik *komutlar* (sayfa: 14) olarak çözümler.
4. Yorumlanan dizgecikleri *dosya listelerine* (sayfa: 27), komutlara ve argümanlara ayırarak çeşitli *kabuk açılımlarını* (sayfa: 22) uygular.
5. Varsa gerekli *yönlendirmeleri* (sayfa: 29) uyguladıktan sonra yönlendirme işleçleri ile terimlerini argüman listesinden kaldırır.
6. Komutu *çalıştırır* (sayfa: 32).

7. İsteğe bağlı olarak komutun işini bitirmesini ve *çıkış durumu* (sayfa: 35) üretmesini bekler.

1.2. Ayrımlama

Ayrımlama işlemi kabukta, sözcükler ve karakterlerden özel anlamlandırmanın kaldırılması anlamında kullanılmıştır. Ayrımlama özel karakterlerin özel davranışlarını ortadan kaldırarak, onların anahtar sözcükler olarak tanımlanmasını ya da parametreler olarak algılanmasını engellemek için kullanılır.

Kabuk *metakarakterlerinin* (sayfa: 9) herbirinin kabuğa özel anlamı vardır ve bunlar sadece kendileri olarak ele alınacaksa tırnak içine alınmalıdır. Komut geçmişine ilişkin özellikler kullanılacaksa *geçmiş yorumlama* (sayfa: 90) karakteri olan `!` işaretinin geçmiş yorumlaması yapılacak şeklinde algılanmasını engellemek için bu karakter tırnak içine alınmalıdır. Geçmiş yorumlaması ile ilgili daha ayrıntılı bilgi edinmek için *Bash'in Geçmişsel Yetenekleri* (sayfa: 111) bölümüne bakınız.

Üç tane ayrımlama mekanizması vardır: *önceleme karakteri*, *tek tırnak* ve *çift tırnak*.

1.2.1. Önceleme karakteri

Tırnak içine alınmamış bir tersbölü işareti `\` Bash önceleme karakteri adını alır. Kendisinden sonra gelen bir karakterin ya da sayısal bir karakter sabitinin değeri `satırsonu` karakteri hariç korunur. Eğer bir satırın sonunda tek başına ve tırnak içine alınmamış bir `\` karakterini izleyen bir `satırsonu` karakteri varsa yani, satırın sonunda bir `\satırsonu` çifti varsa, bu satır ve altındaki satır tek bir satır olarak yorumlanır (Bu durumda bu çift girdiden kaldırılır yani yoksayılır).

1.2.2. Tek Tırnak

Tek tırnak (`'`) içine alınmış karakterlerin değeri korunur. Ancak tek tırnak karakteri, önceleme karakteri ile öncelenmiş olsa bile, tek tırnak karakterleri arasında bulunamaz.

1.2.3. Çift Tırnak

Çift tırnak (`"`) içine alınmış karakterlerin `$`, `` ` ``, `\` ve geçmiş yorumlaması etkinse **`!`** karakterleri hariç değerleri korunur. Çift tırnak içine alınmış `$` ve `` ` `` karakterleri özel anlamlarını korurlar (*Kabuk Yorumları* (sayfa: 22) bölümüne bakınız). `\` karakteri ise `$`, `` ` ``, `"`, `\` veya `satırsonu` karakterlerini öncelemek için kullanılmışsa özel anlamını korur. Çift tırnak içindeki önceleme karakterlerinden bu karakterleri öncelemekte kullanılanları kaldırılır. Özel anlamı olmayan karakterleri öncelemekte kullanılan önceleme karakteri değişmeden kalır. Bir çift tırnak, önceleme karakteri ile öncelenerek çift tırnaklar arasında kullanılabilir. Geçmiş yorumlaması etkinse, çift tırnaklar içinde bir **`!`** varsa ve önceleme karakteri ile öncelenmiş değilse geçmiş yorumlaması uygulanır.

Özel karakterler olan `*` ve `@` çift tırnaklar arasında özel anlama sahiptir (*Kabuk Parametrelerinin Yorumlaması* (sayfa: 24) bölümüne bakınız).

1.2.4. ANSI–C Ayrımlaması

`$'dizge'` biçimindeki sözcükler özel olarak ele alınır. Sözcük, içindeki ANSI C standardında belirtilen tersbölü öncelemeli karakterler yorumlanarak dizgeye dönüştürülür. Varsa tersbölü öncelemeli karakterler aşağıdaki gibi yorumlanır:

`\a`
: uyarı (zil)

`\b`
: gerisilme

`\e`

önceleme karakteri (ANSI C değil)

`\f`
: sayfa ileri

`\n`
: satırsonu

`\r`
: satırbaşı

`\t`
: yatay sekme

`\v`
: düşey sekme

`\\`
: tersbölü

`\'`
: tek tırnak

`\nnn`
: Sekizlik tabanda `nnn` (3 rakamlı bir sayı) olarak değeri verilen sekiz bitlik karakter

`\xHH`
: Onaltılık tabanda `HH` (iki onaltılık rakam) olarak değeri verilen sekiz bitlik karakter

`\c`*x*
: Bir <**Ctrl**>–<*x*> karakteri

Sonuç, dolar işareti yokmuşçasına tek tırnaklar içine alınır.

1.2.5. Yerele Özel Çeviri

`$` işareti ile öncelenmiş çift tırnaklı bir dizge (`$"dizge"`) yerel dile çevrilir. Eğer yerel ayarları `C` ya da `POSIX` ise `$` işareti yoksayılır. Yerel dile çevrilen dizgeden çift tırnaklar kaldırılmaz.

Bazı sistemler `LC_MESSAGES` ortam değişkeni ile belirtilen yerelin ileti kataloglarını kullanırken bazıları da `TEXTDOMAIN` ortam değişkenindeki, genellikle `.mo` sonekli dosya ismi ile belirtilen ileti kataloğunu kullanır. `TEXTDOMAIN` değişkeni kullanıldığında, dosyanın bulunduğu yerin `TEXTDOMAINDIR` ortam değişkeni ile belirtilmesi gerekir. Bazıları da bu her iki değişkeni birlikte kullanabilir: *TEXTDOMAINDIR*`/LC_MESSAGES/`*TEXTDOMAIN*`.mo`.

1.3. Betiklerde Açıklamalar

shopt (sayfa: 52) yerleşik komutunun etkin olduğu etkileşimli veya etkileşimsiz tüm kabuklarda `#` karakteri ile başlayan bir sözcük ve satır sonuna kadar devamındaki tüm karakterler yoksayılır. `interactive_comments` seçeneği etkinleştirilmemiş bir etkileşimli kabukta açıklamalara izin verilmez. Bu seçenek etkileşimli kabuklarda öntanımlı olarak etkindir. Bir kabuğu etkileşimli hale getirmek için ne yapılması gerektiği *Etkileşimli Kabuklar* (sayfa: 70) bölümünde anlatılmıştır.

2. Kabuk Komutları

echo a b c gibi basit bir kabuk komutu, komutun kendisi ile kendisinden sonra gelen ve boşluklarla ayrılmış argümanlardan oluşur.

Basit komutların çeşitli yollarla birarada kullanılmasıyla daha karmaşık kabuk komutları elde edilebilir: bir boruhattı ile birinci basit komutun çıktısı ikinci basit komuta girdi olarak aktarılabilir, bir döngü, bir koşullu ifade ya da başka gruplamalar altında birarada kullanılabilir.

2.1. Basit Komutlar

Bir basit komut en çok rastlanan komut çeşitidir. Boşluklarla ayrılmış sözcüklerden oluşur ve kabuk *denetim işleçleri* (sayfa: 9)nden biri ile sonlanır. İlk sözcük genelde çalıştırılması istenen komuttur, kalanlar ise bu komutun argümanlarıdır.

Bir basit komutun *dönüş durumu* (sayfa: 9), POSIX 1003.1 `waitpid` işlevi ile sağlanan bir *çıkış durumu* (sayfa: 35) ya da komut bir `n` sinyali ile sonlandırılmışsa `128+n`'dir.

2.2. Boruhatları

Bir *boruhattı*, `|` karakteri ile ayrılmış basit komutlar dizisidir.

Bir boruhattının sözdizimi:

```
[time [-p]] [!] komut1 [| komut2 ...]
```

Bir boruhattındaki her komut bir kanal ile sonrakine bağlıdır. Böylece her komut bir önceki komutun çıktısını okur.

`time` anahtar sözcüğü, boruhattındaki iş sonuçlandıkça istatistiklerin basılmasını sağlar. İstatistikler geçen zaman, kullanıcı ve komutun çalıştırılması sırasında kullanılan sistem zamanından oluşur. `-p` seçeneği POSIX çıktı biçemini değiştirmekte kullanılır. `TIMEFORMAT` (sayfa: 66) değişkeninde belirtilen biçemleme dizgesi ile zamanlama bilgilerinin gösterim biçimi ayarlanabilir. Bu biçemler hakkında bilgi *Bash Değişkenleri* (sayfa: 60) bölümünde bulunabilir. `time` anahtar sözcüğü ile kabuk yerleşikleri, kabuk işlevleri ve boruhatlarının zamanlama bilgileri alınabilir. Harici bir `time` komutu ile bu işlem bu kadar kolay olamıyor.

Bir boruhattı eşzamanlamasız (*Komut Listeleri* (sayfa: 15) bölümüne bakınız) çalıştırılmadığında kabuk, boruhattındaki tüm komutların işlerini bitirmesini bekleyecektir.

Bir boruhattındaki her komut kendi altkabuğunda (*Komut Çalıştırma Ortamı* (sayfa: 33) bölümüne bakınız) çalıştırılır. Bir boruhattının çıkış durumu **pipefail** seçeneği etkin olmadıkça son komutun çıkış durumudur (*set Yerleşiği* (sayfa: 49) bölümüne bakınız). **pipefail** etkinse, borutattının dönüş durumu sıfırdan farklı bir değerle çıkan son (en sağdaki) komutun değeridir, değilse ve tüm komutlar başarı ile sonlanmışsa dönüş durumu sıfırdır. Bir boruhattı anahtar sözcüklerden biri olan `!` karakteri ile başlıyorsa, çıkış durumu, açıklanan çıkış durumunun mantıksal tersidir. Kabuk bir değer döndürmeden önce boruhattındaki tüm komutlar sonlanıncaya kadar bekler.

2.3. Komut Listeleri

Bir *liste*; `;`, `&`, `&&`, veya `||` karakteri ile ayrılmış ve isteğe bağlı olarak `;`, `&`, veya `newline` karakterlerinden biri ile sonlandırılmış bir ya da daha fazla sayıdaki boruhattından oluşur.

Bu listelerin işleçleri, `&&` ile `||` aynı öncelikte ve kendi aralarında aynı öncelikte olan `;` ile `&` işleçlerinden daha önceliklidir.

Komutları ayırmak için kullanılan satırsonu karakterleri *bir liste içinde* iki nokta üstüste karakterlerinin eşdeğerleri kabul edilir.

Bir komut `&` denetim işleci ile sonlandırılmışsa, kabuk komutu eşzamanlamaksızın bir altkabukta çalıştırır. Bu *artalanda çalışma* olarak bilinir. Kabuk, komutun işini bitirmesini beklemez ve 0 (doğru) çıkış durumu ile döner. *İş denetimi* (sayfa: 83) etkin değilse ve herhangi bir dolaylı yönlendirmenin bulunmaması halinde eşzamanlamasız komutun girdisi `/dev/null`'daki girdi olur.

`;` ile ayrılmış komutlar sırayla çalıştırılır ve kabuk her komutun sonlanmasını bekler. Dönüş durumu son komutun çıkış durumudur.

`&&` ve `||` denetim işleçleri sırasıyla VE ve VEYA listeleri oluşturur. VE listesinin sözdizimi:

komut1 `&&` *komut2*

komut2 sadece ve sadece *komut1* sıfır çıkış durumu ile dönmüşse çalıştırılır.

VEYA listesinin sözdizimi

komut1 `||` *komut2*

komut2 sadece ve sadece *komut1* sıfırdan farklı bir çıkış durumu ile dönmüşse çalıştırılır.

VE ve VEYA listelerinin çıkış durumu daima son komutun çıkış durumudur.

2.4. Birleşik Komutlar

Birleşik komutlar kabuk programlama oluşumlarıdır. Her oluşum bir anahtar sözcükle veya denetim işleci başlar ve sonlandırıcı bir anahar sözcük veya işleç ile sonlanır. Bir birleşik komut ile ilişkili bir yönlendirme (bkz, *Yönlendirmeler* (sayfa: 29)) aksi belirtilmedikçe birleşik komut içindeki tüm komutlara uygulanır.

Bash komutları gruplamak ve onları tek bir birim olarak çalıştırmak için döngüleri, koşullu komutları ve gruplama mekanizmalarını sağlar.

2.4.1. Döngüler

Bash aşağıdaki döngüleri destekler.

Bilgi

Komutların sözdizimi içinde herhangi bir yerde `;` karakterine rastlanırsa bu karakter bir ya da daha fazla sayıda satırsonu karakteri ile değiştirilir.

2.4.1.1. `until`

`until` deyiminin sözdizimi:

```
until sınama–komutları; do artbileşen–komutlar; done
```

artbileşen–komutlar, *sınama komutları*nın çıkış durumları sıfırdan farklı olduğu sürece çalıştırılır. Deyimin çıkış durumu ise ya *artbileşen–komutlar* içindeki son çalıştırılan komutun çıkış durumudur ya da hiç komut çalıştırılmamışsa sıfırdır.

2.4.1.2. `while`

`while` deyiminin söz dizimi:

```
while sınama–komutları; do artbileşen–komutlar; done
```

artbileşen–komutlar, *sınama komutları*nın çıkış durumları sıfır olduğu sürece çalıştırılır. Deyimin çıkış durumu ise ya *artbileşen–komutlar* içindeki son çalıştırılan komutun çıkış durumudur ya da hiç komut çalıştırılmamışsa sıfırdır.

2.4.1.3. `for`

`for` deyiminin sözdizimi:

```
for isim [in sözcük ...]; do komutlar; done
```

Sözcükler yorumlanır ve sonuçlanan listenin isim ile bağlantılı her üyesi için komutlar çalıştırılır. Deyimin **`in`** *sözcük* parçası yoksa, **`in "$@"`** belirtilmiş gibi kümeyi oluşturan her parametre için komutlar birer kere çalıştırılır (*Özel Parametreler* (sayfa: 21) bölümüne bakınız). Deyimin dönüş durumu çalıştırılan son komutun çıkış durumudur. Sözcüklerin yorumlanmasından bir üye elde edilememişse bir komut çalıştırılmaz ve sıfır çıkış durumu oluşur.

`for` deyimi için aşağıdaki sözdizimi de desteklenmektedir:

```
for (( ifade1 ; ifade2 ; ifade3 )) ; do komutlar ; done
```

Önce, *ifade1* aritmetik ifadesi aşağıda açıklanan kurallara bağlı olarak değerlendirilir (*Kabuk Aritmetiği* (sayfa: 74) bölümüne bakınız). *ifade2* artimetik ifadesinin değeri sıfır oluncaya kadar tekrar tekrar değerlendirilir. *ifade2* aritmetik ifadesinin sıfırdan farklı olduğu durumlarda komutlar çalıştırılır ve *ifade3* aritmetik ifadesi değerlendirilir. Verilmeyen ifade için 1 varmış gibi işlem yapılır. Dönüş değeri listedeki son çalıştırılan komutun çıkış durumudur. Geçersiz bir ifadenin varlığı halinde ise çıkış durumu yanlış olacaktır.

`break` (sayfa: 38) ve **`continue`** (sayfa: 39) deyimleri döngü denetiminde kullanılabilir.

2.4.2. Koşullu Çalıştırma

2.4.2.1. `if`

`if` deyiminin sözdizimi:

```
if sınama–komutları; then
  artbileşen–komutlar;
[elif diğer–sınama–komutları; then
  diğer–altbileşenler;]
[else
  karşı–altbileşenler;]
fi
```

sınama–komutları listesi yorumlandığında dönüş durumu sıfırsa, *artbileşen–komutlar* çalıştırılır; sıfırdan farklı ise **`elif`**'lerin *diğer–sınama–komutları* listesi yorumlanır ve bunların dönüş durumuna göre ya *diğer–altbileşenler* ya da *karşı–altbileşenler* çalıştırılır. Deyimin tamamının dönüş durumu çalıştırılmış olan son komutun çıkış durumudur. Bir komut çalıştırılmamışsa ve hiçbir koşul doğru sonuç vermemişse sıfır döner.

2.4.2.2. `case`

`case` deyiminin sözdizimi:

```
case sözcük in [ [(] kalıp [| kalıp]...) komutlar ;;]... esac
```

`case` deyimi *sözcük* ile eşleşen ilk *kalıp*'a karşı düşen *komutlar*'ı çalıştırır. Kabuk seçeneği **`nocasematch`** etkinse eşleşme alfabetik karakterlerin büyüklüklerine bakılmaksızın uygulanır (bkz, *shopt Yerleşiği* (sayfa: 52)).

Çok sayıda kalıp kullanılmışsa kalıpları ayırmak için | işleci, kalıp listesini sonlandırmak için ise) işleci kullanılır. Her **case** deyimi bir ;; işleci ile sonlandırılmalıdır. *sözcük* bir *kalıp* ile eşleştirmeye çalışılmadan önce, ayrımlama uygulanır ve yaklaşık (~), parametre, komut ve aritmetik yorumlamalarına tabi tutulur.

Herbiri ;; işleci ile sonlandırılmış çok sayıda **case** deyimi ardarda kullanılabilir. Ancak bunların içinden sadece ilk *sözcük*–*kalıp* eşleşmesine bağlı *komutlar* çalıştırılır.

Aşağıdaki örnek betikte **case** deyimi hayvanların bazı özellikleri için kullanılmıştır:

```
echo -n "Bir hayvan ismi yazınız: "
read HAYVAN
echo -n "$HAYVAN "
case $HAYVAN in
  at | kedi | maymun) echo -n "dört bacaklıdır" ;;
   kanarya | kanguru) echo -n "iki bacaklıdır" ;;
                  *) echo -n "için bir veri yok" ;;
esac
```

Hiçbir *kalıp* eşleşmezse dönüş durumu sıfırdır. Aksi takdirde son çalıştırılan komutun çıkış durumu döndürülür.

2.4.2.3. select

The **select** menü üretiminde kolaylık sağlar. **for** deyimine benzer bir sözdizimi vardır:

select *isim* [**in** *sözcük ...*]; **do** *komutlar*; **done**

sözcük ... listesi yorumlanarak öğe listesi üretilir. Öğe listesindeki her öğe başına bir numara eklenerek standart hataya çıktılanır. Eğer **in** *sözcük ...* çifti verilmezse, **in** "$@" verilmiş gibi konuma bağlı parametreler basılır ve standart girdide PS3 istemi ile girdi beklenir. Listede belirtilen numaralardan biri girdi olarak verilirse, o konuma bağlı sözcük ile *isim* eşleştirilir. Girdi satırı boş verilirse, EOF verilinceye kadar komut istemi tekrarlanır. Listede belirtilenler dışında verilen her değer için *isim* null ile eşleştirilir. Okunan satır REPLY değişkeninde tutulur.

Her seçimden sonra bir **break** komutu ile **select** sonlandırılıncaya kadar seçime bağlı olarak *komutlar* çalıştırılır.

Aşağıdaki betik örneğinde, bulunulan dizin içindeki dosyalar listelenir ve kullanıcıdan birini seçmesi istenir. Seçilen dosyanın ismi ve parantez içinde verdiğiniz girdi gösterilir:

```
select fname in *;
do
  echo $fname \($REPLY\) seçtiniz.
  break;
done
```

2.4.2.4. ((…))

((*ifade*))

Verilen aritmetik ifade *Kabuk Aritmetiği* (sayfa: 74) bölümünde açıklandığı gibi yorumlanır. Eğer ifadenin değeri sıfırdan farklı ise dönüş durumu sıfırdır; aksi takdirde 1 dir. Bu gösterim,

let "*ifade*"

ile eşdeğerdedir.

2.4.2.5. [[…]]

[[*ifade*]]

Koşullu *ifade*nin yorumuna bağlı olarak 0 ya da 1 ile döner. İfadeler *Bash Koşullu İfadeleri* (sayfa: 72) bölümünde açıklanan önceliklere göre yorumlanır. `[[` ile `]]` arasındaki sözcüklere sözcük ve dosyaismi yorumlaması uygulanmaz, sadece yaklaşık (~), parametre ve değişken yorumlamalarıyla aritmetik, komut ve süreç yorumlamaları ve ayrımlama uygulanır.

-f gibi mantıksal işleçler öncelikliler arasında sayılabilmeleri için tırnak içine alınmamış olmalıdırlar.

`==` ve `!=` işleçleri kullanıldığında, işlecin sağındaki dizge bir kalıp olarak ele alınır ve *Kalıp Eşleme* (sayfa: 28) bölümünde açıklanan kurallara göre eşleştirme yapılır. Kabuk seçeneği **nocasematch** etkinse eşleşme alfabetik karakterlerin büyüklüklerine bakılmaksızın uygulanır (bkz, *shopt Yerleşiği* (sayfa: 52)). Dizge kalıpla eşleşirse ('==') 0, eşleşmezse ('!=') 1 ile döner. Kalıbın bir bölümünün özellikle dizge olarak yorumlanması isteniyorsa o bölüm tırnak içine alınabilir.

== ve **!=** işleçleriyle aynı öncelik sırasına sahip bir iki terimli daha vardır: **=~** işleci. Kullanıldığında, işlecin sağındaki dizge bir gelişkin düzenli ifade olarak ele alınır ve buna uygun eşleştirilir (`regex`3'teki gibi). Dizge kalıpla eşleşirse dönüş değeri 0, aksi takdirde 1'dir. Eğer düzenli ifade sözdizimsel olarak hatalıysa koşullu ifade 2 değeri ile döner. Kabuk seçeneği **nocasematch** etkinse eşleşme alfabetik karakterlerin büyüklüklerine bakılmaksızın uygulanır (bkz, *shopt Yerleşiği* (sayfa: 52)). Düzenli ifade içindeki parantezli altifadelerle eşleşen altdizgeler dizi değişkeni **BASH_REMATCH**'de saklanır. **BASH_REMATCH**'in 0 indisli elemanı düzenli ifadenin tamamı ile eşleşen dizgeyi içerer. **BASH_REMATCH**'in n indisli elemanı ise düzenli ifadenin n'inci parantezli altifadesi ile eşleşen dizgeyi içerir.

İfadeler aşağıdaki işleç öncelik sırasına göre yorumlanır:

`(` *ifade* `)`

ifade'nin değeri ile döner. Önceliği arttırmak için kullanılır.

`!` *ifade*

ifadenin değeri yanlışsa doğru döner.

ifade1 `&&` *ifade2*

ifade1 ve *ifade2*, her ikisi de doğru ise sonuç doğrudur.

ifade1 `||` *ifade2*

ifade1 ya da *ifade2*, her ikisinden biri doğru ise sonuç doğrudur.

`&&` ve `||` işleçleri ile sonucu elde etmek için *ifade1* ifadesinin değeri yeterliyse *ifade2* yorumlanmaz.

2.4.3. Komutların Gruplanması

Bash komutları gruplayarak tek bir birim gibi çalıştırmayı sağlayan iki yöntem içerir. Gruplanan komutlara sadece komut listesi içindeki yönlendirmeler uygulanabilir. Örneğin liste içindeki tüm komutların çıktısı tek bir veri akışına yönlendirilebilir.

2.4.3.1. `()` ile gruplama

`(` *liste* `)`

Parantez içinde gruplama bir altkabuk ortamının oluşturulması ile sonuçlanır (bkz, *Komut Çalıştırma Ortamı* (sayfa: 33)) ve *liste* içindeki her komut bu altkabukta çalıştırılır. Değişken atamaları da sadece bu altkabuk için geçerli olur.

2.4.3.2. `{}` ile gruplama

```
{ liste; }
```

Bu tür gruplamada ise bir altkabuk oluşturulmaz, listedeki komutlar doğrudan kabukta çalıştırılır. *liste* bir ; veya bir satırsonu karakteri ile sonlandırılmalıdır.

Altkabuk oluşturmaya ek olarak iki gruplama arasında tarihsel sebeplere bağlı olarak küçük bir fark daha vardır. Kaşlı ayraçlar *anahtar sözcük*tür, dolayısıyla *liste*'den birer *boşluk* ile ayrılmalıdır. Parantezler ise işleçtir ve *liste*'den boşluklarla ayrılmamış bile olsalar kabuk tarafından ayrı olarak ele alınırlar.

Her iki gruplama için de dönüş durumu *liste*'nin çıkış durumudur.

3. Kabuk İşlevleri

Kabuk işlevleri, komutları isimli bir grup altında toplayıp gerektiğinde basit bir komut olarak grubu ismiyle çağırarak içindeki komutların çalıştırılmasını sağlar. Kabuk işlevleri için bir altkabuk ya da yeni bir süreç oluşturulmaz, içindeki komutlar doğrudan mevcut kabukta yorumlanır.

İşlevler için aşağıdaki sözdizimi uygulanır:

```
[ function ] isim () birleşik–komut [ yönlendirmeler ]
```

Bu, *isim* isminde bir kabuk işlevi oluşturur. `function` anahtar sözcüğü isteğe bağlıdır. `function` kullanılmışsa parantezler isteğe bağlıdır. *birleşik–komut* işlevin gövdesini oluşturur (bkz, *Birleşik Komutlar* (sayfa: 16)). Bu komut normal olarak `{` ile `}` arasındaki bir *komut–listesi*nden oluşur, fakat evvelce açıklanan birleşik komutlardan biri olabilir. *isim* bir komut ismi olarak verildiğinde *birleşik–komut* çalıştırılır. İşlev çalıştırılırken işlevle ilişkilendirilmiş *yönlendirmeler de uygulanır* (sayfa: 29).

Bir işlev tanımı **unset** yerleşiğinin `-f` seçeneğinde kullanılarak silinebilir (bkz, *unset Yerleşiği* (sayfa: 43)).

İşlev tanımının dönüş durumu bir sözdizimi hatası veya aynı isimde bir işlev olmadıkça sıfırdır. İşlev çalıştırıldığında, işlevin dönüş durumu gövdesindeki komutlardan sonuncusunun dönüş durumu olur.

Tarihsel sebeplerle, kaşlı ayraçlar anahtar sözcük olduklarından dolayı kaşlı ayraç olarak işlenebilmeleri için diğer sözcüklerden boşluklarla ya da satırsonu karakterleri ile ayrılmaları gerekir. Ayrıca, kaşlı ayraç kullanımında, *komut–listesi* bir noktalı virgül, bir **&** ya da bir satırsonu karakteri ile sonlandırılmalıdır.

Bir işlev çalıştırılırken, argümanları *Konumsal Parametreler* (sayfa: 21) olarak işlenir. Özel parametre olan `#` kullanılarak konumsal parametrenin konumu değiştirilebilir. Ancak, 0. konumsal parametre değiştirilemez. İşlevin çalıştırılması sırasında işlevin ismi `FUNCNAME` değişkeninin ilk elemanında tutulur. **declare** yerleşiği kullanılarak işlevin **trace** özelliği belirtilmedikçe veya **set** yerleşiği ile **-o functrace** seçeneği etkinleştirilmedikçe **DEBUG** ve **RETURN** tuzaklarının miras alınması dışında komut çalıştırma ortamının tüm diğer yönleri bir işlev ile çağrıcısı arasında aynıdır. Ayrıca *trap Yerleşiği* (sayfa: 42)'ne de bakınız.

Eğer komut listesinde `return` yerleşik komutu varsa, işlev tamamlandıktan sonra icra işlev çağrısından sonraki komuta geçerek devam eder. **RETURN** tuzağı ile ilişkilendirilmiş bir komut icra kaldığı yerden devam etmeden önce çalıştırılır. İşlev tamamlandığında, konumsal parametrelerin ve `#` özel parametresinin değerleri işlev çağrısından önceki değerlerine döndürülür. `return` ile birlikte bir sayı verilmişse, bu işlevin dönüş durumudur; verilmemişse `return` komutundan önceki son çalıştırılan komutun çıkış durumu işlevin dönüş durumudur.

İşleve özel değişkenler `local` yerleşiği ile belirtilebilir. Bu değişkenler sadece işlev ve onun komut listesindeki komutlar için geçerlidir.

İşlev isimleri ve tanımları **declare** (sayfa: 45) veya **typeset** (sayfa: 57) yerleşik komutlarının `-f` seçeneği ile listelenebilir. **declare** veya **typeset** komutunun `-F` seçeneği sadece işlev isimlerini listeleyecektir (eğer **extdebug** kabuk seçeneği etkinse isteğe bağlı olarak kaynak dosyası ismi ve satır numarasını da). Bu yolla işlevler, **export** (sayfa: 39) yerleşiğine `-f` seçeneği belirtilerek altkabuklara otomatik olarak ihraç edilebilir.

Aynı isimli kabuk işlevleri ve değişkenleri kabuğun çocuklarına çok sayıda eşdeğer isimli girdinin ortam içinde aktarılmasına sebep olacağına dikkat ediniz. Bunun sorunlara yol açabileceği durumları dikkate alınız.

İşlevler bir diğerinin içinden çağrılabilir. Bu şekilde iç içe çağrılara bir sınırlama getirilmemiştir.

4. Kabuk Parametreleri

Bir *parametre* değerlerin saklandığı bir öğedir. Bir `isim`, bir sayı ya da aşağıda listelenmiş özel karakterlerden biri olabilir. Bir *değişken* bir `isim` ile donatılmış bir parametredir. Bir değişkenin bir *değeri* vardır, isteğe bağlı olarak öznitelikleri de olabilir. Öznitelikler, **declare** (sayfa: 45) yerleşik komutu kullanılarak atanabilir.

Bir parametre bir değer atanarak belirtilir. Boş dizge geçerli bir değerdir. Bir değişken belirtildikten sonra sadece **unset** (sayfa: 43) yerleşik komutu ile kaldırılabilir.

Bir değişkenin belirtilmesi aşağıdaki sözdizimi ile yapılabilir:

isim=[*değer*]

Eğer *değer* verilmezse, değişkene değer olarak boş dizge verilmiş kabul edilir. Tüm *değer*lere yaklaşık (~), parametre ve değişken yorumlamaları ile komut, aritmetik yorumlamaları ve ayrımlama (ayrıntılar aşağıda) uygulanır. Bir değişken tamsayılardan oluşmuş özelliklere sahipse, `$((...))` yorumlaması kullanılmamış bile olsa değer bir aritmetik ifade olarak değerlendirilir (bkz, *Aritmetik Yorumlama* (sayfa: 27)). `"$@"` kullanımı dışında (aşağıda açıklanmıştır), sözcük çözümlemesi uygulanmaz. Dosya ismi çözümlemesi uygulanmaz. Atama deyimleri **alias**, **declare**, **typeset**, **export**, **readonly** ve **local** yerleşik komutlarının argümanları olarak verilebilir.

Atama deyiminin bir kabuk değişkenine veya *dizi* (sayfa: 76) indisine bir değer atadığı bağlamda **+=** işleci sona ya da değişkenin önceki değerine ekleme yapmakta kullanılabilir. **+=** işlecinin bir değişkene uygulandığı durumda değişkenin tamsayı özniteliği etkinse *değer* bir aritmetik ifade olarak değerlendirilir ve değişkenin o anki değerine eklenir. **+=** işlecinin bir *dizi* (sayfa: 76) değişkenine birleşik atama kullanılarak uygulandığı durumda ise değişkenin değeri değiştirilmez (**=** kullanılmış gibi) ve yeni değer dizinin en büyük indisinden bir büyük indiste başlayan diziye eklenir. Dizge değerli bir değişkene uygulandığında, *değer* yorumlandıktan sonra değişkenin değerinin ardına eklenir.

4.1. Konumsal Parametreler

Bir *konumsal parametre* tek hane olarak `0` dışında bir ya da daha fazla haneli bir sayı ile belirlenen bir parametredir. Konumsal parametreler, kabuk argümanlarından çağrı sırasında ya da özel olarak **set** yerleşik komutu kullanılarak oluşturulur. `N.` konumsal parametreye `${N}` olarak ya da eğer `N` tek haneli bir sayı ise `$N` olarak erişilebilir. Konumsal parametreler **set** (sayfa: 49) yerleşik komutu ile atanabilir ve **shift** (sayfa: 41) yerleşik komutu ile kaldırılabilir. Konumsal parametreler, bir *kabuk işlevinin* (sayfa: 20) çalıştırılması sırasında geçici olarak değiştirilebilir.

Birden fazla rakamdan oluşan konumsal parametrelerin çağrılması sırasında sayı kaşlı ayraçlar içine alınmalıdır.

4.2. Özel Parametreler

Kabuğun bazı özel parametreleri vardır, bu parametreler atanamaz sadece gerektiğinde kullanılabilir:

*

Birden başlayan konumsal parametreler olarak yorumlanır. Çift tırnak içine alınarak kullanıldığında, her parametresi `IFS` değişkeninin ilk karakterinin ayraç olduğu tek sözcüklük bir dizgedir. Yani `"$*"` sözcüğü `"$1c$2c..."` sözcüğüne eşdeğerdir. Buradaki `c` karakteri `IFS` değişkeninin değerinin ilk karakteridir. Eğer `IFS` değişkeni belirlenmemişse boş bir dizgedir. Bu durumda parametreler bir ayraç karakteri olmaksızın yanyana dizilir.

`@`

Birden başlayan konumsal parametreler olarak yorumlanır. Çift tırnak içine alınarak kullanıldığında, her parametresi ayrı bir sözcük olarak yorumlanır. Yani `"$@"` sözcüğü `"$1" "$2" ...` sözcüklerine eşdeğerdir. Eğer çift tırnak yorumlaması bir sözcük içinde vuku bulursa ilk parametrenin yorumu özgün sözcüğün başlangıç parçasıyle ve son parametrenin yorumu da özgün sözcüğün son parçasıyla bağlıdır. Konumsal parametrelerin bulunmaması halinde `"$@"` ve `$@` hiçlik olarak yorumlanır, yani parametreler silinmiş gibi olur.

`#`

Konumsal parametrelerin numaraları onluk sayılar olarak yorumlanır.

`?`

En son çalıştırılan önalandaki boruhattının çıkış durumudur.

`-`

(Kabuğun kendisi tarafından (örneğin `-i` seçeneği olarak) ya da **`set`** yerleşik komutu ile çağrı sırasında belirtilmiş komut satırı seçenekleri olarak yorumlanır.

`$`

Kabuğun süreç kimliğidir (PID). Bir `()` altkabuğunun varlığı halinde süreç kimliği olarak altkabuğun değil ana kabuğun süreç kimliği döner.

`!`

Artalanda (eşzamanlamasız) en son çalıştırılan komutun süreç kimliğidir.

`0`

Kabuğun ya da kabuk betiğinin ismidir. Bu kabuğun ilklendirilmesi sırasında belirlenir. Bash bir komut dosyası çalıştırıyorsa, `$0` bu dosyanın ismidir (*Kabuk Betikleri* (sayfa: 35) bölümüne bakınız).

Bash `-c` seçeneği ile başlatılmışsa (*Bash'in Çağrılması* (sayfa: 67) bölümüne bakınız), `$0` varsa seçenekle verilen argümandır; yoksa Bash'ı çağıran dosyanın ismidir.

`_`

Kabuk ilk başlatıldığında, kabuğu çağırmakta kullanılan mutlak dosya ismi ya da argüman listesinde belirtilmiş olarak çalıştırılan kabuk betiğinin ismidir. Müteakip olarak, yorumlama sonrası, önceki komuta son argüman olarak yorumlanır. Ayrıca, çalıştırılan her komutu çağırmakta kullanılacak tam dosyayolunu tutar ve ilgili komutun ayarladığı ortam değişkenine yerleştirilir. Örneğin, eposta denetlenirken, bu parametre eposta dosyasının ismini tutar.

5. Kabuk Yorumları

Yorumlama, komut satırı `dizgecik`lere ayrıldıktan sonra uygulanır:

- kaşlı ayraçların yorumlanması
- yaklaşık (~) yorumlaması
- parametre ve değişkenlerin yorumlanması
- komut ikamesi
- aritmetik yorumlama
- sözcüklere ayırma
- dosyaisminin yorumlanması

Yorumlama işlemi yukarıda verilen sırada uygulanır.

Destekleniyorsa bazı sistemlerde, bir ek yorumlama daha vardır: *süreç ikamesi*. Bu; parametre, değişken, ve aritmetik yorumlama ile komut ikamesi olarak aynı zamanda uygulanır.

Sadece, kaşlı ayraçların yorumlanması, sözcüklere ayırma ve dosyaisminin yorumlanmasında yorumlanan sözcük sayısı değişebilir. Diğer yorumlamalarda her sözcük tek sözcük olarak yorumlanır. Burada sadece `"$@"` ve `"${isim[@]}"` yorumlaması ayrıcalıklıdır (*Özel Parametreler* (sayfa: 21) ve *Diziler* (sayfa: 76) bölümlerine bakınız).

Tüm yorumlamalardan sonra `tırnakların kaldırılması` uygulanır (*Tırnak kaldırma* (sayfa: 29) bölümüne bakınız).

5.1. Kaşlı Ayraçların Yorumlanması

Kaşlı ayraç yorumlaması, keyfi dizgeler üretmek amacıyla kullanılan bir mekanizma sağlar. Bu mekanizma *Dosyaismi Yorumlaması* (sayfa: 27)na benzer, ancak üretilen dizgeler mevcut dosya isimleri olmak zorunda değildir. Kaşlı ayraç yorumlamasına esas alınacak kalıplar, isteğe bağlı bir *öndizge* ile başlar, kaşlı ayraçlar içine alınmış virgül ayraçlı dizgelerden veya bir dizilim ifadesinden sonra gelen yine isteğe bağlı bir *sondizge* ile biter. Yorumlama daima soldan sağa doğru yapılır.

Kaşlı ayraç yorumlaması iç içe olabilir. Dizgelerin yorumlanması soldan sağa yapılması dışında bir sıraya bağlı değildir. Örneğin:

```
bash$ echo a{d,c,b}e
ade ace abe
```

Bir dizilim ifadesi

$$\{x..y\}$$

biçimindedir, burada x ve y ya tamsayılardır ya da tek karakterlik dizgelerdir. Tamsayı olarak verildiklerinde, ifade x ve y (dahil) arasındaki sayılara genişletilir. Karakter olarak verildiklerinde ise ifade x ve y (dahil) arasındaki alfabetik sırada harflere genişletilir. x ve y değerlerinin aynı türde olması gerektiğine dikkat ediniz.

Kaşlı ayraç yorumlaması diğer bütün yorumlamalardan önce yapılır ve diğer yorumlamalara özel karakterlerin hepsi korunur. Yorumlama tamamen metinseldir. Bash yorumlanan kalıba ya da parantezler içindeki metne herhangi bir sözdizimsel yorumlama uygulamaz. Parametre yorumlaması ile karışmaması için `${` dizgesi kaşlı ayraç yorumlamasında işlenmez.

Kurallara uygun oluşturulmuş bir kaşlı ayraç yorumlamasında, kaşlı ayraçlar tırnaklar arasında olmamalı ve en azından bir tırnaksız virgül veya geçerli bir dizilim ifadesi bulunmalıdır. Aksi takdirde yorumlama uygulanmaz, verilen kalıp değişmeden kalır.

Bir **`{`** veya **`,`** bir kaşlı ayraç ifadesinin parçası olarak yorumlanmamaları için bir tersbölü karakteri ile öncelenebilir. Parametre yorumlaması ile çelişmekten kaçınmak için **`${`** dizgesi kaşlı ayraç yorumlaması için seçilebilir varsayılmaz.

Bu yapı genellikle yukarıdaki örnektekinden daha uzun ve aynı öneke sahip çok sayıda dizge olduğunda benzer şeyleri defalarca yazmamak için bir kısayol olarak kullanılır:

```
mkdir /usr/local/src/bash/{old,new,dist,bugs}
```

veya

```
chown root /usr/{ucb/{ex,edit},lib/{ex?.?*,how_ex}}
```

5.2. Yaklaşık (~) Yorumlaması

Bir sözcük bir tırnaksız yaklaşık (~) karakteri ile başlıyorsa ve tırnaksız ilk / karakterine kadar olan tüm karakterler (veya tırnaksız bir / yoksa tüm karakterler) bir *yaklaşık–öneki* kabul edilir. Yaklaşık–öneki içindeki karakterlerin hiçbiri tırnaklı değilse, yaklaşık–öneki içindeki karakterlerin olası bir kullanıcı ismi oluşturduğu varsayılır. Eğer bu kullanıcı ismi boşsa yaklaşık işareti HOME kabuk değişkeninin değeri ile değiştirilir. Eğer HOME değişkeni belirlenmemişse, yerine kabuğu çalıştıran kullanıcının ev dizini kullanılır. Aksi takdirde yaklaşık–öneki, belirtilen kullanıcının ev dizini ile değiştirilir.

Yaklaşık–öneki ~+ ise, yaklaşık–öneki PWD kabuk değişkeninin değeri ile değiştirilir. Yaklaşık–öneki ~- ise, varsa OLDPWD kabuk değişkeninin değeri kullanılır.

Yaklaşık–önekinden sonra istemlik olarak bir + veya - işareti konmuş bir sayı geliyorsa, yaklaşık–öneki, **dirs** (sayfa: 77) yerleşiğine bu işaret ve sayı verilerek çağrıldığı durumda dönen dizin yığınındaki ilgili eleman ile değiştirilir. Eğer yaklaşık–öneki ile birlikte verilen sayı bir + veya - işareti içermiyorsa + işareti verilmiş kabul edilir.

Kullanıcı ismi geçersizse ya da yaklaşık yorumlaması başarısız olursa sözcük değişmeden kalır.

Her değişken ataması bir : veya ilk = işaretinden önce yaklaşık işaretinin gelebileceği varsayımıyla kontrol edilir. Bu gibi durumlarda, yaklaşık yorumlaması ayrıca uygulanır. Sonuç olarak, PATH, MAILPATH ve CDPATH değişkenlerine atamalarda yaklaşık işaretli dosya isimleri kullanılabilir ve kabuk değer olarak yorumlanan değeri atar.

Aşağıda tırnaksız yaklaşık–öneklerinin nasıl yorumlandığı gösterilmiştir.

`~`
: $HOME değişkeninin değeri

`~/foo`
: $HOME/foo

`~fred/foo`
: fred isimli kullanıcının ev dizininin foo alt dizini

`~+/foo`
: $PWD/foo

`~-/foo`
: ${OLDPWD-'~-'}/foo

`~N`
: **dirs** +N komutundan elde edilen dizge

`~+N`
: **dirs** +N komutundan elde edilen dizge

`~-N`
: **dirs** -N komutundan elde edilen dizge

5.3. Kabuk Parametrelerinin Yorumlaması

Parametre ve *aritmetik yorumlamaları* (sayfa: 27) ile *komut ikameleri* (sayfa: 26)ni $ karakteri başlatır. Yorumlanacak sembol ya da parametre ismi isteğe bağlı olarak, ismin parçası olarak yorumlanacak karakterlerden oluşan değişkeni korumak üzere kaşlı ayraçlar içine alınabilir.

Kaşlı ayraçlar kullanıldığında, parantezi kapatan ilk } bir tırnaklı dizge içinde ya da tersbölü öncelemeli olmamalı ve bir gömülü *aritmetik yorumlama* (sayfa: 27), *parametre yorumlaması* (sayfa: 24) veya *komut ikamesi* (sayfa: 26) içinde olmamalıdır.

Parametre yorumlaması uygulanacak temel kalıp `${`*parametre*`}` şeklindedir. *parametre* yerine değeri kullanılır. *parametre*, tek haneden geniş bir sayı içeren bir konumsal parametre ise ya da *parametre*den sonraki karakter, parametre isminin bir parçası olarak yorumlanmayacaksa kaşlı ayraçlar zorunludur.

parametre'nin ilk karakteri bir ünlem ise, bir seviyelik dolaylı değişken yönlendirmesi yapılır. Bash, değişken ismi olarak *parametre*'nin kalanından şekillendirilen değişkenin değerini kullanır; bu değişken sonradan yorumlanır ve değeri, yorumlamanın geri kalanında *parametre*'nin değeri olarak kullanılır. Bu işlem *dolaylı yorumlama* olarak bilinir. Bunun istisnası aşağıda açıklanan `${!`*önek*`*}` ve `${!`*isim*`[@]}` yorumlamasıdır. Ünlem işareti sol kaşlı ayraçtan hemen sonra gelmelidir ki dolaylılıktan bahsedilebilsin.

Aşağıdaki durumların herbiri için *sözcük*, *aritmetik yorumlama* (sayfa: 27), *yaklaşık yorumlaması* (sayfa: 23) ve *komut ikamesi* (sayfa: 26)ne konu olur.

Altdizge yorumlaması uygulanamadığında, Bash bir parametrenin boş olması ya da atanmamış olması durumlarının varlığına bakar; bu durumda iki noktanın olmaması bir parametrenin sadece atanmamış olmasına bakılması ile sonuçlanır. İki noktanın olması durumunda işleç her iki durumun mevcudiyeti yanında değerin boş olmamasına da bakar; iki nokta olmadığında işleç sadece mevcudiyete bakar.

${*parametre***:-***sözcük***}**
: *parametre* atanmamışsa ya da boşsa, *sözcük* açılımı kullanılır. Aksi takdirde, *parametre* değeri kullanılır.

${*parametre***:=***sözcük***}**
: *parametre* atanmamışsa ya da boşsa, önce *parametre* ye *sözcük* açılımı atanır sonra da *parametre* nin değeri kullanılır. Konumsal parametreler ve özel parametreler bu yolla atanamayabilir.

${*parametre***:?***sözcük***}**
: *parametre* atanmamışsa ya da boşsa, *sözcük* açılımı (ya da *sözcük* verilmemişse olmadığını belirten bir ileti) standart hataya yazılır ve kabuk, etkileşimsizse çıkar. Aksi takdirde, *parametre* değeri kullanılır.

${*parametre***:+***sözcük***}**
: *parametre* atanmamışsa ya da boşsa, hiçbir şey kullanılmaz, aksi takdirde yerine *sözcük* açılımı kullanılır.

${*parametre***:***konum***}**
${*parametre***:***konum***:***uzunluk***}**
: *parametre* içerisinde *konum* dan başlayan *uzunluk* karakterlik altdizge açılır. *uzunluk* verilmemişse, *konum* dan başlayan ve sona kadar giden altdizge açılır. *konum* ve *uzunluk* aritmetik ifadelerdir (*Kabuk Aritmetiği* (sayfa: 74) bölümüne bakınız). Bu *Altdizge Açılımı* olarak bilinir.

  uzunluk sıfır ya da pozitif bir sayıya karşılık olmalıdır. *konum* negatif bir sayıya karşılıksa değeri, *parametre* değerinin sonundan itibaren sayılır. *parametre* `@` ise, sonuç, *konum* dan başlayan *uzunluk* konumsal parametredir. *parametre* `@` veya `*` ile indislenmiş bir dizi ismi ise, sonuç, **${***parametre***[***konum***]}** ile başlayan *uzunluk* üyeli bir dizidir. Bir negatif *konum* belirtilen dizinin en büyük indisinden bir büyük indise göre ele alınır. **:-** yorumlaması ile karışmaması için bir negatif *konum* **:** karakterinden en az bir boşluk ile ayrılmalıdır.Altdizge indislemesi konumsal parametreler kullanılmadıkça sıfırdan başlar, konumsal parametrelerin varlığı halinde ise birden başlar.

${!*önek****}**
${!*önek***@}**
: *önek* ile başlayan isimler, `IFS` özel değişkeninin ilk karakteri ile ayrılan değişken isimleri olarak yorumlanır.

${!*isim***[*]}**
${!*isim***[@]}**
: *isim* bir dizi değişkeni ise dizi indislerinin bir listesi *isim*'e atanır. *isim* bir dizi değilse, *isim* tanımlıysa 0'a değilse null'a genişletilir. **@** kullanılmışsa ve yorumlama çift tırnaklar içinde görünüyorsa her indis ayrı bir sözcüğe genişletilir.

${#*parametre***}**

parametre nin açılım değerindeki karakter sayısını verir. *parametre* `*` veya `@` ise değer, konumsal parametrelerin sayısıdır. *parametre*, `@` veya `*` ile indislenmiş bir dizi ismi ise, değer, dizideki üye sayısıdır.

${*parametre***#***sözcük***}**
${*parametre***##***sözcük***}**

sözcük dosyaismi açılımı (sayfa: 27)ndaki gibi bir kalıp üretecek şekilde yorumlanır. Eğer kalıp, *parametre* nin yorum değerinin başlangıcı ile eşleşirse, yorumlama sonucu olarak, *parametre*'nin yorum sonucundan en kısa eşleşme kalıbı (`#` durumu) veya en uzun eşleşme kalıbı (`##` durumu) silinerek kalan değer atanır. Eğer *parametre* `@` ya da `*` ise kalıp silme işlemi sıra ile konumsal parametrelerin her birine uygulanır ve yorumlama değeri sonuçlanan liste olur. Eğer *parametre*, `@` ya da `*` ile indislenmiş bir dizi değişkeni ise, kalıp silme işlemi sıra ile dizi üyelerinin her birine uygulanır ve yorumlama değeri sonuçlanan liste olur.

${*parametre***%***sözcük***}**
${*parametre***%%***sözcük***}**

sözcük dosyaismi yorumlamasındaki gibi bir kalıp üretecek şekilde yorumlanır. Eğer kalıp, *parametre* nin yorum değerinin son bölümü ile eşleşirse, yorumlama sonucu, silinen en kısa eşleşme kalıplı (`#` durumu) ya da en uzun eşleşme kalıplı (`##` durumu) *parametre* nin yorum değeridir. Eğer *parametre* `@` ya da `*` ise kalıp silme işlemi sıra ile konumsal parametrelerin her birine uygulanır ve yorumlama değeri sonuçlanan liste olur. Eğer *parametre*, `@` ya da `*` ile indislenmiş bir dizi değişkeni ise, kalıp silme işlemi sıra ile dizi üyelerinin her birine uygulanır ve yorumlama değeri sonuçlanan liste olur.

${*parametre***/***kalıp***/***dizge***}**

kalıp dosyaismi yorumlamasındaki gibi bir kalıp üretmek üzere yorumlanır. *parametre* yorumlanır ve onun değerine karşılık olan en uzun *kalıp* eşleşmesi *dizge* ile değiştirilir. Eğer *kalıp* bir `/` imi ile başlıyorsa tüm *kalıp* eşleşmeleri *dizge* ile değiştirilir. Normalde sadece ilk eşleşme değiştirilir. Eğer *kalıp*, `#` ile başlıyorsa *parametre*nin yorumlanan değerinin başlangıcında eşleştirilmelidir. Eğer *kalıp*, `%` ile başlıyorsa *parametre*nin yorumlanan değerinin sonunda eşleştirilmelidir. Eğer *dizge* boşsa*kalıp* eşleşmeleri silinir ve kalıp'tan sonraki `/` işareti atlanabilir. Eğer *parametre*, `@` veya `*` ise ikame işlemi sıra ile her konumsal parametreye uygulanır ve sonuç bir liste olur. Eğer *parametre*, `@` veya `*` ile indislenmiş bir dizi değişkeni ise ikame işlemi sırayla dizinin her üyesine uygulanır ve sonuç bir liste olur.

5.4. Komut İkamesi

Komut ikamesi ile bir komutun çıktısının komutun kendisi ile değiştirilmesi sağlanır. Komut ikamesi aşağıdaki dizilimlerle karşılaşıldığında uygulanır:

`$(`*komut*`)`

or

komut`` ` ``

Bash bu dizilimlerin sonundaki satırsonu karakterini silip, içindeki *komut*'u önce standart çıktıda çalıştırır ve komutun çıktısını onu argüman olarak kullanan komutun standart girdisine yerletirir. Gömülü satırsonları silinmez, ancak sözcük ayrımlaması sırasında silinmiş olabilirler. `$(cat` *file*`)` komut ikamesi daha hızlı olan eşdeğeri, `$(<` *file*`)` ile değiştirilebilir. Örneğin `echo 'cat dosya'` komutunu kabukta çalıştırırsanız `dosya`'nın içeriği standart çıktıya dökümlenir.

Eski tarz olan sola yatık tektırnaklı biçem kullanıldığında tersbölü işareti `$`, `` ` `` veya `\` karakterlerini öncelemek dışında anlamını korur. Bir tersbölü ile öncelenmemiş ilk sola yatık tek tırnak komut ikamesini sonlandırır. `$(`*komut*`)` biçemi kullanıldığında parantezler arasındaki herşey komut olarak ele alınır ve hiçbir şey özel olarak anlamlandırılmaz.

İç içe komut ikameleri kullanılabilir. Sola yatık tektırnaklı iç içeliklerde içteki tırnaklar tersbölü ile öncelenmelidir.

Yorumlanan alandaki karakterler çift tırnaklar arasına alınmışsa sözcük ayrımlaması ve dosyaismi yorumlaması uygulanmaz.

5.5. Aritmetik Yorumlama

Aritmetik yorumlama, aritmetik ifadelerin sonuçlarının kullanılmasını sağlar. Aritmetik yorumlamaya konu sözdizimi aşağıdaki gibidir:

`$(( ` *ifade* ` ))`

ifade çift tırnaklar içine alınmış gibi yorumlanır, ancak parantezleriçinde yer alan bir çift tırnak özel bir şekilde ele alınmaz. *ifade* içindeki tüm dizgeciklere *parametre yorumlaması* (sayfa: 24) ve *komut ikamesi* (sayfa: 26) ile *sözcüklere ayırma* (sayfa: 27) uygulanır. Aritmetik yorumları iç içe olabilir.

İfadenin değerlendirilmesi *Kabuk Aritmetiği* (sayfa: 74) bölümünde listelenen kurallar çerçevesinde yapılır. İfade geçersizse, Bash standart çıktıya başarısızlık ile ilgili bir ileti gönderir ve ikame gerçekleşmez.

5.6. Süreç İkamesi

Süreç ikamesi, isimli veri hatları (FIFO'lar) ya da açık dosyaları isimlendirme yöntemi olan `/dev/fd` yönteminin desteklendiği sistemlerde kullanılır. Yoruma konu olacak sözdizimi aşağıdaki gibidir:

`<(`*liste*`)`

veya

`>(`*liste*`)`

liste içindeki süreçler girdi ya da çıktıları bir FIFO ya da `/dev/fd` içindeki bir dosyaya bağlanarak çalıştırılır. Çalıştırılan komuta, bu dosyanın ismi, açılım sonucunda argüman olarak aktarılır. `>(`*liste*`)` biçemi kullanıldığında, *liste* için girdi, ilgili dosyaya yazıldığı zaman verilir. `<(`*liste*`)` biçemi kullanıldığında ise, *liste* nin çıktısı, ilgili dosya okunarak elde edilir. `<` veya `>` işareti ile parantez arasında boşluk bırakılmamalıdır. Boşluk bırakılırsa yapı, bir süreç ikamesi değil bir yönlendirme olarak yorumlanacaktır.

Mümkün olduğunca süreç ikamesi; *aritmetik yorumlama* (sayfa: 27), *parametre ve değişken yorumlaması* (sayfa: 24) ve *aritmetik yorumlama* (sayfa: 27) ile aynı anda uygulanır.

5.7. Sözcüklere Ayırma

Kabuk, sözcüklere ayırmak için çift tırnaklar arasında olmayan *parametre yorumlaması* (sayfa: 24) ve *aritmetik yorumlamaları* (sayfa: 27) ile *komut ikameleri* (sayfa: 26)nin sonuçlarını tarar.

Kabuk `$IFS` değişkenindeki her karakteri bir ayraç olarak ele alır ve diğer yorumlamaların sonuçlarını sözcüklere ayırmak için bu karakterleri kullanır. `IFS` değişkeni atanmamışsa, onun değeri yerine öntanımlı olarak `<space><tab><newline>` dizgesi kullanılır. `IFS` değişkeni bu öntanımlı değer dışında bir değerle atanmışsa, sözcüğün başındaki ve sonundaki `boşluk` ve `sekme` karakterleri bu boşluk karakterlerinden biri `IFS` değeri içinde olduğu sürece yoksayılır. `IFS` boşluk karakterlerinden biri ile birlikte verilen `IFS` boşluk karakteri olmayan herhangi bir karakter bir alanı ayırmakta kullanılabilir. `IFS` boşluk karakterlerinden oluşan bir dizi de bir ayraç olarak kullanılabilir. `IFS` değişkeni boş olarak atanmışsa sözcüklere ayırma işlemi yapılmaz.

Apaçık boş argümanlar ("" veya '') değişmeden kalır. Değer üretmeyen parametre yorumlamalarının sonucu olan tırnaksız boş argümanlar kaldırılır. Değeri olamayan bir parametrenin yorumunun sonucundaki çift tırnaklı bir boş argüman değişmeden kalır.

Yorumlama yapılamamışsa sözcük ayrımlama da uygulanmaz.

5.8. Dosyaismi Yorumlaması

Sözcük ayrımlaması yapıldıktan sonra `-f` seçeneği verilmedikçe (*set Yerleşiği* (sayfa: 49) bölümüne bakınız.), Bash her sözcükte `*`, `?` ve `[` karakterlerini arar. Bunlardan biri varsa, sözcük bir *kalıp* olarak ele alınır ve sözcük, kalıp ile eşleşen dosya isimlerinin alfabetik sıralı bir listesi ile değiştirilir. Bir eşleşme bulunamamışsa ve `nullglob` kabuk seçeneği iptal edilmişse, sözcük değişmeden kalır. `nullglob` seçeneği etkin ise ve bir eşleşme bulunamamışsa sözcük silinir. `failglob` kabuk seçeneği etkin ise ve bir eşleşme bulunamamışsa bir hata iletisi basılır ve komut çalıştırılmaz. `nocaseglob` kabuk seçeneği etkin ise eşleşme alfabetik karakterlerin harf büyüklüklerine bakılmaksızın uygulanır.

Bir kalıp dosya ismi üretimi için kullanılmışsa ve `dotglob` kabuk seçeneği etkin değilse, bir dosya isminin başındaki `.` karakteri veya hemen ardından gelen bir / doğrudan eşleştirilmelidir. Bir dosya ismi eşleşmesinde / karakteri daima doğrudan eşleşmiş olmalıdır. Diğer durumlarda `.` karakterine bir özellik atfedilmez.

`nocaseglob`, `nullglob`, `failglob` ve `dotglob` seçeneklerinin açıklamaları için **`shopt`** *yerleşik komutunun açıklamalarına bakınız* (sayfa: 52).

`GLOBIGNORE` kabuk değişkeni bir kalıpla eşleşen dosya isimleri kümesini sınırlamakta kullanılabilir. `GLOBIGNORE` ile bir kalıp atanmışsa, dosya isimleri kümesi içindeki dosya isimlerinden biri aynı zamanda bu kalıpla da eşleşiyorsa o dosya ismi listeden çıkarılır. `GLOBIGNORE` atandığında ve null olmadığında `.` ve `..` dosya isimleri daima yoksayılır. Bununla birlikte `GLOBIGNORE` değişkenine null olmayan bir değerin atanması, `dotglob` kabuk seçeneğini etkinleştirilmesi gibi bir etkiye de sahiptir, bu bakımdan `.` ile başlayan tüm dosya isimleri eşlenecektir. `.` ile başlayan dosya isimlerinim yoksayılmasını sağlayan eski davranışa dönmek için `GLOBIGNORE` değişkenine atanan kalıplardan birini `.*` yapın. `GLOBIGNORE` değişkeni kaldırıldığında, `dotglob` seçeneği de etkin olmaktan çıkarılır.

5.8.1. Kalıp Eşleme

Aşağıda anlamları açıklanan özel kalıp karakterleri dışındaki tüm kalıp karakterleri kendisi ile eşleşir. Bir kalıp içinde boş karakter ('\0') bulunmamalıdır. Bir karakterin öncesine bir tersbölü gelmişse eşleşme sırasında önceleyen yoksayılır. Özel kalıp karakterlerinin normal karakterler olarak davranması istenirse tırnak içine alınmalıdırlar.

Özel kalıp karakterlerinin anlamları:

`*`
: Boş dizge dahil herhangi bir dizge eşleşir.

`?`
: Herhangi bir tek karakter eşleşir.

`[...]`
: Parantez içindeki karakterlerden herhangi biri eşleşir. Aralarında bir tire işareti ile belirtilen bir karakter çiftinin oluşturduğu *aralık ifadesi*ndeki aralığa düşen karakterlerin herbiri eşleşir. Bu aralık, yerele özgü karakter kodlamasına uyan bir aralıktaki karakterlerden oluşur. İlk `[` parantezinden hemen sonra bir `!` veya `^` karakteri geliyorsa bu karakterden sonraki karakterlerin dışında kalan karakterler eşleşir. `-` karakteri sadece enbaşta ya da en sonda ise kendisi ile eşleşir. `]` karakteri ise ilk karakter olursa kendisi ile eşleşebilir. Aralık ifadesindeki karakterlerin sırası `LC_COLLATE` kabuk değişkenindeki yerele göre oluşturulur.

  Örneğin, öntanımlı C yerelinde, `[a-dv-z]` ifadesi `[abcdvwxyz]` ifadesine eşgeğerdir. Birçok yerel, karakterleri sözlük sıralamasında sıralar ve bu yerellerde `[abcdvwxyz]` ifadesi yerine örneğin `[aBbCcDdvVwWxXyYz]` ifadesi eşdeğer olabilir. Türkçe için bu ifade `[abcçdvyz]` ile eşdeğerdir.

`[` ile `]` arasında karakter sınıfları `[:`*sınıf*`:]` sözdizimi ile belirtilir. Buradaki *sınıf* POSIX standardında tanımlanan aşağıdaki sınıflardan biri olabilir:

```
alnum   alpha   ascii   blank   cntrl   digit   graph   lower
print   punct   space   upper   word    xdigit
```

Eşleşme karakter sınıfındaki herhangi bir karakter için oluşur. **word** karakter sınıfı harflerle, rakamlarla ve _ karakteri ile eşleşir.

`[` ile `]` arasında bir eşdeğer sınıf `[=`*c*`=]` sözdizimi ile belirtilebilir. Buradaki eşleşme yerele göre *c* karakteri ile aynı karşılaştırma ağırlığındaki karakterlere uygulanır.

`[` ile `]` arasında `[.`*sembol*`.]` sözdizimi ile *sembol* karşılaştırma sembolü eşleştirilebilir.

`extglob` kabuk seçeneği **shopt** yerleşiğini kullanarak etkinleştirilirse bazı ek kalıp eşleşme işleçleri kullanılabilir. Aşağıdaki açıklamalarda geçen *kalıp–listesi*, `|` karakteri ayraç olarak kullanılmış bir ya da daha fazla kalıptan oluşmuş bir listeyi ifade etmektedir. Birleşik kalıplar aşağıdaki alt kalıplardan biri ya da bir kaçı ile oluşturulabilir:

?(*kalıp–listesi***)**
: Belirtilen kalıpların sıfır ya da biri eşlenir.

(kalıp–listesi***)**
: Belirtilen kalıpların sıfır ya da daha fazlası eşlenir.

+(*kalıp–listesi***)**
: Belirtilen kalıpların bir ya da daha fazlası eşlenir.

@(*kalıp–listesi***)**
: Belirtilen kalıpların biri eşlenir.

!(*kalıp–listesi***)**
: Belirtilen kalıpların biri dışındaki herşey eşlenir.

5.9. Tırnak kaldırma

Yorumlamalar uygulandıktan sonra, yukarıdaki yorumlamaların sonucu olmayan ve tırnak içine alınmamış tüm `\`, `'` ve `"` karakterleri kaldırılır.

6. Yönlendirmeler

Bir komut çalıştırılmadan önce, kabuk tarafından yorumlanacak özel bir yazım şekli kullanılarak girdisi ve çıktısı yönlendirilebilir. Yönlendirme, kabuğun çalıştırma ortamında kullanılmak üzere dosyaların açılması ve kapatılması için de kullanılabilir. Aşağıdaki yönlendirme işleçleri basit bir komutun öncesinde veya sonrasında ya da içinde kullanılabilir. Yönlendirmeler yazıldıkları sırada yani soldan sağa yorumlanır.

Aşağıdaki açıklamalarda, dosya tanıtıcı numara verilmemişse ve yönlendirme işlecinin ilk karakteri `<` ise yönlendirme standart girdiden (dosya tanım numarası 0), işleç `>` ise standart çıktıya yapılır (dosya tanım numarası 1).

Aşağıdaki açıklamalarda yönlendirme işlecinin bir sözcükten sonra geldiği durumlarda, aksi belirtilmedikçe, sözcüğe *kaşlı ayraç yorumlaması* (sayfa: 23), *yaklaşık yorumlaması* (sayfa: 23), *parametre yorumlaması* (sayfa: 24), *komut ikamesi* (sayfa: 26), *aritmetik yorumlama* (sayfa: 27), *tırnak kaldırma* (sayfa: 29), *sözcüklere ayırma* (sayfa: 27) ve *dosyaismi yorumlaması* (sayfa: 27) uygulanır. Sözcüğün yorumlama sonucu bir sözcükten fazlası ile sonuçlanırsa, Bash bir hata üretir.

Yönlendirmelerin sırası önemlidir. Örneğin,

```
ls > dizin.liste 2>&1
```

komutunda standart girdi (dosya tanıtıcı 1) ve standart hata (dosya tanıtıcı 2) birlikte *dizin.liste* dosyasına yönlendirilirken,

```
ls 2>&1 > dizin.liste
```

komutunda standart çıktı *dizin.liste* dosyasına yönlendirilir, çünkü standart hata, *dizin.liste* dosyasına yönlendirilmeden önce standart çıktıya kopyalanmıştır.

Bash aşağıda açıklandığı gibi çeşitli dosya isimlerini yönlendirmelerin içinde özel olarak ele alır:

/dev/fd/*fd*
: *fd* bir tamsayı ise dosya tanıtıcı *fd* kopyalanır.

/dev/stdin
: Dosya tanıtıcı 0 kopyalanır. (Standart girdi)

/dev/stdout
: Dosya tanıtıcı 1 kopyalanır. (Standart çıktı)

/dev/stderr
: Dosya tanıtıcı 2 kopyalanır. (Standart hata)

/dev/tcp/*konak/port*
: *konak* geçerli bir konak ismi ya da Internet adresi ise ve *port* bir tamsayı port numarası ya da servis ismi ise Bash, ilgili sokete bir TCP bağlantısı açmaya çalışır.

/dev/udp/*konak/port*
: *konak* geçerli bir konak ismi ya da Internet adresi ise ve *port* bir tamsayı port numarası ya da servis ismi ise Bash, ilgili sokete bir UDP bağlantısı açmaya çalışır.

Bir dosyanın açılması veya oluşturulması sırasında bir hata oluşursa, yönlendirme gerçekleşmez.

9'dan büyük dosya tanıtıcıları kullanılarak yapılan yönlendirmelerde dosya tanıtıcıların kabuğun dahili olarak kullandıkları ile çakışabileceği dikkate alınmalıdır.

6.1. Girdi Yönlendirmesi

Girdi yönlendirmesinde, verilen *sözcük*'ün yorumlanması ile elde edilen isme sahip dosya, verilmişse dosya tanıtıcı *n* üstünde, aksi takdirde standart girdide (dosya tanıtıcı 0) açılır.

Girdi yönlendirmesi olarak değerlendirilecek sözdizimi:

[*n*]<*sözcük*

6.2. Çıktı Yönlendirmesi

Çıktı yönlendirmesinde, verilmişse dosya tanıtıcı *n*, aksi takdirde standart çıktı (dosya tanıtıcı 1), verilen *sözcük*'ün yorumlanması ile elde edilen isme sahip dosyaya yazılır. Dosya yoksa oluşturulur, varsa önce dosyanın içeriği silinir.

Çıktı yönlendirmesi olarak değerlendirilecek sözdizimi:

[*n*]>[|]*sözcük*

Yönlendirme işleci > ise, noclobber seçeneği **set** yerleşiği ile etkinleştirilmişse ve *sözcük*'ün yorumlanması ile elde edilen isme sahip dosya varsa ve bir normal dosya ise yönlendirme gerçekleşmez. Yönlendirme işleci > ya da >| ise ve noclobber seçeneği etkin değilse dosya varsa bile yönlendirme yapılmaya çalışılır.

6.3. Yönlendirilmiş Çıktının Eklenmesi

Yönlendirilmiş çıktının eklenmesi işleminde, verilmişse dosya tanıtıcı *n*, aksi takdirde standart çıktı (dosya tanıtıcı 1), verilen *sözcük*'ün yorumlanması ise elde edilen isme sahip dosyaya eklenir. Dosya yoksa oluşturulur.

Çıktının eklenmesi olarak değerlendirilecek sözdizimi:

```
[n]>>sözcük
```

6.4. Standart Çıktı ve Standart Hatanın Yönlendirmesi

Bash, hem standart çıktı (dosya tanıtıcı 1) hem de standart hatayı (dosya tanıtıcı 2), *sözcük*'ün yorumlanması ile elde edilen isme sahip dosyaya yönlendirebilir.

Standart çıktı ve standart hatanın yönlendirilmesine konu iki sözdizimi vardır:

```
&>sözcük
```

ve

```
>&sözcük
```

Bu iki biçemden birincisinin kullanılması önerilir. Her ikisi de aşağıdakine eşdeğerdir:

```
>sözcük 2>&1
```

6.5. Uzun Metinlerin Yönlendirilmesi

Bu tür yönlendirmede kabuk, sadece *sözcük*'ün bulunduğu satıra gelene kadar olan tüm satırları okur.

Uzun Metinlerin Yönlendirilmesine konu sözdizimi:

```
<<[-]sözcük
        Çok satırlı metin
sonlandırıcı
```

sözcük üzerinde parametre, komut, aritmetik ve dosyaismi yorumlamaları uygulanmaz.

sözcük tırnak içine alınmışsa, *sonlandırıcı*, *sözcük*'ün tırnaklar kaldırılmış halidir ve *Çok satırlı metin* üzerinde herhangi bir yorumlama işlemi uygulanmaz.

sözcük tırnak içine alınmamışsa, *Çok satırlı metin* üzerinde parametre, komut ve aritmetik yorumlar uygulanır, yani metin içindeki \, $ ve ` karakterleri \ ile öncelenmiş olmalıdır, ayrıca \satırsonu çiftleri de yoksayılır (yani bu satırlardan sonraki satırlar satırın devamı olarak ele alınır).

Yönlendirme işleci <<- ise, *Çok satırlı metin* ve *sonlandırıcı* içindeki sekme karakterleri ayrılır, böylece kabuk betikleri içindeki kullanımlarda metnin olduğu gibi ele alınması mümkün olur.

6.6. Dizgelerin Yönlendirilmesi

Metin yönlendirmenin bir başka biçimi:

```
<<< sözcük
```

sözcük yorumlanır ve standart girdideki komuta arzedilir.

6.7. Dosya Tanıtıcılarının Kopyalanması

Yönlendirme işleci

[n]<&sözcük

girdi dosya tanıtıcılarını kopyalamakta kullanılır. *n* ile belirtilen dosya tanıtıcı, *sözcük*'ün yorumlanması ile elde edilen sayı numaralı dosya tanıtıcıya kopyalanır. *sözcük* ile belirtilen sayı bir dosya tanıtıcısına karşılık değilse yönlendirme hatası oluşur. *sözcük*, - olarak yorumlanmışsa, *n* dosya tanıtıcısı kapatılır. *n* verilmezse standart girdi (dosya tanıtıcısı 0) kullanılır.

Yönlendirme işleci

[n]>&sözcük

çıktı dosya tanıtıcılarını kopyalamak anlamında kullanılır. *n* verilmezse standart çıktı (dosya tanıtıcısı 1) kullanılır. *sözcük* ile belirtilen sayı bir dosya tanıtıcısına karşılık değilse yönlendirme hatası oluşur. Özel bir durum olarak, *n* verilmezse ve *sözcük* bir sayı değilse standart çıktı ve standart hata önceden açıklandığı gibi yönlendirilir.

6.8. Dosya Tanıtıcıların Taşınması

Yönlendirme işleci

[n]<&rakam-

dosya tanıtıcı *rakam*'ı dosya tanıtıcı*n*'e, *n* belirtilmemişse standart girdiye (dosya tanıtıcı 0) taşır. *rakam n*'e kopyalandıktan sonra kapatılır.

Benzer şekilde, yönlendirme işleci

[n]>&rakam-

dosya tanıtıcı *rakam*'ı dosya tanıtıcı*n*'e, *n* belirtilmemişse standart çıktıya (dosya tanıtıcı 1) taşır.

6.9. Dosya Tanıtıcılarının Okuma ve Yazma Amacıyla Açılması

Yönlendirme işleci

[n]<>sözcük

sözcük'ün yorumlanması ile elde edilen isme sahip dosyanın hem okuma hem de yazma amacıyla *n* dosya tanıtıcısı üstünde açılmasını sağlar. Eğer dosya yoksa oluşturulur.

7. Komutların Çalıştırılması

7.1. Basit Komut Yorumlaması

Bir basit komut çalıştırıldığı zaman, kabuk aşağıdaki yorumları, atamaları ve yorumlamaları soldan sağa uygular.

1. Çözümleyicinin yönlendirme ve değişken atamaları (bunlar komut isminden öncedir) olarak işaretlediği sözcükler daha sonra işlenmek üzere saklanır.
2. Yönlendirme ve değişken ataması olmayan sözcükler yorumlanır (*Kabuk Yorumları* (sayfa: 22) bölümüne bakınız). Yorumlamadan sonra kalan sözcükler varsa, ilk sözcük komut ismi olarak, kalanlar ise argümanlar olarak ele alınır.
3. Yönlendirmeler, *Yönlendirmeler* (sayfa: 29) bölümünde açıklandığı şekilde uygulanır.

4. Değişken atamalarındaki = işaretlerinden sonra gelen metine *yaklaşık yorumlaması* (sayfa: 23), *parametre yorumlaması* (sayfa: 24), *komut ikamesi* (sayfa: 26), *aritmetik yorumlama* (sayfa: 27) ve *tırnak kaldırma* (sayfa: 29) uygulandıktan sonra sonuç değişkene atanır.

Sonuçlanan bir komut ismi yoksa, değişken atamaları, kullanılan kabukta ortam değişkeni olarak yorumlanır. Aksi takdirde değişkenler sadece çalıştırılan komutun ortamına eklenir ve kabuk ortamına etkisi olmaz. Bir değişkenin salt–okunur değerine atama yapılmaya çalışılırsa bir hata oluşur ve komut sıfırdan farklı bir çıkış durumu ile sonlanır.

Sonuçlanan bir komut ismi yoksa, yönlendirmeler uygulanır, fakat kullanılan kabuk ortamı etkilenmez. Bir yönlendirme hatası oluşur ve komut sıfırdan farklı bir çıkış durumu ile sonlanır.

Yorumlamadan sonra bir dosya ismi kalırsa, sonraki bölümde anlatıldığı gibi çalıştırma gerçekleştirilir, aksi takdirde komut işlemsiz sonlanır. Yorumlamanın sonucu bir komut ikamesi ise komutun çıkış durumu son çalıştırılan komutun çıkış durumudur. Komut ikamesi yoksa, komut sıfır çıkış durumu ile tamamlanır.

7.2. Komutun Bulunması ve Çalıştırılması

Bir komut, sözcüklere ayrıldıktan sonra, sonuçlananlar bir basit komut ve bir istemlik argüman listesi ise aşağıdaki eylemler oluşur:

1. Komutun isminde / işaretleri yoksa kabuk komutun yerini bulmaya çalışır. Bu isimde bir kabuk işlevi varsa, *Kabuk İşlevleri* (sayfa: 20) bölümünde açıklandığı gibi işlev çağrılır.

2. İsim bir işlev ismi değilse, kabuk bu kez de onu kabuk yerleşikleri arasında arar, varsa yerleşik komut çalıştırılır.

3. İsim bir işlev ya da yerleşik komut değilse ve / işaretleri de içermiyorsa, Bash bu isimde bir dosya içeren dizini bulmak üzere `$PATH` ile verilmiş her dizine bakar. Bash, çoklu `PATH` aramaları yapmamak için çalıştırılabilir dosyaların tam dosya yollarını bir tabloda tutar (*hash Yerleşiği* (sayfa: 40) bölümüne bakınız). `$PATH` dizinlerinde arama işlemi, dosya, arama tablosunda yoksa uygulanır. Arama başarısız olursa, kabuk bir hata iletisi gösterir ve 127 çıkış durumu ile döner.

4. Arama başarılı ise veya komut ismi bir ya da daha fazla / içeriyorsa, kabuk bu uygulamayı ayrı bir çalıştırma ortamında çalıştırır. Verilen isme argüman numarası olarak 0 ve varsa komutun argümanlarına 1 den başlayan argüman numaraları atanır.

5. Bu çalıştırma işlemi, dosyanın çalıştırılabilir ve bir dizin olmaması sebebiyle gerçekleşmezse, dosya bir kabuk betiği olarak kabul edilir ve *Kabuk Betikleri* (sayfa: 35) bölümünde açıklandığı gibi çalıştırılır.

6. Komut eşzamanlamasız olarak başlatılamazsa, kabuk komutun işini bitirmesini bekler ve bittiğinde komutun çıkış durumu ile döner.

7.3. Komut Çalıştırma Ortamı

Kabuğun aşağıdaki özelliklere sağlayan bir *çalıştırma ortamı* vardır:

- **`exec`** (sayfa: 39) yerleşiğine yapılan yönlendirmelerle değiştirilmiş olarak, çağrı sırasında kabuktan alınan dosyaları açar.

- Çalışma dizini, **`cd`** (sayfa: 38), **`pushd`** (sayfa: 78) veya **`popd`** (sayfa: 77) yerleşikleri tarafından geçilen ya da çağrı sırasında kabuktan alınan dizine ayarlanır.

- Dosya oluşturma kipi maskesi, ya **`umask`** (sayfa: 43) ile belirlenen ya da kabuğun kendini başlatan kabuktan aldığı değere ayaranır.

- Sinyal kapanları **trap** (sayfa: 42) tarafından belirlenir.
- Kabuk parametreleri, değişken atamalarına göre veya **set** ile ya da kabuğun kendini başlatan kabuktan aldığı parametrelere göre ayarlanır.
- Kabuk işlevlerini çalıştırma sırasında veya kabuğun kendini başlatan kabuktan aldıkları ile tanımlar.
- Seçenekleri çağrı sırasında (ya öntanımlı olarak ya da komut satırı argümanları olarak) veya **set** (sayfa: 49) yerleşiği ile belirler.
- Seçenekleri **shopt** (sayfa: 52) yerleşiği ile etkinleştirir.
- Kabuk takma adlarını **alias** (sayfa: 43) yerleşiğini kullanarak tanımlar (*Takma Adlar* (sayfa: 75) bölümüne bakınız).
- çeşitli süreç kimlikleri, bunların artalan işleri dahil (*Komut Listeleri* (sayfa: 15) bölümüne bakınız), `$$` değeri ve `$PPID` değeri

Bir kabuk işlevi ya da kabuk yerleşiği olmayan bir basit komut çalıştırıldığında, aşağıdaki özellikleri sağlayan ayrı bir çalıştırma ortamında çağrılır. Aksi belirtilmedikçe, değerler kabuktan alınır.

- kabukların açık dosyaları artı komuta yönlendirilmiş eklemeler ve değişiklikler
- çalışma dizini
- dosya oluşturma kipi maskesi
- ihraç edilmek üzere işaretli kabuk değişkenleri ve işlevler, komut için ihraç edilmiş değişkenler ile ortamda atanmış değişkenler (*Ortam* (sayfa: 34) bölümüne bakınız)
- kabuk tarafından yakalanan sinyaller, kabuğun kendini çalıştıran kabuktan alınan değerlere sıfırlanır ve kabuk tarafından yoksayılan sinyaller yoksayılır.

Bu ayrı ortamda çağrılan bir komut kabuğun çalıştırma ortamını etkileyemez.

Komut ikamesi, parantezlerle gruplanmış komutlar ve eşzamansız komutlar mevcut kabuk ortamının bir kopyası olan bir alt–kabuk ortamında başlatılır, fakat kabuk tarafından yakalanan sinyaller kabuğun başlangıç esnasında kendini başlatan kabuktan aldığı değerlere atanır. Bir boruhattının parçası olarak çağrılan yerleşik komutlar ayrıca bir altkabuk ortamında çalıştırılır. Altkabuk ortamında yapılan değişiklikler kabuğun çalıştırma ortamını etkileyemez.

Bir komuttan sonra bir `&` geliyorsa ve iş denetimi etkin değilse, komut için öntanımlı standart girdi `/dev/null`, yani boş dosyadır. Aksi takdirde, çağrılan komut, çağrı kabuğunun yönlendirmelerle değiştirilen dosya tanıtıcılarını miras alır.

7.4. Ortam

Bir komut çağrıldığında, ona *ortam* adı verilen bir dizgeler dizisi verilir. Bu, `isim=değer` şeklindeki isim–değer çiftlerinden oluşan bir listedir.

Bash, ortamı değiştirmek için çeşitli yollar sağlar. Çağrı durumunda, kabuk kendi ortamını tarar ve bulduğu her isim için çocuk süreçlere aktarılmak üzere otomatik olarak işaretleyerek bir parametre oluşturur. Çalıştırılan komutlar ortamı miras alır. **export** (sayfa: 39) ve **declare** (sayfa: 45) `-x` komutları, parametrelerin ve işlevlerin ortama eklenmesini ve ortamdan silinmesini sağlar. Ortamdaki bir parametrenin değeri değişirse, yeni değer eskisiyle yer değiştirerek ortamın parçası haline gelir. Çalıştırılan komut tarafından miras alınan ortam kabuğun başlangıç ortamından oluşur. Bu ortam, kabukta değiştirilebilen değerler, eksi **export** `-n` ve **unset** (sayfa: 43) komutları tarafından kaldırılan çiftler, artı **export** ve **declare** `-x` komutları üzerinden eklemelerden oluşur.

Bir basit komut veya işlevin ortamı, *Kabuk Parametreleri* (sayfa: 21) bölümünde açıklandığı gibi parametre atamaları ile öncelenerek geçici olarak büyütülebilir. Bu atama deyimleri sadece komut tarafından görülen ortamı etkiler.

`-k` seçeneği etkinse (*set Yerleşiği* (sayfa: 49) bölümüne bakınız), tüm parametre atamaları komut ismini öncelemek yerine doğrudan komut ortamına yerleştirilir.

Bash bir harici komutu çağırırsa, `$_` değişkeni komutun tam dosyayolunu içerir ve ortam içindeki komuta aktarılır.

7.5. Çıkış Durumu

Kabuğun amaçlarına uygun olarak, sıfır çıkış durumu ile çıkan bir komut başarılıdır. Sıfırdan farklı bir çıkış durumu ise başarısızlık göstergesidir. Böylece sayılardan oluşan hem başarı durumunu hem de başarısızlık nedenlerinin ifade edilebildiği bir yapı sağlanır. Bir komut, numarası N olan bir ölümcül sinyal ile sonlandırılırsa, Bash, çıkış durumu olarak 128+N değerini kullanır.

Bir komut bulunamamışsa, bir çocuk süreç oluşturulur ve bu süreç 127 çıkış durumu ile döner. Bir komut bulunmuş ancak çalıştırılabilir değilse, 126 dönüş durumu döner.

Yönlendirme ya da yorumlama sırasında bir hatadan dolayı komutun çalıştırılması başarısız olursa, sıfırdan büyük bir çıkış durumu döner.

Çıkış durumu, *Koşullu Çalıştırma* (sayfa: 17) deyimleri ve *Komut Listeleri* (sayfa: 15) tarafından kullanılır.

Bash yerleşiklerinin de başarı durumunda sıfır ve başarısızlık durumunda sıfırdan farklı bir çıkış durumu ile dönerek, çıkış durumlarnın koşul ve liste yapılarında kullanılabilmesi sağlanmıştır. Tüm yerleşikler yanlış kullanım halinde 2 çıkış durumu ile döner.

7.6. Sinyaller

Bash etkileşimli ise, bir *sinyal kapanının* (sayfa: 42) yokluğunda, `SIGTERM` sinyalini yoksayarken `SIGINT` sinyalini yakalar ve elde eder (yani **`wait`** yerleşiğine kesme uygulanabilir). `SIGTERM` sinyalinin yoksayılması **`kill 0`** komutu ile bir etkileşimli kabuğun öldürülememesi demektir. Bash bir `SIGINT` sinyali aldığında, çalışan döngülerden çıkılır. Tüm durumlarda, `SIGQUIT` sinyalini yoksayar. Eğer *iş denetimi* (sayfa: 83) etkinse, Bash `SIGTTIN`, `SIGTTOU` ve `SIGTSTP` sinyallerini yoksayar.

Bash tarafından başlatılan yerleşik olmayan komutlar, komutun başlatıldığı kabuğu başlatan kabuktan miras alınan değerleri kullanabilen sinyal yakalayıcılara sahiptir. İş denetimi etkin değilse, eşzamansız komutlar bu miras alınan sinyal yakalayıcılara ek olarak `SIGINT` ve `SIGQUIT` sinyallerini de yoksayar. Komut ikamesinin sonucu olarak çalıştırılan komutlar klavyeden verilen `SIGTTIN`, `SIGTTOU` ve `SIGTSTP` iş denetim sinyallerini yoksayar.

Kabuk öntanımlı olarak bir `SIGHUP` sinyali aldığında çıkar. Çıkmadan önce, bir etkileşimli kabuk altında çalışan ya da durmuş tüm işlere `SIGHUP` sinyali gönderir. Durmuş işler aldıkları `SIGHUP` sinyalini uygulayabilmek için önce bir `SIGCONT` sinyali gönderir. Kabuk bir işe tekrardan bir `SIGHUP` sinyali göndermemek için ya **`disown`** (sayfa: 85) yerleşiğini kullanarak onları iş tablosundan kaldırır ya da **`disown`** `-h` komutunu kullanarak `SIGHUP` sinyalini bir daha almamasını sağlar.

Eğer `huponexit` kabuk seçeneği **`shopt`** (sayfa: 85) yerleşiği ile etkinleştirilmişse ve bir etkileşimli giriş kabuğu çıkarsa, Bash bütün işlere bir `SIGHUP` sinyali gönderir.

Bash bir komutun sona ermesini beklerken bir sinyal kapanının çalıştırılması için bir sinyal alırsa, komut işini tamamlayana kadar *sinyal kapanı* (sayfa: 42) çalıştırılmaz. Bash **`wait`** yerleşiği üzerinden bir eşzamansız komut için beklerken bir sinyal kapanının çalıştırılması için bir sinyal alırsa, **`wait`** yerleşiği 128 den büyük bir çıkış durumu ile döner ve ardından sinyal kapanı çalıştırılır.

8. Kabuk Betikleri

Bir kabuk betiği kabuk komutlarını içeren bir metin dosyasıdır. Bash çağrılırken böyle bir dosya seçenek olmayan ilk argüman olarak verilmişse ve seçenekler arasında `-c` ve `-s` seçenekleri yoksa (*Bash'in Çağrılması* (sayfa: 67) bölümüne bakın), Bash dosyadan okuduğu komutları çalıştırır ve çıkar. Bu tür bir işlemde kullanılan kabuk etkileşimsiz kabuktur. Kabuk dosyayı önce çalıştırıldığı dizinin içinde arar, bulamazsa `$PATH` içindeki dizinlere bakar.

Bash bir kabuk betiğini çalıştırırken, özel parametre `0`'ı kabuğun ismine değil dosya ismine atar ve diğer konumsal parametreler de varsa argümanlara atanır. Olmayan argümanların konumsal parametreleri kaldırılır.

Bir kabuk betiği, **`chmod`** komutu ile çalıştırma biti 1 yapılarak çalıştırılabilir yapılabilir. Bash betiğin içindeki komutları `$PATH` dizinlerinde arar ve bulduğunda bir altkabuk oluşturup onu orada çalıştırır. Başka bir deyişle, `dosyaismi` bir çalıştırılabilir kabuk betiği ise

```
dosyaismi argumanlar
```

ile

```
bash dosyaismi argumanlar
```

komutları birbirine eşdeğerdir. Bu altkabuk kendi kendini ilklendirir, yani etkisi, betiği yorumlayan ve komutlarının yerlerini bilen bir kabuktan onun çocuğu olarak yeni bir kabuk çağırmaya benzer. (**`hash`** yerleşiği için *Bourne Kabuğu Yerleşikleri* (sayfa: 38) bölümüne bakınız.)

Unix'in çoğu sürümleri bunu işletim sisteminin komut çalıştırma mekanizmasının bir parçası yapar. Eğer bir betiğin ilk satırı `#!` çifti ile başlarsa, satırın kalanı dosya içindeki programın yorumlayıcısıdır. Burada Bash, **`awk`**, **`perl`** ya da betik dosyasının yazıldığı dile ait bir yorumlayıcı belirtebilirsiniz.

Yorumlayıcı için yorumlayıcı satırında yorumlayıcı isminden sonra, ya da betiği çağırırken betik isminden hemen sonra tek bir seçenek belirtilebilir, bu son durumda betiğe verilebilecek diğer argümanlar da ondan sonra verilir. Bash bu eylemi işletim sisteminden birşey beklemeden yerine getirir.

Bilgi

Unix'in bazı eski sürümlerinde yorumlayıcı ismi ve tek seçeneği için toplam 32 karakterlik bir sınırlama vardır.

Bash betikleri çoğunlukla `#! /bin/bash` satırı ile başlar (Bash'ın `/bin` içinde bulunduğu varsayımıyla). Bu satır, betik başka bir kabuk altında çalıştırılsa bile betiğin Bash tarafından yorumlanmasını sağlar.

IV. Yerleşik Kabuk Komutları

İçindekiler

Yerleşik komutlar kabuğun kendi içinde bulunur. Bir yerleşik komutun ismi bir *basit komutun* (sayfa: 15) ilk sözcüğü olarak kullanılmışsa, kabuk başka bir uygulamayı başlatmaksızın komutu doğrudan doğruya çalıştırır. Yerleşik komutlar ayrı uygulamalar olarak çalıştırılmaları halinde sorunlar çıkarabilecek ya da ayrı bir uygulama olarak çalıştırılması imkansız işlevselliği sağlamak için gereklidir.

Bu oylumda Bourne Kabuğundan alıntılanan yerleşikler ile Bash'ın kendisine özgü ya da sonradan eklenmiş yerleşik komutlarından kısaca bahsedilecektir.

Bu oylumda yer verilmemiş yerleşik komutlar başka bölümlerde ele alınmıştır: İş denetimine Bash arayüzü sağlayanlar *İş Denetim Yerleşikleri* (sayfa: 84) bölümünde, dizin yığını *Dizin Yığını Yerleşikleri* (sayfa: 77) bölümünde, komut geçmişi *Bash Geçmiş Yerleşikleri* (sayfa: 111) bölümünde, ve programlanabilir tamamlama *Programlanabilir Tamamlama Yerleşikleri* (sayfa: 107) bölümünde anlatılmıştır.

Yerleşiklerin çoğu POSIX ya da Bash tarafından eklenmiştir.

Aksi belirtilmedikçe, **–** ile öncelenmiş seçenekler kabul eden belgelenmiş her yerleşik komut seçeneklerin sonunu işaretlemek için **–** kabul eder. Örneğin, **:**, **true**, **false** ve **test** yerleşikleri seçenek kabul etmezler.

1. Bourne Kabuğu Yerleşikleri

Aşağıdaki kabuk yerleşik komutları Bourne Kabuğundan miras alınmıştır. Bu komutlar POSIX standardında belirtildiği gibi gerçeklenmiştir.

1.1. : (iki nokta üstüste)

```
: [argümanlar]
```

Yönlendirmelerin uygulanması ve *argümanlar*'ın yorumlanması dışında hiçbir şey yapılmaz. Dönüş durumu sıfırdır.

1.2. . (nokta)

```
. dosyaismi [argümanlar]
```

Kullanılan kabuk kapsamında *dosyaismi*'nden komutlar okunur ve çalıştırılır. *dosyaismi* / içermiyorsa *dosyaismi*'nin yerini bulmak için `PATH` değişkeni kullanılır. Bash POSIX kipinde değilse ve `$PATH` içinde *dosyaismi* yoksa bulunulan dizine bakılır. Verilmiş *argümanlar* varsa, *dosyaismi* çalıştırılırken bunlar konumsal parametreler haline gelir. Aksi takdirde, konumsal parametreler değiştirilmez. Dönüş durumu son çalıştırılan komutun çıkış durumudur. Hiç komut çalıştırılmazsa sıfır durumu ile döner. *dosyaismi* bulunamazsa ya da okunamazsa, dönüş durumu sıfırdan faklı olur. Bu yerleşik **source** (sayfa: 56) yerleşiğine eşdeğerdir.

1.3. break Yerleşiği

```
break [n]
```

`for`, `while`, `until` veya `select` döngülerinden çıkılmasını sağlar. `n` verilmişse, dışa doğru n. döngüden çıkılır. `n >= 1` olmalıdır, bu koşul sağlanıyorsa dönüş durumu sıfırdır.

1.4. cd Yerleşiği

```
cd [-L|-P] [dizin]
```

Bulunulan dizinden *dizin*'e geçilmesini sağlar. *dizin* verilmezse `HOME` kabuk değişkeni ile belirtilen dizine geçilir. `CDPATH` kabuk değişkeni mevcutsa bu, arama yolu olarak kullanılır. *dizin* bir / ile başlıyorsa `CDPATH` kullanılmaz.

`-P` seçeneği verilmişse sembolik bağlar izlenmez. Sembolik bağların izlenmesi öntanımlıdır ve `-L` seçeneğine atanmıştır. *dizin* olarak `-` verilirse, `$OLDPWD` kullanılır.

CDPATH değişkenindeki boş olmayan değer ya da ilk argüman olarak **-** kullanılmışsa ve dizin değişikliği gerçekleşmişse yeni çalışma dizininin mutlak dosyayolu standart çıktıya yazılır.

Dizin değiştirme gerçekleşirse dönüş durumu sıfırdır, gerçekleşmezse sıfırdan farklıdır.

1.5. `continue` Yerleşiği

continue [*n*]

`for`, `while`, `until` veya `select` döngülerinin sonraki yinelemesinden devam edilir. *n* verilirse dışa doğru n. döngüden devam edilir. `n >= 1` olmalıdır, bu koşul sağlanıyorsa dönüş durumu sıfırdır.

1.6. `eval` Yerleşiği

eval [*argümanlar*]

argümanlar tek bir komut oluşturacak şekilde birleştirilip okunur ve çalıştırılır, çıkış durumu **eval**'ın çıkış durumu olarak döner. Hiç *argüman* verilmezse veya *argümanlar* boşsa sıfır durumu döner.

1.7. `exec` Yerleşiği

exec [**-cl**] [**-a** *isim*] [*komut* [*argümanlar*]]

komut verilirse, yeni bir süreç oluşturulmadan kabuğa yerleştirilir. `-l` seçeneği verilmişse, *komut*'a aktarılan 0. argümanın başına kabuk bir tire işareti koyar. Bu **login** uygulamasının yaptığıdır. `-c` seçeneği *komut*'un boş bir ortamda çalıştırılmasına sebep olur. `-a` seçeneği verilirse kabuk, *komut*'a *isim*'i 0. argüman olarak aktarır. Bir *komut* verilmezse, kullanımdaki kabuk ortamını etkilemekte yönlendirmeler kullanılabilir. Yönlendirme hatası yoksa, dönüş durumu sıfırdır; aksi takdirde sıfırdan farklıdır.

1.8. `exit` Yerleşiği

exit [*n*]

n durumu ile dönerek kabuk çıkar. *n* verilmezse son çalıştırılan komutun çıkış durumu döner. `EXIT` üzerindeki herhangi bir sinyal kapanı kabuk sonlanmadan önce çalıştırılır.

1.9. `export` Yerleşiği

export [**-fn**] [**-p**] [*isim*[=*değer*]]

Ortamdaki çocuk sürece aktarılacak her *isim*'i imler. `-f` seçeneği verilmişse *isim*'ler kabuk işlevleridir, aksi takdirde kabuk değişkenleridir. `-n` seçeneği verilirse aktarılacak *isim*'ler artık imlenmez. *isim* verilmemişse veya `-p` seçeneği verilmişse aktarılan *isim*'lerin listesi gösterilir. `-p` seçeneği çıktının girdi olarak tekrar kullanılabilir biçimde gösterilmesini sağlar. Bir değişken isminden sonra =*değer* geliyorsa *değer* değişkenin değeri yapılır.

İsimler geçersiz bir değişken ismi değilse veya `-f` seçeneği verilmişse ve isimlerin hepsi kabuk işlevi ise ya da geçersiz bir seçenek verilmemişse çıkış durumu sıfırdır.

1.10. `getopts` Yerleşiği

getopts *seçenek–dizgesi* [*argümanlar*]

getopts kabuk betikleri tarafından konumsal parametreleri çözümlemekte kullanılır. *seçenek–dizgesi* tanınan seçenek karakterlerini içerir; bir karakterden sonra bir iki nokta üstüste karakteri geliyorsa seçeneğin ondan bir boşlukla ayrılmış bir argümana sahip olduğu umulur. **:** ve ? karakterleri seçenek karakterleri olarak kullanılamaz. Her çağrılışında **getopts** işlenecek sonraki argümanın indisini OPTIND değişkenine ve *isim* değişkenini de mevcut değilse ilklendirerek sonraki seçeneği *isim* kabuk değişkenine yerleştirir. OPTIND kabuğun ya da bir kabuk betiğinin her çağrılışında 1 ile ilklendirilir. Bir seçenek bir argüman gerektirdiğinde **getopts** argümanı OPTARG değişkenine yerleştirir. Kabuk, OPTIND değişkenini otomatik olarak sıfırlamaz; eğer kullanılacak parametrelerin yeni bir kümesi varsa aynı kabuk çağrısı içinde **getopts**'a yapılan çoklu çağrılar arasında elle sıfırlanmalıdır.

Seçeneklerin sonuna gelindiğinde, **getopts** sıfırdan büyük bir çıkış durumu ile çıkar. OPTIND ilk seçenek olmayan argümanın indisine ayarlanır ve ? karakteri *isim*'e yerleştirilir.

getopts normalde konumsal parametreleri çözümlese de *argümanlar* verilmişse bunları çözümler.

getopts hataları iki yolla raporlayabilir. Eğer *seçenek–dizgesi*'nin ilk karakteri bir **:** ise *sessiz* hata raporlaması kullanılır. Geçersiz seçenekler veya eksik seçenekler saptandığında normal işlem tanı iletileri basılır. OPTERR değişkeninin değeri 0 ise *seçenek–dizgesi*'nin ilk karakteri bir **:** olmasa bile hata iletileri gösterilmez.

Geçersiz bir seçenek görüldüğünde, **getopts** ? karakterini *isim*'e yerleştirir ve hata raporlaması sessiz değilse bir hata iletisi basar ve OPTARG değişkenini kaldırır. **getopts** sessiz hata raporlama kipinde ise bulunan seçenek karakterleri OPTARG değişkenine yerleştirilir ve hiçbir tanı iletisi basılmaz.

Eğer bir gerekli argüman bulunamazsa ve **getopts** sessiz kipte değilse *isim*'e ? karakteri yerleştirilir, OPTARG kaldırılır ve bir tanı iletisi basılır. **getopts** sessiz kipte ise *isim*'e **:** karakteri yerleştirilir ve OPTARG'a bulunan seçenek karakteri yerleştirilir.

1.11. **hash** Yerleşiği

hash [**-lr**] [**-p** *dosyaismi*] [**-dt**] [*isim*]

isim argümanları olarak belirtilen komutların tam dosya yollarını hatırlar, böylece müteakip çağrılarda aranmalarına gerek kalmaz. Komutlar $PATH içinde listelenmiş dizinler aranarak bulunur. -p seçeneği dosyayolu aramalarını engeller ve *isim*'in konumu olarak *dosyaismi* kullanılır. -r seçeneği hatırlanan tüm konumları kabuğun unutmasına sebep olur. -d seçeneği her *isim* için hatırlanan konumu kabuğun unutmasına sebep olur. -t seçeneği verildiğinde her isme karşılık gelen dosya konumları basılır. -t seçeneği ile çok sayıda *isim* verilirse, *isim* hatırlanan tam dosyayolundan önce basılır. -l seçeneği çıktının girdi olarak tekrar kullanılmasını sağlayacak biçimde basılmasını sağlar. Hiç argüman belirtilmezse ya da sadece -l seçeneği belirtilirse hatırlanan komutlar hakkında bilgi basılır. Bulunamayan bir *isim* yoksa veya geçersiz bir seçenek verilmemişse dönüş durumu sıfırdır.

1.12. **pwd** Yerleşiği

pwd [**-LP**]

Bulunulan dizinin mutlak dosyayolunu basar. -P seçeneği verilirse basılan dosyayolu sembolik bağları içermez. -L seçeneği verilirse, basılan dosyayolu sembolik bağları içerebilir. Bulunulan dizinin ismi saptanırken bir hata oluşmamışsa ya da geçersiz bir seçenek verilmemişse dönüş durumu sıfırdır.

1.13. **readonly** Yerleşiği

readonly [**-apf**] [*isim*[=*değer*]] ...

Her *isim*'i salt–okunur olarak imler. Bu *isim*'lerin değerleri müteakip çağrılarda değiştirilemez. -f seçeneği verilirse, her *isim* bir kabuk işlevidir. -a seçeneği verilirse, her *isim* bir dizi değişkenidir. Argüman olarak hiç

isim verilmezse veya -p seçeneği verilirse salt–okunur isimlerin hepsi basılır. -p seçeneği gösterilen çıktının girdi olarak yeniden kullanılabilecek biçimde gösterilmesini sağlar. Bir değişken isminden sonra bir =*değer* geliyorsa *değer* değişkenin değeri yapılır. Geçersiz bir seçenek verilmedikçe, *isim* argümanları bir geçersiz değişken ya da işlev ismi olmadıkça veya -f seçeneği ile verilen bir isim geçersiz bir kabuk işlevi değilse dönüş durumu sıfırdır.

1.14. `return` Yerleşiği

```
return [n]
```

Bir kabuk işlevinden *n* dönüş durumu ile çıkılmasına sebep olur. *n* verilmezse işlev içinde son çalıştırılan komutun çıkış durumu, dönüş durumu olur. Bu yerleşik ayrıca `.` (veya **source**) yerleşiği ile çalıştırılan bir betiğin dönüş durumu olarak ya *n* ya da son çalıştırılan komutun çıkış durumuyla sonlandırılmasında da kullanılabilir. **RETURN** sinyal kapanı ile ilişkilendirilmiş bir komut, icra betik veya işlevden sonra kaldığı yerden devam etmeden önce çalıştırılır. **return** bir işlevin dışında kullanılmışsa ve **return** kullanılan betiğin çalıştırılması `.` veya **source** yerleşikleri ile yapılmamışsa dönüş durumu sıfırdan farklıdır.

1.15. `shift` Yerleşiği

```
shift [n]
```

Konumsal parametreleri *n* kadar sola kaydırır. İşlem sonunda $*n*>+1 ... $# konumsal parametreleri $1 ... $#−*n*+1 haline gelir. $#'dan *n*+1'e kadar olan parametreler kaldırılır. *n* >= 0 ve *n* =< $# olmalıdır. *n* sıfır ya da $# den büyükse konumsal parametreler değişmez. *n* verilmezse, öntanımlı olarak 1 kabul edilir. *n*> $# ve *n*< 0 değilse dönüş durumu sıfırdır, aksi takdirde sıfırdan farklıdır.

1.16. `test` Yerleşiği — [

```
test [ifade]

[ [ifade] ]
```

Koşullu *ifade* değerlendirilir. Her işleç ve terim ayrı bir argüman olmalıdır. İfadeler *Bash Koşullu İfadeleri* (sayfa: 72) bölümünde açıklanan önceliklerin birleşiminden oluşur. **test** seçeneklerin sonunu imleyen **–** argümanını kabul etmeyip yoksaydığı gibi bir seçenek de kabul etmez.

[kullanımında komutun son argümanı bir] olmalıdır.

İfadeler aşağıdaki işleçler kullanılarak birleştirilebilir. İşleçlerin öncelikleri verildikleri sıradadır.

! *ifade*
: *ifade* yanlışsa sonuç doğrudur.

(*ifade*)
: Sonuç *ifade*'nin değeridir. Bu kullanım işlecin normal önceliğini arttırmak içindir.

ifade1 -a *ifade2*
: Hem *ifade1* hem de *ifade2* doğru ise sonuç doğrudur.

ifade1 -o *ifade2*
: *ifade1* veya *ifade2* doğru ise sonuç doğrudur.

test ve **[** yerleşikleri koşullu ifadeleri argümanlarının sayısına bağlı bazı kurallara göre değerlendirir.

0 argüman

İfade yanlıştır.

1 argüman
: Sadece ve sadece argüman boş değilse ifade doğrudur.

2 argüman
: İlk argüman `!` ise ve ikinci argüman sadece ve sadece boş ise ifade doğrudur. İlk argüman tek terimli koşul işleçlerinden biri ise (*Bash Koşullu İfadeleri* (sayfa: 72) bölümüne bakınız) ve tek terimlinin sonucu doğru ise ifadenin sonucu doğrudur. İlk argüman geçerli bir tek terimli işleç değilse, ifade yanlıştır.

3 argüman
: İkinci argüman iki terimli koşul işleçlerinden biri ise (*Bash Koşullu İfadeleri* (sayfa: 72) bölüne bakınız), ifadenin sonucu ilk ve üçüncü argümanların terimleri olduğu iki terimlinin sonucudur. İlk argüman `!` ise sonuç, ikinci ve üçüncü argümanın kullanıldığı iki argümanlı sınamanın zıddıdır. İlk argüman tek başına `(` ve üçüncü argüman tek başına `)` ise sonuç, ikinci argümanın tek terimli sınamasının sonucudur. Aksi takdirde, ifade yanlıştır. `-a` ve `-o` işleçlerinin bu durumda iki terimli işleçleri olduğu varsayılır.

4 argüman
: İlk argüman `!` ise sonuç, kalan argümanların oluşturduğu üç terimli ifadenin sonucudur. Aksi takdirde, ifade yukarıdaki kurallar kullanılarak önceliklere göre çözümlenir ve değerlendirilir.

5 argüman ve fazlası
: İfade yukarıdaki kurallar kullanılarak önceliklere göre çözümlenir ve değerlendirilir.

1.17. `times` Yerleşiği

```
times
```

Kabuk ve altkabukları tarafında kullanılan kullanıcı ve sistem sürelerini gösterir. Dönüş durumu sıfırdır.

1.18. `trap` Yerleşiği

```
trap [-lp] [argüman] [sinyal ...]
```

argüman içindeki komutlar, kabuk *sinyal* sinyalini aldığında okunur ve çalıştırılır[1]. *argüman* bir boş dizge ise her *sinyal* sinyali kabuk ve onu çağıran komutlar tarafından yoksayılır[2]. *argüman* verilmezse (ve tek bir *sinyal* varsa) ya da `-` verilirse, belirtilen tüm sinyallere kabuk başlatıldığındaki değerleri yerleştirilir. [3]. Sadece `-p` seçeneği verilmişse **`trap`**, her sinyal numarası ile ilişkili komutların listesini kabuk girdisi olarak kullanılabilecek biçimde gösterir. `-l` seçeneği kabuğun sinyal isimlerini numaraları ile birlikte listelemesini sağlar. Her *sinyal* ya bir sinyal ismi ya da bir sinyal numarası olarak verilmelidir. Sinyal isimleri harf büyüklüğüne duyarlıdır ve `SIG` öneki isteğe bağlıdır. Eğer *sinyal* `0` ya da `EXIT` ise *argüman* kabuk çıkarken çalıştırılır. Eğer *sinyal* olarak `DEBUG` verilmişse, *argüman* komutu her basit komuttan önce, **`for`** komutundan, **`case`** komutundan, **`select`** komutundan, her aritmetik **`for`** komutundan ve bir kabuk işlevinde çalıştırılacak ilk komuttan önce çalıştırılır. `DEBUG` kapanının etkisi ile ilgili daha ayrıntılı açıklama için **`shopt`** yerleşiğinin `extglob` *seçeneğinin açıklamasına* (sayfa: 54) bakınız. Eğer bir *sinyal* `ERR` ise *argüman* komutu, aşağıdaki koşullara konu olan sıfırdan farklı çıkış durumuna sahip her komut için çalıştırılır. Başarısız komut, hemen ardından **`until`** veya **`while`** gelen bir komut listesinin parçasıysa ya da **`if`** deyiminde sınamanın bir parçasıysa veya bir **`&&`** veya **`||`** listesinin bir parçasıysa veya komutun dönüş durumu `!` ile zıddına ayarlanmışsa, `ERR` sinyal kapanı çalıştırılmaz. Bunlar **`errexit`** seçeneği tarafından uyulan koşullarla aynıdır. *sinyal* **`RETURN`** ise *argüman* ile belirtilen komut çalıştırılır.

Kabuk girdilerinde yoksayılan sinyaller yakalanamaz ve sıfırlanamaz. Yoksayılmayıp yakalanan sinyallere alt süreç içinde oluşturuldukları zamanki özgün değerleri atanır.

Bir *sinyal* geçersiz bir sinyal belirtmedikçe yerleşiğin dönüş durumu sıfırdır.

1.19. `umask` Yerleşiği

```
umask [-p] [-S] [kip]
```

Kabuk sürecinin dosya oluşturma maskesini *kip* olarak ayarlar. *kip* bir rakam ile başlıyorsa sekizlik bir sayı olarak yorumlanır; rakamla başlamıyorsa, **chmod** komutundaki gibi bir sembolik kip maskesi olarak yorumlanır. Eğer *kip* verilmezse, maskenin mevcut değeri gösterilir. `-S` seçeneği bir *kip* olmaksızın verilirse, maske sembolik kipte gösterilir. `-p` seçeneği bir *kip* olmaksızın verilirse, maske, kabuğa girdi olarak verilebilecek biçemde gösterilir. *kip* değiştirilebilmişse veya *kip* argümanı verilmemişse dönüş durumu sıfırdır, aksi takdirde sıfırdan farklıdır.

Bilgi

kip bir sekizlik sayı olarak yorumlandığında, maske içindeki her numara 7 den çıkarılır. Yani, `022` olarak verilen maske, `755` izinlerine karşılıktır.

1.20. `unset` Yerleşiği

```
unset [-fv] [isim]
```

isim'i belirtilen her işlev ya da değişken kaldırılır. Hiçbir seçenek verilmemişse ya da `-v` seçeneği verilmişse, her *isim* bir kabuk değişkenidir. `-f` seçeneği verilmişse, her *isim* bir kabuk işlevidir ve işlev tanımı kaldırılır. Salt–okunur değişkenler ve işlevler kaldırılamaz. Belirtilen *isim* salt–okunur olmadıkça dönüş durumu sıfırdır.

2. Bash Yerleşik Komutları

Bu kısımda eşsiz veya Bash'e sonradan eklenmiş yerleşik komutlardan bahsedilmiştir. Buradaki komutlardan bazıları POSIX standardında belirtilmiştir.

2.1. `alias` Yerleşiği

```
alias [-p] [isim[=değer] ...]
```

Argümansız veya `-p` seçeneği ile **alias**, kabukta girdi olarak yeniden kullanılabilecek biçemde takma adların listesini gösterir. Argümanlar verilmişse, her *isim* takma ad olarak alınır ve *değer* bu takma ada atanır. Eğer *değer* verilmemişse, takma adın *isim* ve *değer*'i basılır. Takma adlar *Takma Adlar* (sayfa: 75) bölümünde anlatılmıştır.

2.2. `bind` Yerleşiği

```
bind [-m tuşeşlem] [-lpsvPSV]
bind [-m tuşeşlem] [-q işlev] [-u işlev] [-r tuş–dizisi]
bind [-m tuşeşlem] -f dosyaismi
bind [-m tuşeşlem] -x tuş–dizisi:kabuk–komutu
bind [-m tuşeşlem] tuş–dizisi:işlev–ismi
bind readline–komutu]
```

Kullanımdaki Readline (*Komut Satırının Düzenlenmesi* (sayfa: 87) bölümüne bakınız) tuş ve işlev kısayollarını gösterir, bir Readline işlevi ya da makrosunu bir tuş dizisine bağlar veya bir Readline değişkenini atar. Seçenek olmayan her argüman bir *Readline ilklendirme dosyası* (sayfa: 90)ndaki gibi bir komuttur, fakat her kısayol veya komut ayrı bir argüman olarak verilmilidir; örn. `"\C-x\C-r":re-read-init-file`. Seçenekler verildiğinde aşağıdaki anlamlara gelir:

-m *tuşeşlemi*
Müteakip bağlantıların etkileneceği tuşeşlemi olarak *tuşeşlemi* kullanılır. Burada geçerli *tuşeşlemi* isimleri,`emacs`, `emacs-standard`, `emacs-meta`, `emacs-ctlx`, `vi`, `vi-move`, `vi-command` ve `vi-insert`'dir. `vi` ile `vi-command` ve `emacs` ile `emacs-standard` eşdeğerdir.

-l
Tüm Readline işlevlerini listeler.

-p
Readline işlev isimlerini ve kısayollarını, girdi olarak ya da bir Readline ilklendirme dosyasında kullanılabilir biçemde gösterir.

-P
Kullanımdaki Readline işlev isimlerini ve kısayollarını listeler.

-v
Readline değişken isimlerini ve değerlerini, girdi olarak ya da bir Readline ilklendirme dosyasında kullanılabilir biçemde gösterir.

-V
Kullanımdaki Readline değişken isimlerini ve değerlerini listeler.

-s
Makrolara ve dizgelere bağlı Readline tuş dizilerini, girdi olarak ya da bir Readline ilklendirme dosyasında kullanılabilir biçemde gösterir.

-S
Makrolara ve dizgelere bağlı Readline tuş dizilerini gösterir.

-f *dosyaismi*
Tuş kısayollarını *dosyaismi*'nden okur.

-q *işlev*
İsmi belirtilen *işlev*'i çağıran tuşlar hakkında sorgu.

-u *işlev*
İsmi belirtilen *işlev*'i çağıran tüm tuş kısayollarını kaldırır.

-r *tuş–dizisi*
tuş–dizisi için mevcut bağlantıları kaldırır.

-x *tuş–dizisi*:*kabuk–komutu*
tuş–dizisi'nin her girilişinde *kabuk–komutu*'nun çalıştırılmasını sağlar.

Geçersiz bir seçenek verilmedikçe veya bir hata oluşmadıkça dönüş durumu sıfırdır.

2.3. `builtin` Yerleşiği

builtin [*kabuk–yerleşiği* [*argümanlar*]]

Bir *kabuk–yerleşiği*'ne *argümanlar*'ı aktararak çalıştırır. Bu bir kabuk işlevinin bir kabul yerleşiği ile aynı isimde atanması durumunda faydalıdır. *kabuk–yerleşiği* geçerli bir kabuk yerleşiği değilse dönüş durumu sıfırdan farklıdır.

2.4. `caller` Yerleşiği

caller [*ifade*]

Etkin altyordam çağrısının (**.** veya **source** ile çalıştırılan bir betik) bağlamı ile döner.

ifade olmaksızın, **caller** o anki altyordam çağrısının kaynak dosya ismini ve satır numarasını gösterir. Eğer *ifade* olarak negatiften farklı bir tamsayı belirtilmişse **caller** o anki icra çağrısı yığıtındaki o konuma karşılık gelen kaynak dosyası, satır numarası ve altyordam ismini gösterir. Bu ek bilgi örneğin bir yığıt izini basmakta kullanılabilir. Anlık çerçeve 0. çerçevedir.

Kabuk bir altyordam çağrısını çalıştırmıyor olmadıkça dönüş durumu 0'dır, aksi takdirde *ifade* çağrı yığıtında geçerli bir konuma karşılık değildir.

2.5. **command** Yerleşiği

command [**-pVv**] *komut* [*argümanlar* ...]

komut komutunu *komut* isimli kabuk işlevini yoksayarak *argümanlar* ile çalıştırır. Sadece kabuk yerleşiklerini ve PATH araması ile bulunan komutları çalıştırır. ls isimli bir kabuk işlevi varsa ve işlevin içinde **command ls** varsa, işlev çağrısında işlev ardışık olarak çağrılmak yerine **ls** komutu çalıştırılacaktır. -p seçeneği, tüm standart uygulamaların bulunmasını garanti eden PATH için bir öntanımlı değerle kullanılacağı anlamına gelir. Bu durumda *komut* bulunamazsa veya bir hata oluşursa 127 durumu ile aksi takdirde *komut*'un çıkış durumu ile döner.

-V ya da -v seçeneği verilmişse, bir *komut* açıklaması basılır. -v seçeneği komutu çağırmada kullanılan dosya ismini ya da komutu belirten tek bir sözcük gösterilmesini sağlar. -V seçeneği ise daha ayrıntılı açıklama gösterilmesini sağlar. Bu durumda dönüş durumu *komut* bulunursa sıfır, aksi takdirde sıfırdan farklı olur.

2.6. **declare** Yerleşiği

declare [**-afFirtx**] [**-p**] [*isim*[=*değer*] ...]

Değişkenlerin özellikleri ile bildirilmesini sağlar. Hiçbir *isim* belirtilmezse değişkenleri ve değerlerini listeler.

-p seçeneği ile her *isim* için değerini ve özellikleri gösterecektir. -p seçeneği ile birlikte verilen diğer seçenekler yoksayılır. -F seçeneği işlev tanımlarının basılmasını engeller; sadece işlev isimleri ve özellikleri gösterilir. **extdebug** kabuk seçeneği **shopt** kullanarak etkinleştirilmişse ek olarak işlevin tanımlandığı kaynak dosya ismi ve satır numarası gösterilir. -F seçeneği -f seçeneğini uygular. Aşağıdaki seçenekler değişkenlere belirtilen özelliklerle çıktılanmasını veya değişken özelliklerinin verilmesini sınırlamakta kullanılabilir:

-a
: Her *isim* bir *dizi* (sayfa: 76) değişkenidir.

-f
: Sadece işlev isimleri kullanılır.

-i
: Değişken bir tamsayı olarak ele alınır ve değişkene bir değer atandığında *aritmetik değerlendirme* (sayfa: 74) uygulanır.

-r
: İsimleri salt–okunur yapar. Bu isimlere sonradan değer atanamaz, değiştirilemez ve kaldırılamaz.

-t
: Her *isim*'e **trace** özniteliği verir. İzlenen işlevler **DEBUG** ve **RETURN** tuzaklarını çağıran kabuktan miras alırlar. **trace** özniteliğinin değişkenler için özel bir anlamı yoktur.

`-x`
: Her ismi müteakip komutlara ortam üzerinden aktarılması için imler.

`-` yerine `+` kullanılarak özellik kapatılabilir. Bir işlevde kullanıldığında **`declare`** her *isim*'i **`local`** komutu kullanılmış gibi yerel yapar. Eğer değişken isminden hemen sonra bir *=değer* varsa *değer* değişkene atanır.

Şu hallerin gerçekleşmemesi şartıyla çıkış durumu sıfırdır: bir geçersiz seçenek saptanırsa, `-f foo=bar` kullanarak bir işleve atama yapmaya çalışıldığında, bir salt–okunur değişkene bir değer atanmaya çalışıldığında, bir dizi değişkenine birleşik atama sözdizimi (*Diziler* (sayfa: 76) bölümüne bakınız) kullanmaksızın bir değer atanmaya çalışıldığında, *isim*'lerden biri geçersiz bir kabuk değişkeni ismi ise, bir salt–okunur değişkenin salt–okunurluğu kaldırılmak istendiğinde, bir dizi değişkeninden dizi olmasını ortadan kaldıracak bir atama yapılmaya çalışıldığında veya mevcut olmayan bir işlevin `-f` seçeneği ile gösterilmesi istendiğinde.

2.7. `echo` Yerleşiği

`echo` `[-neE]` `[`*argümanlar* `...]`

Boşluklarla ayrılmış, bir satırsonu ile sonlandırılmış *argümanlar* çıktılanır. Dönüş durumu daima sıfırdır. `-n` seçeneği ile satırsonu karakteri baskılanır. `-e` seçeneği ile aşağıdaki tersbölü öncelemeli karakterlerin yorumlanması etkinleştirilir. `-E` seçeneği bu öncelenmiş karakterlerinin yorumlanmasının öntanımlı olarak etkin olduğu sistemlerde bile yorumlamayı kapatır. `xpg_echo` kabuk seçeneği öntanımlı olarak bu öncelenmiş karakterlerin yorumlanıp yorumlanmayacağının dinamik olarak saptanması için kullanılabilir. **`echo -`**'yi seçeneklerin sonu anlamında yorumlamaz.

`echo` aşağıdaki öncelenmiş dizilimleri yorumlar:

`\a`
: sesli uyarı (bell)

`\b`
: gerisilme

`\c`
: kendisinden sonra gelen satırsonu karakterini kaldırır.

`\e`
: escape

`\f`
: sayfa ilerletme

`\n`
: satırsonu

`\r`
: satırbaşı

`\t`
: yatay sekme

`\v`
: düşey sekme

`\\`
: tersbölü

\0*nnn*
: sekizlik değeri *nnn* olan sekiz bitlik karakter (0, 1, 2 ya da 3 haneli olabilir)

\x*hh*
: onaltılık değeri *hh* olan sekiz bitlik karakter (1 ya da 2 haneli olabilir)

2.8. `enable` Yerleşiği

enable [**-n**] [**-p**] [**-f** *dosyaismi*] [**-ads**] [*isim* ...]

Yerleşik kabuk komutlarını etkinleştirir ve kaldırır. Bir yerleşiğin iptal edilmesi durumunda, kabuk, yerleşik komutlara diğer komutlardan önce bakmasına rağmen bir tam dosyayolu belirtilmeksizin aynı isimdeki bir uygulamanın kabuk komutu olarak kullanılması sağlanır. Bu, `-n` seçeneği ile *isim*'leri belirtilen yerleşikler için uygulanır, seçenek verilmezse yerleşikler etkinleştirilir. Örneğin **test** yerleşiği yerine **test** uygulamasının yerleşik komut olarak kullanılmasını sağlamak için **enable -n test** yazmalısınız.

`-p` seçeneği verilirse ya da *isim* argümanları verilmezse kabuk yerleşikleri listelenir. Başka hiçbir argüman verilmezse tüm etkin kabuk yerleşikleri listelenir. `-a` seçeneği ile her yerleşik etkin olup olmadığı belirtilerek listelenir.

Dinamik yüklemenin desteklendiği sistemlerde, `-f` seçeneği ile yeni bir *isim* yerleşik komutunun *dosyaismi* ile belirtilen paylaşımlı nesneden yüklenmesi sağlanır. `-d` seçeneği `-f` ile yüklenen yerleşiği silmek için kullanılır.

Hiç seçenek verilmezse kabuk yerleşiklerinin bir listesi gösterilir. `-s` seçeneği POSIX'e özel yerleşiklerin etkinleştirilmesini sınırlar. `-s` seçeneği `-f` seçeneği ile birlikte kullanılırsa yeni yerleşik bir *özel yerleşik* (sayfa: 58) haline gelir.

Bir *isim* geçersiz bir kabuk yerleşiği olmadıkça ya da bir paylaşımlı nesneden yeni bir yerleşiği yüklerken hata olmadıkça dönüş durumu sıfırdır.

2.9. `help` Yerleşiği

help [**-s**] [*kalıp*]

Yerleşik komutlar hakkında yardım bilgisi gösterir. *kalıp* belirtilmişse, **help** *kalıp* ile eşleşen tüm komutlar için yardım bilgisi gösterir, aksi takdirde yerleşikler listelenir. `-s` seçeneği ile yardım bilgisi yerine komutun kullanımını gösteren sözdizimini basar. *kalıp* ile eşleşme olmaması dışında dönüş durumu sıfırdır.

2.10. `let` Yerleşiği

let *ifade* [*ifade*]

Kabuk değişkenleri üzerinde aritmetik işlemler uygulanmasını sağlar. Her *ifade* *Kabuk Aritmetiği* (sayfa: 74) bölümünde açıklanan kurallara göre değerlendirilir. Son *ifade*'nin sonucu 0 ise dönüş durumu 1 dir, aksi takdirde 0 dır.

2.11. `local` Yerleşiği

local [*seçenek*] *isim*[=*değer*] ...

Her argüman için *isim* isimli bir yerel değişken oluşturulur ve *değer* atanır. *seçenek*, **declare** tarafından kabul edilen seçenekler olabilir. **local** sadece bir işlev içinde kullanılabilir; *isim* değişkeninin sadece işlev ve çocuklarının etki alanında görünür olmasını sağlar. **local** bir işlevin dışında kullanılmış ya da *isim* bir salt–okunur değişken olmadıkça dönüş durumu sıfırdır.

2.12. `logout` Yerleşiği

```
logout [n]
```

Bir giriş kabuğundan çıkılmasını sağlar. Kabuğun atasına *n* durumu döner.

2.13. `printf` Yerleşiği

```
printf [-v değişken] biçim [argümanlar]
```

biçim'in denetimi altında *argümanlar*'ı standart çıktıya biçimleyerek yazar. *biçim* üç tür nesne içeren bir dizgedir: salt karakterler basitçe standart çıktıya kopyalanır, öncelenmiş karakterler dönüştürülüp standart çıktıya kopyalanır ve biçim belirtimleri; belirtimlerin her biri sırayla karşı düşen *argümanlar*'ın basılmasını sağlar. Standart **printf**(1) biçimlerine ek olarak, `%b` **printf**'in tersbölü öncelemeli karakterlerin karşı düşen *argüman*'da yorumlanmasını sağlar, (çıktıyı sonlandıran **\c** hariç, **\'**, **\"** ve **\?** içindeki tersbölüler kaldırılmaz ve **\0** ile başlayan sekizlik öncelemeler dört rakama kadar olabilir) ve `%q` **printf**'in karşı düşen *argüman*'ın kabuk girdisi olarak kullanılabilecek biçimde çıktılamasını sağlar.

`-v` seçeneği çıktının standart çıktıya basılması yerine *değişken*'e atanmasını sağlar.

biçim tüm *argümanlar* tüketilinceye kadar yeniden kullanılır. *argümanlar*'ın sayısı yetersizse, kalan biçim belirtimleri için türüne göre sıfır ya da boş dizge kullanılır. Bir hata oluşmazsa dönüş durumu sıfırdır, aksi takdirde sıfırdan farklıdır.

2.14. `read` Yerleşiği

```
read [-ers] [-a dizi–ismi] [-d sonlandırıcı] [-n krkt–sayısı] [-p istem]
     [-t zamanaşımı] [-u dosya–tanıtıcı]  [isim ...]
```

Standart girdiden ya da **-u** seçeneği ile sağlanan *dosya–tanıtıcı*'dan tek satır okur ve ilk sözcük ilk *isim*'e, ikinci sözcük ikinci *isim*'e ve böyle giderek, aradaki sözcükler ve ayraçlar son *isim*'e kadar atanır. Girdi akımından okunan sözcükler isimlerden daha azsa kalan isimlere boş değerler atanır. `IFS` değişkeninin değerindeki karakterler satırı sözcüklere ayırmakta kullanılır. Tersbölü karakteri (`\`) satırın devam ettirilmesi için ve kendinden sonra gelen özel karakterlerin yorumlanması için kullanılabilir. Hiç isim verilmemişse, satırın tamamı okunur ve `REPLY` değişkenine atanır. Dosyasonu karakteri (`Ctrl-D`) saptanmadıkça, komut zamanaşımına düşmedikçe ya da **-u** seçeneği ile sağlanan *dosya–tanıtıcı* geçersiz olmadıkça dönüş durumu sıfırdır. Seçenekler verildiklerinde aşağıdaki anlamlara gelirler:

-a *dizi–ismi*
: Sözcükler sırayla *dizi–ismi* dizisinin elemanlarına atanır. Dizinin ilk elemanının indisi 0 dır. Atama yapılmadan önce *dizi–ismi* dizisinin tüm elemanları silinir. Diğer *isim* argümanları yoksayılır.

-d *sonlandırıcı*
: *sonlandırıcı*'nın ilk karakteri girdi satırını sonlandırmak için satırsonu karakteri yerine kullanılır.

-e
: Satırı sağlamakta Readline (*Komut Satırının Düzenlenmesi* (sayfa: 87) bölümüne bakınız) kullanılır.

-n *krkt–sayısı*
: Satırın okunması *krkt–sayısı* karakterde sona erer. Satırın kalanı yoksayılır.

-p *istem*
: Girdi beklendiğini belirtecek *istem* dizgesi satırsonu karakteri olmaksızın görüntülenir. İstem sadece girdi bir uçbirimden gelecekse gösterilir.

-r
Bu seçenek verildiğinde tersbölü karakteri bir önceleme karakteri olarak ele alınmaz, dolayısıyla tersbölü–satırsonu çifti satırın alt satırda devam edeceğini belirtmekte kullanılamaz.

-s
Sessiz kip. Girdi bir uçbirimden geliyorsa karakterler yansılanmaz.

-t *zamanaşımı*
Satır *zamanaşımı* saniye sonra hala sonlandırılmamışsa **read** zamanaşımına düşer ve hata döner. Bu seçenek, girdi bir uçbirim ya da boruhattından okunmuyorsa etkisizdir.

-u *dosya–tanıtıcı*
Girdi*dosya–tanıtıcı*'dan okunur.

2.15. `set` Yerleşiği

set [**--abefhkmnptuvxBCHP**] [**-o** *seçenek*] [*argüman* ...]

Seçeneksiz ve argümansız olarak **set**, tüm kabuk değişkenlerinin ve işlevlerinin isim ve değerlerini yerelin alfabetik sıralamasına göre dizerek, girdi olarak yeniden kullanılabilir biçimde gösterir. Salt–okunur değişkenler sıfırlanamazlar. POSIX kipinde sadece kabuk değişkenleri listelenir.

Seçenekler verildiğinde kabuk özellikleri olarak atanır ya da kaldırılır. Seçenekler belirtildiğinde aşağıdaki anlamlara gelir:

-a
Müteakip komutların ortamına aktarılmak üzere değiştirilen veya oluşturulan işlev ve değişkenleri imler.

-b
Sonlandırılan artalan işlerin durumunu sonraki birincil komut isteminde gösterilmesini yerine anında raporlanmasını sağlar.

-e
Bir `until` veya `while` döngüsünden hemen sonra gelen bir komut listesinin bir kısmında, bir `if` deyiminin sınama parçasında, bir `&&` veya `||` listesinin bir parçasında hata oluşmadıkça veya komutun dönüş durumu `!` kullanarak ters çevrilmedikçe bir *basit komut* (sayfa: 15) sıfırdan farklı bir çıkış durumu ile çıkarsa anında çıkar. `ERR` için bir sinyal kapanı belirtilmişse, kabuk çıkmadan önce çalıştırılır.

-f
Dosya ismi üretimini (globbing) iptal eder.

-h
Çalıştırmak için aranan komutları bulur ve yerlerini hatırlar (hash) Bu seçenek öntanımlı olarak etkindir.

-k
Atama deyimleri şeklindeki tüm argümanları komut isminden önce belirtmek yerine ortama yerleştirir.

-m
İş denetimi (sayfa: 83) etkinleştirilir.

-n
Komutları okur ama onları çalıştırmaz; bu bir betiği sözdizimi hatalarına karşı denetlemek için yararlıdır. Bu seçenek etkileşimli kabuklarda yok sayılır.

-o *seçenek–ismi*

seçenek–ismi olarak aşağıdakilerden biri verildiğinde karşılığı olan seçenekler kullanılmış olur:

`allexport`
: `-a` seçeneği ile aynıdır.

`braceexpand`
: `-B` seçeneği ile aynıdır.

`emacs`
: Bir `emacs` tarzı satır düzenleme arayüzü kullanılır (*Komut Satırının Düzenlenmesi* (sayfa: 87) bölümüne bakınız).

`errexit`
: `-e` seçeneği ile aynıdır.

`errtrace`
: **`-E`** seçeneği ile aynıdır.

`functrace`
: **`-T`** seçeneği ile aynıdır.

`hashall`
: `-h` seçeneği ile aynıdır.

`histexpand`
: `-H` seçeneği ile aynıdır.

`history`
: *Bash'in Geçmişsel Yetenekleri* (sayfa: 111) bölümünde anlatıldığı gibi komut geçmişini etkinleştirir. Bu seçenek etkileşimli kabuklarda öntanımlı olarak etkindir.

`ignoreeof`
: Bir etkileşimli kabuk dosyasonu (EOF) karakterine rastlanınca çıkmaz.

`keyword`
: `-k` seçeneği ile aynıdır.

`monitor`
: `-m` seçeneği ile aynıdır.

`noclobber`
: `-C` seçeneği ile aynıdır.

`noexec`
: `-n` seçeneği ile aynıdır.

`noglob`
: `-f` seçeneği ile aynıdır.

`nolog`
: Yoksayılır.

`notify`

`-b` seçeneği ile aynıdır.

`nounset`
`-u` seçeneği ile aynıdır.

`onecmd`
`-t` seçeneği ile aynıdır.

`physical`
`-P` seçeneği ile aynıdır.

`pipefail`
Etkinse bir boruhattının dönüş değeri ya sıfırdan farklı bir durumla çıkan son (en sağdaki) komutun değeridir ya da boruhattındaki tüm komutlar başarılıysa sıfırdır. Bu seçenek öntanımlı olarak etkin değildir.

`posix`
Standart uyumu için POSIX standardındakinden farklı öntanımlı işlemde Bash davranışını değiştirir (*Bash POSIX Kipi* (sayfa: 80) bölümüne bakınız). Bu, Bash'ın standarda tam uyumunu sağlamak maksadıyla yapılmıştır.

`privileged`
`-p` seçeneği ile aynıdır.

`verbose`
`-v` seçeneği ile aynıdır.

`vi`
Bir `vi` tarzı satır düzenleme arayüzü kullanılır.

`xtrace`
`-x` seçeneği ile aynıdır.

`-p`
Ayrıcalıklı kipi etkinleştirir. Bu kipte, `$BASH_ENV` ve `$ENV` dosyaları işlenmez, kabuk işlevleri ortamdan miras alınmaz ve `SHELLOPTS` değişkeni ortamda görünüyorsa yoksayılır. Kabuk, gerçek kullanıcı (grup) kimliği ile aynı olmayan etkin kullanıcı (grup) kimliği ile başlatılmışsa ve `-p` seçeneği verilmemişse, bu eylemler alınır ve etkin kullanıcı (grup) kimliği, gerçek kullanıcı (grup) kimliğine ayarlanır. Başlatırken `-p` seçeneği verilmişse, etkin kullanıcı (grup) kimliği sıfırlanmaz. Bu seçeneğin kapatılması etkin kullanıcı ve grup kimliklerinin gerçek kullanıcı ve grup kimliklerine ayarlanmasına sebep olur.

`-t`
Tek bir komutu okuyup çalıştırdıktan sonra çıkar.

`-u`
Parametre yorumlaması uygulanırken bir hata sonucu değişkenlerin kaldırılmasına benzer bir davranış gösterir. Standart hataya bir hata iletisi yazılır ve etkileşimsiz kabuk çıkar.

`-v`
Kabuk girdi satırlarını okundu olarak basar.

`-x`

Basit komutların, **for**komutlarının, **case** komutlarının, **select** komutlarının ve aritmetik **for** komutlarının ve bunların argümanlarının veya ilişkili sözcük listelerinin yorumlandıktan sonra ve çalıştırılmadan önce izleme listesini basar. **PS4** değişkeninin değeri yorumlanır ve sonuçlanan değer komuttan ve komutun yorumlanan argümanlarından önce basılır.

-B

Kabuk *kaşlı ayraç yorumlaması* (sayfa: 23) uygular. Bu seçenek öntanımlı olarak etkindir.

-C

Çıktının > kullanılarak yönlendirilmesini ve <> kullanılarak mevcut dosyaların üzerine yazılmasını engeller.

-E

Etkinse **ERR** üstündeki bir tuzak kabuk işlevlerince miras alınır, komut ikameleri ve komutlar bir alt kabuk ortamında çalıştırılır. **ERR** tuzağı normalde böyle durumlarda miras alınmaz.

-H

! tarzı *geçmiş ikamesini* (sayfa: 113) etkinleştirir. Bu seçenek etkileşimli kabuklarda öntanımlı olarak etkindir.

-P

Verildiğinde, örneğin çalışılan dizini değiştirmek için **cd** gibi bir komut uygulandığında, sembolik bağları izlemez. Yerine fiziksel dizin kullanılır. Öntanımlı olarak, çalışılan dizini değiştiren komutlar uygulandığında, Bash mantıksal dizin zincirlerini izler.

Örneğin, /usr/sys dizini /usr/local/sys dizinine bir sembolik bağ ise:

```
$ cd /usr/sys; echo $PWD
/usr/sys
$ cd ..; pwd
/usr
```

set -P verilirse:

```
$ cd /usr/sys; echo $PWD
/usr/local/sys
$ cd ..; pwd
/usr/local
```

-T

Etkinse **DEBUG** ve **RETURN** üstündeki bir tuzak kabuk işlevlerince miras alınır, komut ikameleri ve komutlar bir alt kabuk ortamında çalıştırılır. **DEBUG** ve **RETURN** tuzakları normalde böyle durumlarda miras alınmaz.

--

Bu seçenekten sonra bir argüman verilmezse konumsal parametreler kaldırılır. Aksi takdirde, konumsal parametreler bazıları - ile başlasa bile argümanlara ayarlanır.

-

Seçeneklerin sonunu belirtir. Kalan tüm argümanlar konumsal parametrelere ayarlanır. -x ve -v seçenekleri kapatılır. Argüman yoksa konumsal parametreler değişmeden kalır.

Seçeneklerdeki - işaretleri yerine + kullanıldığında bu seçenekler kapatılır. Ayrıca, seçenekler kabuğun çağrılması sırasında da kullanılabilir. Seçeneklerin mevcut listesi $- içinde bulunabilir.

Kalan N *argüman* konumsal parametrelerdir ve $1, $2, ... $N şeklinde atanır. Özel parametre # ise N'e ayarlanır.

Geçersiz seçenek verilmediği sürece dönüş durumu daima sıfırdır.

2.16. `shopt` Yerleşiği

shopt [**-pqsu**] [**-o**] [*seçenek–ismi* ...]

İsteğe bağlı kabuk davranışlarını kontrol eden değişken değerlerini açıp kapar. Seçeneksiz ya da `-p` seçeneği ile tüm denetim seçeneklerini etkin olup olmadıklarını belirterek listeler. `-p` seçeneğinin farkı listlenenlerin kabuğa bir girdi olarak verilebilecek biçemde olmasıdır. Diğer seçeneklerin anlamları:

`-s`
: Belirtilen her *seçenek–ismi*'ni etkinleştirir (set'in s'si)

`-u`
: Belirtilen her *seçenek–ismi*'ni iptal eder (unset'in u'su).

`-q`
: Normal çıktıyı engeller; *seçenek–ismi*'nin etkinleştirme sonucunu dönüş durumu belli eder. Belirtilen tüm *seçenek–ismi*'leri etkinleştirilmişse dönüş durumu sıfırdır, aksi takdirde sıfırdan farklıdır.

`-o`
: **set** (sayfa: 49) yerleşiğinin `-o` seçeneğinde kullanılabilecek *seçenek–ismi* değerlerini sınırlar.

`-s` ve `-u` seçenekleri birlikte verilemez, bir *seçenek–ismi* ile birlikte verilmezlerse sadece kendilerine uyan seçenek isimlerini listelerler.

Aksi belirtilmedikçe, öntanımlı olarak **shopt** seçenekleri kapalıdır (off).

Seçenek isimleri listelenirken tüm seçenekler etkinse (on) dönüş durumu sıfırdır, aksi takdirde sıfırdan faklıdır. Seçeneklerin etkinleştirilmesi ve kaldırılması sırasında bir *seçenek–ismi* geçersiz bir seçenek ismi olmadıkça dönüş durumu sıfırdır.

seçenek–ismi olarak belirtilebilecek seçenekler:

`cdable_vars`
: Etkinleştirildiğinde **cd** (sayfa: 38) yerleşiğinin argümanı olarak dizin olmayan bir isim verildiğinde bunun geçilecek dizinin isminin atandığı bir değişkenin ismi olarak kabul edilmesini sağlar.

`cdspell`
: Etkinleştirildiğinde, **cd** komutunda bir dizin isminin yazılışında ufat tefek hatalar varsa düzeltilecektir. Bu hatalar; takdim–tehir, eksik bir karakter ve bir karakterin birden fazla yazılmasıdır. Bir düzeltme uygulanırsa düzeltilen dizin basılır ve komut çalıştırılır. Bu seçenek sadece etkileşimli kabuklarda kullanılır.

`checkhash`
: Etkinleştirildiğinde, Bash, bir komutu çalıştırmadan önce komut tablosuna bakar. Komut tablosunda komutu bulamazsa `PATH` dizinlerine bakar.

`checkwinsize`
: Etkinleştirildiğinde, Bash, her komuttan sonra pencere boyutunu ve lazımsa `LINES` ve `COLUMNS` değerlerindeki değişiklikleri kontrol eder.

`cmdhist`
: Etkinleştirildiğinde, Bash, aynı geçmiş girdisindeki çok satırlı bir komutun tüm satırlarını kaydetmeye çalışır. Böylece çok satırlı komutların yeniden düzenlenebilmesi kolaylaşır.

`dotglob`

Etkinleştirildiğinde, Bash, dosyaismi yorumlaması sonuçlarına `.` ile başlayan dosyaisimlerini dahil eder.

`execfail`
Etkinleştirildiğinde, **`exec`** yerleşiğine bir argüman olarak verilen bir dosya çalıştırılamadığında bir etkileşimsiz kabuk çıkmaz. Bir etkileşimli kabuk **`exec`** başarısız olduğunda zaten çıkmaz.

`expand_aliases`
Etkinleştirildiğinde, takma adlar *Takma Adlar* (sayfa: 75) bölümünde açıklandığı gibi yorumlanır. Bu seçenek etkileşimli kabuklarda öntanımlı olarak etkindir.

`extdebug`
Etkinse hata ayıklayıcılar tarafından kullanılmak üzere tasarlanmış davranış etkinleştirilir:

1. *declare Yerleşiği* (sayfa: 45)nin `-F` seçeneği bir argüman olarak sağlanmış her işleve karşılık düşen kaynak dosya isimlerini ve satır numaralarını gösterir.
2. Eğer komut **`DEBUG`** tuzağı tarafından çalıştırılmışsa sıfırdan farklı bir değerle döner, sonraki komut atlanır ve çalıştırılmaz.
3. Eğer komut **`DEBUG`** tuzağı tarafından çalıştırılmışsa 2 değeri ile döner ve kabuk bir altyordamı (`.` veya **`source`** yerleşikleri ile çalıştırılan bir kabuk betiği ya da kabuk işlevi) çalıştırarak bir **`return`** çağrısını taklit eder.
4. **`BASH_ARGC`** ve **`BASH_ARGV`** açıklamalarında belirtildiği gibi güncellenir (bkz, *Kabuk Değişkenleri* (sayfa: 59)).
5. İşlev izleme etkinleştirilir:
   `(` *komut* `)`
   ile çağrılan komut ikamesi, kabuk işlevleri ve alt kabuklar **`DEBUG`** ve **`RETURN`** tuzaklarını miras alır.
6. Hata izleme etkinleştirilir:
   `(` *komut* `)`
   ile çağrılan komut ikamesi, kabuk işlevleri ve alt kabuklar **`ERROR`** tuzağını miras alır.

`extglob`
Etkinleştirildiğinde, *Kalıp Eşleme* (sayfa: 28) bölümünde açıklanan özellikler etkinleştirilir.

`extquote`
Etkinse, çit tırnak içine alınmış `${`*parametre*`}` genişletmesi içinde `$'`*dizge* ve `$"`*dizge*`"` biçimi tırnak içine alma uygulanır. Bu seçenek öntanımlı olarak etkindir.

`failglob`
Etkinse dosyayolu yorumlaması sırasında dosya isimleri kalıplarla eşleşmezse sonuç bir yorumlama hatasıdır.

`force_fignore`
Etkinse `FIGNORE` (sayfa: 62) kabuk değişkeni tarafından belirtilen sonekler sözcük tamamlama uygulanırken yoksayılan sözcükler tek olası tamamlamalar olsalar bile sözcüklerin yoksayılmalarına sebep olurlar. Bu seçenek öntanımlı olarak etkindir.

`gnu_errfmt`
Etkinse kabuk hata iletileri standart GNU hata iletileri biçiminde yazılır.

`histappend`
: Kabuk çıkarken normalde geçmiş listesini, ismi `HISTFILE` değişkeninin değerinden alınan dosyanın üzerine yazar. Seçenek etkinleştirildiğinde geçmiş listesi bu dosyaya eklenir.

`histreedit`
: Etkinleştirildiğinde, Readline kullanımda ise, bir başarısız geçmiş ikamesini düzenlemesi için kullanıcıya bir fırsat tanınır.

`histverify`
: Etkinleştirildiğinde, Readline kullanımda ise geçmiş ikamesi kabuk çözümleyiciye hemen aktarılmaz. Bunun yerine, değişikliklerin yapılabilmesi için sonuç satırı Readline düzenleme tamponuna yüklenir.

`hostcomplete`
: Etkinleştirildiğinde, Readline kullanımda ise, Bash, `@` içeren bir sözcüğe konakismi tamamlaması uygulamaya çalışır (*Readline Sizin Yerinize Yazsın* (sayfa: 102) bölümüne bakınız). Bu seçenek öntanımlı olarak etkindir.

`huponexit`
: Etkinleştirildiğinde, bir etkileşimli giriş kabuğu çıktığında Bash, tüm işlere `SIGHUP` sinyali gönderecektir (*Sinyaller* (sayfa: 35) bölümüne bakınız).

`interactive_comments`
: Etkileşimli kabuklarda `#` ile başlayan bir sözcük ve devamındaki karakterlerin yoksayılması sağlanır. Bu seçenek öntanımlı olarak etkindir.

`lithist`
: Etkinleştirildiğinde, `cmdhist` seçeneği de etkinse, çok satırlı komutların satırları mümkün olduğunca `;` karakterleri ile ayrılarak değil gömülü satırsonları korunarak kaydedilir.

`login_shell`
: Kabuk, bir giriş kabuğu olarak başlatılırsa bu seçeneği etkinleştirir (*Bash'in Çağrılması* (sayfa: 67) bölümüne bakınız). Ancak değer değiştirilmeyebilir.

`mailwarn`
: Etkinleştirildiğinde, Bash'in eposta denetimi için kullandığı bir dosyaya son denetlemeden sonra erişildiğinde "`eposta-dosyası` *içindeki eposta okundu*" iletisini gösterir.

`no_empty_cmd_completion`
: Etkinleştirildiğinde, Readline kullanımda ise, bir boş satır tamamlanmaya çalışıldığında mümkün tamamlamaları bulmak için Bash, `PATH` dizinlerinde arama başlatmayacaktır.

`nocaseglob`
: Etkinleştirildiğinde, dosyaismi yorumlaması uygulanırken harf büyüklüğüne duyarsız dosyaismi eşlemesi yapılır.

`nocasematch`
: Etkinse **`case`** veya **`[[`** koşullu komutları çalıştırılırken eşleşme uygulandığında Bash kalıpları harf büyüklüğüne duyarsız kalarak eşleştirir.

`nullglob`
: Etkinleştirildiğinde, Bash, kendisine hiçbir dosya uymayan dosyaismi kalıplarının kendileri yerine boş dizge döndürmelerine izin verir.

`progcomp`
: Etkinleştirildiğinde, *programlanabilir tamamlama yetenekleri* (sayfa: 105) etkinleştirilir. Bu seçenek öntanımlı olarak etkindir.

`promptvars`
: Etkinleştirildiğinde, *komut istemi* (sayfa: 78) dizgeleri yorumlandıktan sonra bir de parametre yorumlaması, komut ikamesi, aritmetik yorumlaması ve tırnak kaldırmaya maruz kalır. Bu seçenek öntanımlı olarak etkindir.

`restricted_shell`
: Kabuk bu seçeneği, *sınırlı kipte* (sayfa: 80) başlatıldığında etkinleştirir. Değeri değişmeyebilir. Bash, başlatma dosyaları çalıştırılırken, başlangıç dosyalarının kabuğun sınırlı kipte olup olmadığını keşfetmesini sağlayarak sıfırlanmamasını sağlar.

`shift_verbose`
: Etkinleştirildiğinde, konumsal parametrelerin kaydırma miktarı parametre sayısını aşarsa **`shift`** *yerleşiği* (sayfa: 41) bir hata iletisi basar.

`sourcepath`
: Etkinleştirildiğinde, **`source`** *yerleşiğine* (sayfa: 56) argüman olarak verilen bir dosyanın yerini bulmak için `PATH` değeri kullanılır. Bu seçenek öntanımlı olarak etkindir.

`xpg_echo`
: Etkinleştirildiğinde, **`echo`** *yerleşiği* (sayfa: 46) öntanımlı olarak tersbölü öncelemeli karakterleri yorumlar.

Tüm *seçenek–ismi*'leri etkinse, seçenekler listelendiğinde dönüş durumu sıfırdır, aksi takdirde sıfırdan farklıdır. Seçenekleri etkinleştirirken ya da kaldırırken bir *seçenek–ismi* bir geçersiz kabuk seçeneği olmadıkça dönüş durumu sıfırdır.

2.17. `source` Yerleşiği

`source` *dosyaismi*

`.` *(nokta) yerleşiği* (sayfa: 38) ile eşdeğerdir.

2.18. `type` Yerleşiği

`type` [**`-afptP`**] [*isim* ...]

Her *isim*'in nasıl yorumlanması gerektiğini belirtir. Örneğin, **`type type`** komutu *type is a shell builtin* çıktılar (anlamı: "type bir kabuk yerleşiğidir").

`-t` seçeneği verildiğinde, *isim* bir takma ad ise `alias`, bir işlev ise `function`, bir yerleşik komut ise `builtin`, bir disk dosyası ise `file` veya bir anahtar sözcük ise `keyword` sözcüğünü basar. *isim* bulunamazsa, hiçbir şey basılmaz ve **`type`** bir başarısızlık durumu ile döner.

`-p` seçeneği verildiğinde, *isim* `-t` seçeneği ile `file` çıktısını vermiyorsa hiçbir şey dönmez, aksi takdirde çalıştırılacak disk dosyasının ismi basılır.

`-P` seçeneği her *isim* için, **`-t`** dosya döndürmese bile bir yol araması yapılmasını zorlar.

Komut çırpılanmışsa **`-p`** ve **`-P`** çırpılı değeri basar, dosyanın ilk olarak `$PATH` içinde görünmesi gerekli değildir.

`-a` seçeneği verildiğinde, *isim* çalıştırılabilir dosyasının bulunabileceği yerleri basar. Bu, sadece ve sadece `-p` seçeneği kullanılmamışsa takma adları ve işlevleri de içerir.

`-f` seçeneği belirtilmişse **`command`** yerleşiğindeki gibi **`type`** kabuk işlevlerini bulmaya çalışmaz.

*isim*lerden biri bulunursa dönüş durumu sıfırdır, hiçbiri bulunamazsa sıfırdan farklıdır.

2.19. `typeset` Yerleşiği

`typeset` [**`-afFrxi`**] [**`-p`**] [*isim*[*=değer*] …]

`typeset` komutu Korn kabuğu ile uyumluluk için vardır. Yerine **`declare`** *yerleşiğinin* (sayfa: 45) kullanılması önerilir.

2.20. `ulimit` Yerleşiği

`ulimit` [**`-acdefilmnpqrstuvxSH`**] [*sınır*]

`ulimit` kabuk tarafından başlatılan süreçlerin kullanabildiği özkaynaklar üzerinde, sistem buna izin veriyorsa, denetim sağlar. Seçenekler verildiğinde şu anlamlara gelir:

`-S`
: Bir özkaynakla ilişkili yazılımsal sınırı (soft limit) raporlar ya da değiştirir.

`-H`
: Bir özkaynakla ilişkili donanımsal sınırı (hard limit) raporlar ya da değiştirir.

`-a`
: Mevcut tüm sınırlamalar gösterilir.

`-c`
: Oluşan `core` dosyalarının azami boyu (core file size).

`-d`
: Bir sürecin veri segmanının azami boyu (data seg size).

`-e`
: Azami zamanlama önceliği (nice).

`-f`
: Kabuk ve çocukları tarafından yazılan dosyaların azami boyu (file size).

`-i`
: Askıdaki sinyallerin azami sayısı.

`-l`
: Belleğe kilitlenebilen azami uzunluk (max locked memory).

`-m`
: Azami bellek boyu (max memory size).

`-n`
: Açık dosya tanıtıcılarının azami sayısı (open files).

`-p`
: Boruhattı tamponunun boyu (pipe size).

`-q`
: POSIX ileti kuyruğunun azami bayt sayısı.

`-r`

Azami anında çalıştırma zamanlaması önceliği.

`-s`
Yığının azami boyu (stack size).

`-t`
İşlemci zamanının saniye cinsinden azami miktarı (cpu time).

`-u`
Tek bir kullanıcının kullanabileceği azami süreç sayısı (max user processes).

`-v`
Bir sürecin kullanabileceği sanal belleğin azami miktarı (virtual memory).

`-x`
Dosya kilitlerinin azami sayısı.

sınır verilmişse, belirtilen özkaynağın yeni değeridir; özel *sınır* değerleri mevcut donanımsal sınır için `hard`, mevcut yazılımsal sınır için `soft` ve sınırsız için `unlimited`'dir. Aksi takdirde, `-H` seçeneği belirtilmedikçe belirtilen özkaynak için mevcut yazılımsal sınır gösterilir. Yeni sınırlar ayarlanırken ne `-H` ne de `-S` seçeneği verilmişse hem yazılımsal hem de donanımsal sınırlar ayarlanır. Hiç seçenek verilmezse `-f` seçeneği verilmiş kabul edilir. Değerler `-t` için saniye cinsinden, `-p` için 512 baytlık blokların sayısı olarak, `-n` ve `-u` için birimsiz, kalan seçenekleri için 1024 baytlık blokların sayısı olarak belirtilmelidir.

Bir geçersiz seçenek ya da değer belirtilmedikçe veya yeni bir sınır belirtilirken bir hata oluşmadıkça dönüş durumu sıfırdır.

2.21. `unalias` Yerleşiği

```
unalias [-a] [isim ... ]
```

Belirtilen her *isim* takma adlar listesinden kaldırılır. `-a` seçeneği verilirse, tüm takma adlar kaldırılır. Takma adlar *Takma Adlar* (sayfa: 75) bölümünde açıklanmıştır.

3. Özel Yerleşikler

Tarihsel sebeplerle, POSIX standardında *özel* olarak sınıflandırılmış yerleşik komutlar vardır. Bash, POSIX kipinde çalıştırılırken, özel yerleşikler diğer yerleşiklerden üç hususta ayrılır:

1. Komut aranırken özel yerleşiklere kabuk işlevlerinden önce bakılır.
2. Bir özel yerleşik bir hata durumu ile dönerse, bir etkileşimsiz kabuk çıkar.
3. Komutttan önce verilen atama deyimleri komut işini tamamladıktan sonra da kabukta etkin olarak kalır.

Bash'ın POSIX kipinde çalıştırılmaması durumunda, bu yerleşikler Bash yerleşik komutlarından farklı davranmaz. Bash POSIX kipi *Bash POSIX Kipi* (sayfa: 80) bölümünde anlatılmıştır.

POSIX'e özel yerleşikler:

```
break : . continue eval exec exit export readonly return set
shift trap unset
```

V. Kabuk Değişkenleri

İçindekiler

Bu oylumda Bash tarafından kullanılan kabuk değişkenleri anlatılacaktır. Bash bazı değişkenleri otomatikman öntanımlı değerlerle atar.

1. Bourne Kabuğu Değişkenleri

Bash bazı kabuk değişkenlerini Bourne kabuğunun kullandığı gibi kullanır. Bazı durumlarda, Bash değişkenlere bir öntanımlı değer atar.

`CDPATH`
: **`cd`** yerleşik komutu için dosya arama yolu olarak kullanılan dizinlerin `:` işaretleri ile ayrılmış listesi.

`HOME`
: Kullanıcının ev dizini; **`cd`** yerleşik komutu için öntanımlı dizindir. Bu değişkenin değeri ayrıca *yaklaşık yorumlaması* (sayfa: 23) tarafından da kullanılır.

`IFS`
: Alanları ayırmakta kullanılan karakterlerin listesi; kabuk yorumlarının parçası olarak kabuk sözcükleri ayırırken kullanır.

`MAIL`
: Bu parametreye bir dosyaismi atanmışsa ve `MAILPATH` değişkeni atanmamışsa, bir eposta geldiğinde Bash epostanın belirtilen dosyada olduğuna dair kullanıcıyı bilgilendirir.

`MAILPATH`
: Kabuğun belirli aralıklarla yeni postaları kontrol ettiği dosyaların `:` işaretleri ile ayrılmış listesi. Her liste girdisi, posta dosyasına yeni bir posta geldiğinde, dosyaisimleri iletilerden bir `?` işareti ile ayrılarak bir ileti belirtebilir. İletinin metni içinde kullanıldığında `$_` değişkeni o anki posta dosyasının ismi olarak yorumlanır.

`OPTARG`
: **`getopts`** yerleşiği tarafından işlenen son seçenek argümanının değeri.

`OPTIND`
: **`getopts`** yerleşiği tarafından işlenen son seçenek argümanının indisi.

`PATH`
: Kabuğun komutları aradığı dizinlerin `:` işaretleri ile ayrılmış listesi. Sıfır uzunluklu (boş) bir `PATH` değeri içinde bulunulan dizini ifade eder. Böyle bir dizin ismi arada `::` biçiminde görüneceği gibi ilk veya son `:` olarak da görünebilir.

`PS1`
: Birincil komut istemi dizgesi. Öntanımlı değeri `\s-\v\$ ` dır. `PS1` gösterilmeden önce yorumlanan öncelenmiş karakterlerin tam listesini *Komut İsteminin Kontrol Edilmesi* (sayfa: 78) bölümünde bulabilirsiniz.

`PS2`
: İkincil komut istemi dizgesi. Öntanımlı değeri `> ` dır.

2. Bash Değişkenleri

Bu değişkenler Bash tarafından atanır ve kullanılır, ancak diğer kabuklar bu değişkenleri normalde kullanmaz.

Bash tarafından kullanılan bir kaç değişken bu kılavuzun başka bölümlerinde anlatılmıştır. İş denetim yeteneklerini kontrol eden değişkenleri *İş Denetim Değişkenleri* (sayfa: 86) bölümünde bulabilirsiniz.

`BASH`
: Bash'in çalışan kopyasının çalıştırılmasında kullanılan tam dosya yollu.

`BASH_ARGC`
: Değerleri o anki bash çalıştırma çağrı yığıtının her çerçevesindeki parametrelerin sayısı olan bir dizi değişkeni. O anki altyordam parametrelerinin sayısı yığıtın tepesinde bulunur. Bir altyordam çalıştırıldığında aktarılan parametre sayısı **`BASH_ARGC`**'ye üstten basılır. Kabuk **`BASH_ARGC`**'yi sadece ek hata ayıklama kipindeyken etkin kılar. (**`shopt`** yerleşiğinin **`extdebug`** seçeneğinin *açıklamasına bakınız* (sayfa: 54).)

`BASH_ARGV`
: O anki bash çalıştırma çağrı yığıtındaki tüm parametreleri içeren dizi değişkeni. Son altyordamın son parametresi yığıtın tepesinde bulunur; ilk çağrının ilk parametresi ise en altta bulunur. Bir altyordam çalıştırıldığında parametreleri **`BASH_ARGV`**'ye basılır. Kabuk **`BASH_ARGV`**'yi sadece ek hata ayıklama kipindeyken etkin kılar. (**`shopt`** yerleşiğinin **`extdebug`** seçeneğinin *açıklamasına bakınız* (sayfa: 54).)

`BASH_COMMAND`
: Bir sinyal kapanının sonucu olarak çalıştırılıyor olmadıkça o an çalışmakta olan ya da çalışmaya hazır olan komut.

`BASH_ENV`
: Bash bir kabuk betiğini çalıştırmak için çağrıldığında bu değişken atanmışsa, değişkenin değeri betiği çalıştırmadan önce okunacak *başlatma dosyası* (sayfa: 69)nın ismi olarak yorumlanır ve kullanılır.

`BASH_EXECUTION_STRING`
: `-c` çağrı seçeneğine belirtilen komut argümanı.

`BASH_LINENO`
: Üyeleri `FUNCNAME` dizisinin üyelerinin kaynak dosyalarda bulunduğu satır numaraları olan dizi değişkeni. `${BASH_LINENO[$i]}`, `${FUNCNAME[$i]}` çağrıldığında işlevin kaynak dosyasındaki satır numarasıdır. Kaynak dosyasının ismi ise `${BASH_SOURCE[$i]}`'dir. O anki satır numarasını elde etmek için `LINENO` değişkenini kullanın.

`BASH_REMATCH`
: Üyeleri **`[ [`** koşullu komutuna `=~` iki terimli işleci ile atanan dizi değişkeni (bkz, *Koşullu Çalıştırma* (sayfa: 17)). 0 indisli eleman düzenli ifadenin tamamı ile eşleşen dizge parçasını içerir. n indisli eleman ise n'inci parantezli alt ifadeyle eşleşen dizge parçasını içerir. Bu değişken salt–okunurdur.

`BASH_SOURCE`
: Üyeleri `FUNCNAME` dizisinin üyelerinin kaynak dosyalarının isimleri olan dizi değişkeni.

`BASH_SUBSHELL`
: Bir alt kabuk ya da bir alt kabuk ortamının her çatallanışında değeri bir

`BASH_VERSION`
: Bash'in kullanılan kopyasının sürüm numarası.

BASH_VERSINFO
: Bash'in kullanılan kopyasının sürüm bilgilerini saklamakta kullanılan salt–okunur bir *dizi değişkeni* (sayfa: 76). Dizi elemanlarına atanan değerler:

  BASH_VERSINFO[0]
  : Ana sürüm numarası (dağıtım).

  BASH_VERSINFO[1]
  : Alt sürüm numarası (sürüm).

  BASH_VERSINFO[2]
  : Yama seviyesi.

  BASH_VERSINFO[3]
  : Derlemenin sürümü.

  BASH_VERSINFO[4]
  : Dağıtımın durumu (örn., beta1).

  BASH_VERSINFO[5]
  : MACHTYPE değişkeninin değeri.

COLUMNS
: **select** yerleşiği tarafından seçim listelerini basarken uçbirim genişliğini saptamakta kullanılır. Bir SIGWINCH sinyali alındığında otomatik olarak atanır.

COMP_WORDBREAKS
: Sözcük tamamlama uygulanırken Readline kütüphanesince sözcük ayracı olarak ele alınacak karakterler. COMP_WORDBREAKS **unset** ile kaldırıldıktan sonra hemen sıfırlansa bile kendine özel niteliklerini yitirir.

COMP_CWORD
: O anki imleç konumunu içeren sözcüğün ${COMP_WORDS}'ü için bir indis. Bu değişken sadece *programlanabilir tamamlama* (sayfa: 105) araçları tarafından çağrılan kabuk işlevlerinde kullanılır.

COMP_LINE
: O anki komut satırı. Bu değişken sadece *programlanabilir tamamlama* (sayfa: 105) araçları tarafından çağrılan kabuk işlevleri ve disk komutları için kullanılır.

COMP_POINT
: O anki komutun başlangıcına göre o anki imleç konumunun indisi. O andaki imleç konumu o anki komutun sonunda ise değişkenin değeri ${#COMP_LINE}'a eşittir. Bu değişken sadece *programlanabilir tamamlama* (sayfa: 105) araçları tarafından çağrılan kabuk işlevleri ve disk komutları için kullanılır.

COMP_WORDS
: O andaki komut satırındaki tek tek sözcüklerin oluşturduğu dizi değişkeni. Sözcükler kabuk çözümleyicisinin ayrımsayabileceği şekilde kabuk ötekarakterleri ile birbirlerinden ayrılırlar. Bu değişken sadece *programlanabilir tamamlama* (sayfa: 105) araçları tarafından çağrılan kabuk işlevlerinde kullanılır.

COMPREPLY
: Bash'in *programlanabilir tamamlama* (sayfa: 105) araçları tarafından çağrılan bir kabuk işlevi tarafından üretilen mümkün tamamlamaları okuduğu dizi değişkeni.

DIRSTACK

Dizin yığınının o anki içeriğinin bulunduğu dizi değişkeni. Dizinler yığın içinde, **dirs** yerleşiği tarafından gösterildiği sırada bulunur. Bu dizi değişkeninin elemanlarına atama yoluyla yığındaki dizinler değiştirilebirse de, dizinleri eklemek ve kaldırmak için **pushd** ve **popd** yerleşikleri kullanılmalıdır. Bu değişkene atama yapılarak o anki dizin değiştirilemez. DIRSTACK kaldırılırsa, hemen ardından sıfırlansa bile kendine özgü niteliklerini kaybeder.

EMACS
Kabuk **t** değeriyle başlarken Bash ortamda bu değişkeni bulursa kabuğun bir emacs kabuk tamponunda çalıştığını varsayar ve satır düzenlemeyi kapatır.

EUID
O anki kullanıcının etkin kullanıcı kimliğinin numarası. Salt–okunurdur.

FCEDIT
fc yerleşik komutunun -e seçeneği tarafından öntanımlı olarak kullanılan metin düzenleyici.

FIGNORE
Dosyaismi tamamlaması uygulanırken yoksayılacak soneklerin **:** ayraçlı listesi. Soneki FIGNORE içindeki girdilerden biri ile eşleşen bir dosya, eşleşen dosya isimleri listesinden çıkarılır. Bir örnek değer: .o:~.

FUNCNAME
O an çalışan kabuk işlevinin ismi. O anki çalıştırma çağrı yığıtındaki tüm kabuk işlevlerinin isimlerini içeren bir dizi değişkeni. 0 indisli eleman o an çalışmakta olan kabuk işlevinin ismidir. En alttaki eleman ise "main" işlevidir. Bu değişken sadece bir kabuk işlevi çalışırken vardır. FUNCNAME'e yapılan değer atamaları etkisizdir ve bir hata durumu döner. FUNCNAME, **unset** komutu ile kaldırılırsa, ardından tekrar atansa bile, sahip olduğu özel nitelikleri kaybeder.

GLOBIGNORE
Dosyaismi yorumlaması tarafından yoksayılan dosyaisimleri kümesini tanımlayan kalıpların **:** ayraçlı listesi. Bir dosya ismi, hem bir dosyaismi yorumlama kalıbı ile hem de GLOBIGNORE içindeki kalıplardan biri ile eşleşiyorsa, eşleşenler listesinden kaldırılır.

GROUPS
O anki kullanıcının üyesi olduğu grupların listesini tutan bir dizi değişkeni. Değişkene yapılan atamalar etkisizdir ve bir hata durumu döner. GROUPS, **unset** komutu ile kaldırılırsa, ardından tekrar atansa bile, sahip olduğu özel nitelikleri kaybeder.

histchars
Geçmiş yorumlaması (sayfa: 113), hızlı ikame ve sembolleştirmeyi kontrol eden en çok üç karakter. İlk karakter geçmiş yorumlamasının başlatılmasını sağlayan *geçmiş yorumlama* karakteridir ve normalde ! işaretidir. İkinci karakter, bir satırdaki ilk karakter olduğunda 'hızlı ikame'yi imleyen karakterdir ve normalde ^ işaretidir. İstemlik olan üçüncü karakter ise, bir sözcüğün ilk karakteri olarak bulunduğunda satırın kalanının açıklama olmasını sağlayan karakterdir ve normalde # işaretidir. Kabuk çözümleyicisinin satırın kalanını bir açıklama sayması için bu karakterin burada atanması gerekli değildir.

HISTCMD
O anki komutun geçmiş numarası ya da geçmiş listesindeki indisi. HISTCMD, **unset** komutu ile kaldırılırsa, ardından tekrar atansa bile, sahip olduğu özel nitelikleri kaybeder.

HISTCONTROL

Geçmiş listesinde kayıtlı kaç komutun olduğunu denetleyen değerlerin `:` ayraçlı listesi. Eğer değer listesi **ignorespace** içeriyorsa bir boşluk karakteri ile başlayan satırlar geçmiş listesine kaydedilmez. Bir **ignoredups** değeri varsa önceki geçmiş girdisi ile aynı olan satırlar kaydedilmez. Bir **ignoreboth** değeri **ignorespace** ve **ignoredups** için kısaltmadır. Bir **erasedups** değeri o anki satırla aynı olan tüm önceki satırlar satır kaydedilmeden önce geçmiş listesinden silinmesine sebep olur. Buraya kadar bahsedilenler dışında bir değer yoksayılır. `HISTCONTROL` eğişkeni **unset** ile ortamdan kaldırılırsa ya da geçerli bir değer içermiyorsa kabuk çözümleyici tarafından okunan `HISTIGNORE` değerine konu tüm satırlar geçmiş listesine kaydedilir. Bir çok–satırlı birleşik komutun ikinci ve sonraki satırları denetlenmez ve `HISTCONTROL` değişkeninin değerine bakılmaksızın geçmişe eklenir. Değişken kaldırılırsa ya da bu üçünden farklı bir değer atanırsa bütün satırlar geçmiş listesine eklenir.

`HISTFILE`
Komut geçmişinin kaydedildiği dosyanın ismi. Öntanımlı değeri `~/.bash_history`'dir.

`HISTFILESIZE`
Geçmiş dosyasındaki satırların azami sayısı. Bu değişkene bir değer atandığında geçmiş dosyasında bu değerden fazla satırlar varsa silinir. Etkileşimli bir kabuk çıkarken kabuğun geçmiş listesi dosyaya yazıldıktan sonra bu değerden fazla olan satırları silinir. Öntanımlı değeri 500'dür.

`HISTIGNORE`
Geçmiş listesine hangi satırların kaydedilmesi gerektiğine karar vermek için kullanılan kalıpların `:` ayraçlı listesi. Her kalıp bu listenin başına yerleştirilir ve kalıp satırın tamamı ile uyumlu olmalıdır. Her kalıp, `HISTIGNORE` tarafından belirtilen denetimler uygulandıktan sonra satırda test edilir. Normal kabuk kalıp eşleştirme karakterine ek olarak `&` önceki geçmiş satırını ifade eder. `&` bir tersbölü ile öncelenebilir; bir eşleştirme öncesi bu tersbölü kaldırılır. Bir çok–satırlı birleşik komutun ikinci ve sonraki satırları denetlenmez ve `HISTIGNORE` değişkeninin değerine bakılmaksızın geçmişe eklenir.

`HISTIGNORE`, `HISTCONTROL`'ün işlevselliğini de altında toplar. Bir `&` kalıbı `ignoredups`'a eşdeğerken, bir `[ ]*` kalıbı `ignorespace`'e eşdeğerdir. Bu iki kalıp bir `:` ile birleştirilerek `ignoreboth`'un işlevselliği de sağlanır.

`HISTSIZE`
Geçmiş listesinde hatırlanan komutların azami sayısı. Öntanımlı değeri 500'dür.

`HISTTIMEFORMAT`
Bu değişken ortamda mevcutsa ve boş değerli değilse değeri **history** yerleşiği tarafından gösterilen her geçmiş girdisi ile ilişkili zaman damgasını basacak `strftime` için bir biçim dizgesi olarak kullanılır. Bu değişken tanımlıysa zaman damgaları geçmiş dosyasına yazılır böylece kabuk oturumlarına karşı korunabilirler.

`HOSTFILE`
Kabuk bir konak isminin tamamlamaya çalıştığında okuması gereken `/etc/hosts` ile aynı biçemdeki bir dosyanın ismini içerir. Olası konak ismi tamamlamaları kabuk çalışırken değişebilir; değer değiştikten sonra yapılan bir başka konak ismi tamamlamasında, Bash yeni dosyanın içeriğini mevcut listeye ekler. `HOSTFILE` değer verilmeden atanmışsa, Bash olası konak ismi tamamlamalarının listesini `/etc/hosts` dosyasından okumaya çalışır. `HOSTFILE` kaldırılırsa konak ismi listesi temizlenir.

`HOSTNAME`
O anki konak ismi.

`HOSTTYPE`
Bash'in üzerinde çalıştığı makinayı tanımlayan bir dizge.

IGNOREEOF
: Tek girdi olarak EOF karakteri alındığında kabuğun eylemini kontrol eder. Atandığında değeri, kabuk çıkmadan önce bir girdi satırındaki ilk karakter olarak okunabilen ardışık EOF karakterlerinin sayısını gösterir. Değişken bir sayısal olmayan değere sahipse ya da boşsa öntanımlı değer 10 dur. Eğer değişken mevcut değilse, EOF kabukta girdi sonunu belirtir. Bu sadece etkileşimli kabuklarda etkilidir.

INPUTRC
: Readline ilklendirme dosyasının ismi. Dosyanın öntanımlı değeri, ~/.inputrc'dir.

LANG
: LC_ ile başlayan bir değişkenle özellikle seçilmemiş bir bir kategori için yerel kategoriyi saptamakta kullanılır.

LC_ALL
: Bu değişken bir yerel kategori belirten LC_ ile başlayan değişkenler ile LANG değişkeninin değeri yerine kendi değerini geçerli yapar.

LC_COLLATE
: Bu değişken dosyaismi yorumlamasının sonuçlarını sıralamada, aralık ifadelerinin davranışlarını saptamada, denklik sınıflarında, *dosyaismi yorumlaması* (sayfa: 27) ve kalıp eşleştirme içindeki alfabetik dizilimlerde harf sıralamasını saptamakta kullanılır.

LC_CTYPE
: Bu değişken *dosyaismi yorumlaması* (sayfa: 27) ve kalıp eşleştirme içindeki karakter sınıflarının davranışını ve karakterlerin yorumlanmasını belirler.

LC_MESSAGES
: Bu değişken $ ile öncelenmiş çift tırnak içindeki dizgelerin çevirilerinin kullanıldığı yereli belirler (*Yerele Özel Çeviri* (sayfa: 14) bölümüne bakınız).

LC_NUMERIC
: Bu değişken numara biçemlemesi için kullanılan yerel kategoriyi belirler.

LINENO
: O an çalışmakta olan kabuk işlevi ya da betik içindeki satırın numarasıdır.

LINES
: **select** yerleşiği tarafından seçim listelerini basarken sütun uzunluğunu saptamakta kullanılır. Bir SIGWINCH sinyali alındığında otomatik olarak atanır.

MACHTYPE
: Bash'in üzerinde çalıştığı sistem türünü açıklayan standart GNU *işlemci–firma–sistem* biçeminde bir dizgedir.

MAILCHECK
: Kabuğun posta için MAILPATH veya MAIL değişkenlerinde belirtilen dosyaları ne sıklıkta (saniye cinsinden) kontrol edeceğini belirler. Öntanımlı 60 saniyedir. Postanın kontrol zamanı geldiğinde kabuk komut istemini göstermeden önce bunu yapar. Bu değişken kaldırılırsa ya da değeri sıfırdan büyük ya da eşit olmayan bir sayı olarak atanırsa kabuk posta denetimini iptal eder.

OLDPWD
: **cd** yerleşiği tarafından atanan önceki dizin.

OPTERR
: 1 değeri atandığında Bash, **getopts** yerleşik komutu tarafından üretilen hata iletilerini gösterir.

OSTYPE
: Bash'in üzerinde çalıştığı işletim sistemini açıklayan bir dizge.

PIPESTATUS
: Önalan boruhattında en–son–çalışan süreçlerin çıkış durumu değerlerinin bir listesini içeren bir *dizi değişkeni* (sayfa: 76).

POSIXLY_CORRECT
: Bash başlatılırken bu değişken ortamda ise, başlatma dosyalarını okumadan önce, `--posix` çağrı seçeneği verilmişçesine, kabuk *POSIX kipine* (sayfa: 80) girer. Eğer kabuk çalışırken atanırsa,

```
set -o posix
```

komutu çalıştırılmışçasına POSIX kipi etkinleştirilir.

PPID
: Kabuğu çağıran kabuğun süreç kimliği. Bu salt–okunur bir değişkendir.

PROMPT_COMMAND
: Atanmışsa, değeri her birincil komut isteminden (`$PS1`) önce çalıştırılacak komut olarak yorumlanır. Daha ayrıntılı bilgi için *Komut İsteminin Kontrol Edilmesi* (sayfa: 78) bölümüne bakınız.

PS3
: Bu değişkenin değeri **select** komutu için istem olarak kullanılır. Bu değişken atanmamışsa, **select** komutu `#?` ile istemde bulunur.

PS4
: Değişkenin değeri, **set** *yerleşiğinin* (sayfa: 49) `-x` seçeneği ile etkinleştirilen yansılamalı komut satırından önce basılan istemdir. `PS4`'ün ilk karakteri, adreslemenin çoklu seviyelerini belirtecek şekilde gerektikçe defalarca kopyalanır. Öntanımlı değeri `+` 'tur.

PWD
: **cd** yerleşiği ile geçilen dizin.

RANDOM
: Bu parametre her zaman 0 ile 32767 arasında üretilmiş bir rasgele tamsayıdır. Bu değişkene bir değer atanması rasgele sayı üretecini tohumlar[4].

REPLY
: **read** yerleşiği için öntanımlı değişken.

SECONDS
: Bu değişken kabuk başlatıldığından beri kaç saniye geçtiğini gösterir. Bu değişkene bir değer atanması sayacı bu değere ayarlar ve yorumlanan değer, atanan değer artı atamadan itibaren geçen saniyelerdir.

SHELL
: Kabuğa tam dosya yolu bu ortam değişkeninde tutulur. Kabuk başlatıldığında bu değişken tanımlı değilse Bash değer olarak o anki kullanıcının giriş kabuğunun tam dosyayolunu atar.

SHELLOPTS

Etkin kabuk seçeneklerinin `:` ayraçlı listesidir. Listedeki her sözcük **set** *yerleşiğinin* (sayfa: 49) `-o` seçeneğinin geçerli bir argümanıdır. `SHELLOPTS` içinde görünen seçenekler **set -o** komutunun çıktısında `on` olarak raporlanır. Bash başlatıldığında, bu değişken ortamda ise listedeki her kabuk seçeneği başlatma dosyaları okunmadan önce ekinleştirilir. Bu salt–okunur bir değişkendir.

`SHLVL`
Bash'in her yeni kopyası başlatıldığında değeri 1 artar. Bu değer, Bash kabuklarınızın ne kadar derinlikte iç içe olduğunu gösteren bir sayaç olarak düşünülmüştür.

`TIMEFORMAT`
Bu parametrenin değeri `time` anahtar sözcüğü ile başlayan boruhatları için zamanlama bigisinin nasıl belirtileceğini gösteren bir biçem dizgesi olarak kullanılır. `%` karakteri bir zaman değeri veya başka bir bilgi olarak yorumlanan bir önceleme dizgesi girer. Önceleme dizgelerinin anlamları aşağıda açıklanmıştır; köşeli parantezler istemlik kısımları gösterir.

`%%`
`%` sabiti.

`%[p][l]R`
Saniye cinsinden geçen zaman.

`%[p][l]U`
Kullanıcı kipinde harcanan saniye cinsinden işlemci zamanı.

`%[p][l]S`
Sistem kipinde harcanan saniye cinsinden işlemci zamanı.

`%P`
`(%U + %S) / %R` olarak hesaplanan işlemci yüzdesi.

İstemlik bir `p` ondalık hanelerin sayısını belirten bir rakamdır. 0 değeri ile bir ondalık nokta gösterilmez ya da çıktı bir kesir olur. Ondalık noktadan itibaren en çok üç hane belirtilebilir. Üçten büyük değerler üç olarak değiştirilir. `p` belirtilmezse 3 değeri kullanılır.

İstemlik `l` dakikaları da içererek daha uzun biçemde, *dd***m***ss.kk* biçiminde gösterir. `p`'nin değeri kesir (*kk*) kısmının olup olmayacağını belirler.

Bu değişken atanmamışsa, Bash aşağıdaki değerin varlığını kabul eder.

```
$'\nreal\t%3lR\nuser\t%3lU\nsys\t%3lS'
```

Değer boşsa, zamanlama bilgisi gösterilmez. Biçem dizgesi gösterilirken sonuna bir satırsonu eklenir.

`TMOUT`
Sıfırdan büyük bir değer atanmışsa **TMOUT**, **read** yerleşiği için öntanımlı zamanaşımı olarak ele alınır. Girdi bir uçbirimden beklendiğinde **TMOUT** saniye sonra girdi hala görünmemişse **select** (sayfa: 18) komutu sonlanır.

Kabuk etkileşimliyse, atanan değer birincil komut istemi gösterildikten sonra girdi için beklenecek saniye sayısı olarak yorumlanır. Belirtilen süre bitimine dek bir girdi olmazsa Bash sonlanır.

`TMPDIR`
Tanımlıysa değeri Bash tarafından kabuğun kullanımı için Bash'in oluşturduğu geçici dosyaları içerecek dizinin ismi olarak kullanılır.

`UID`
O anki kullanıcının sayısal gerçek kullanıcı kimliği. Bu salt–okunur bir değişkendir.

VI. Bash'in Özellikleri

İçindekiler

Bu oylumda sadece Bash'de bulunan özellikler açıklanmıştır.

1. Bash'in Çağrılması

bash [*uzun–seçen*] [**-ir**] **[-abefhkmnptuvxdBCDHP]** [**-o** *seçen*] [**-O** *shopt_seçen*] [*argüman* ...]
bash [*uzun–seçen*] [**-abefhkmnptuvxdBCDHP**] [**-o** *seçen*] [**-O** *shopt_seçen*] **-c** *dizge* [*argüman* ...]
bash [*uzun–seçen*] **-s** [**-abefhkmnptuvxdBCDHP**] [**-o** *seçen*] [**-O** *shopt_seçen*] [*argüman* ...]

Tek karakterlik kabuk *komut satırı seçenekleri* (sayfa: 49) yanında kullanabileceğiniz çok karakterli seçenekler de vardır.Bu seçenekler tanınabilmeleri için komut satırında tek karakterli seçeneklerden önce verilmelidir.

`--debugger`
: Kabuk başlatılmadan önce çalıştırılacak hata ayıklama profili için düzenleme yapar. Ek *hata ayıklama kipini* (sayfa: 54) ve *kabuk işlevi izlemeyi* (sayfa: 50) etkinleştirir.

`--dump-po-strings`
: `$` ile öncelenmiş tüm çift tırnaklı dizgelerin bir listesi GNU **gettext** PO (portable object – uyarlanabilir nesne) dosya biçeminde standart çıktıya basılır. Çıktı biçemi dışında `-D` seçeneğine eşdeğerdir.

`--dump-strings`
: `-D` seçeneğine eşdeğerdir.

`--help`
: Standart çıktıda bir kullanım iletisi gösterir ve başarıyla çıkar.

`--init-file` *dosyaismi*
`--rcfile` *dosyaismi*
: Bir etkileşimli kabukta *dosyaismi* dosyasındaki komutları çalıştırır (`~/.bashrc` yerine).

`--login`

`-l` seçeneğine eşdeğerdir.

`--noediting`
Kabuk etkileşimliyken komut satırlarını okuyacak GNU Readline kütüphanesi kullanılmaz (*Komut Satırının Düzenlenmesi* (sayfa: 87) bölümüne bakınız).

`--noprofile`
Bash bir giriş kabuğu olarak çağrıldığında sistem çapında başlatma dosyası olan `/etc/profile` veya `~/.bash_profile`, `~/.bash_login` ve `~/.profile` kişisel ilklendirme dosyalarından herhangi birini yüklemez.

`--norc`
Bir etkileşimli kabukta `~/.bashrc` ilklendirme dosyasını okumaz. Kabuk **sh** olarak çağrılmışsa bu seçenek öntanımlı olarak etkindir.

`--posix`
Öntanımlı işlemin POSIX standardından farklı olduğu yerde standarda uyum için Bash'in davranışını değiştirir. Bash POSIX kipinin açıklaması için *Bash POSIX Kipi* (sayfa: 80) bölümüne bakınız.

`--restricted`
Kabuğu bir *sınırlı kabuk* (sayfa: 80) yapar.

`--verbose`
`-v` seçeneğine eşdeğerdir. Kabuk girdi satırlarını okunduğu gibi basar.

`--version`
Bash'in bu kopyası için sürüm bilgilerini gösterir ve çıkar.

set yerleşiği ile kullanılamayan, çağrı sırasında verilebilen tek karakterlik seçenekler de vardır.

`-c` *dizge*
Seçenekleri işledikten sonra *dizge*den komutları okuyup çalıştırır ve çıkar. Kalan argümanlar `$0` ile başlayan konumsal parametrelere atanır.

`-i`
Kabuk etkileşimli çalışmaya zorlanır. Etkileşimli kabuklar *Etkileşimli Kabuklar* (sayfa: 70) bölümünde açıklanmıştır.

`-l`
Bu kabuğu **login** tarafından doğrudan çağrılmışçasına etkin yapar. Kabuk etkileşimli ise bu, **exec -l bash** komutu ile bir giriş kabuğu başlatmakla eşdeğerdir. Kabuk etkileşimsiz ise, giriş kabuğu başlatma dosyaları çalıştırılacaktır. **exec bash --login** komutu o anki kabuğu bir Bash giriş kabuğu ile değiştirecektir. Bir giriş kabuğunun özel davranışlarının açıklamaları için *Bash Başlatma Dosyaları* (sayfa: 69) bölümüne bakınız.

`-r`
Kabuğu *sınırlı kabuk* (sayfa: 80) yapar.

`-s`
Bu seçenek verilmişse veya seçenek işlemlerinden geriye argüman kalmıyorsa, komutlar standart girdiden okunur. Bu seçenek bir etkileşimli kabuk çağrılırken konumsal parametrelerin atanmasına izin verir.

`-D`
`$` ile öncelenmiş çift tırnak içine alınmış bütün dizgelerin bir listesi standart çıktıya basılır. Bunlar yerel C ya da POSIX değilken yerel dile çeviriye konu dizgelerdir (*Yerele Özel Çeviri* (sayfa: 14) bölümüne bakınız). Bu seçenek hiçbir komutun çalıştırılmamasına sebep olan `-n` seçeneğini uygular.

`[-+]O [`*shopt_seçeneği*`]`
: *shopt_seçeneği*, **shopt** (sayfa: 52) yerleşiği tarafından kabul edilen kabuk seçeneklerinden biridir. *shopt_seçeneği* verilmişse, `-O` seçeneği etkinleştirir; `+O` ise seçeneği kaldırır. *shopt_seçeneği* verilmezse **shopt** tarafından kabul edilen kabuk seçeneklerinin isimleri ve değerlerinin tümü standart çıktıya basılır. Çağrı seçeneği `+O` ise çıktı, girdi olarak yeniden kullanılabilmesini sağlayacak biçemde basılır.

`--`
: `--` seçeneklerin sonunu gösterir ve bundan sonra verilen seçenekler yoksayılır. `--` den sonraki argümanlar ise dosyaisimleri ve argümanlar olarak ele alınır.

Bir giriş kabuğu, sıfırıncı argümanının ilk karakteri `-` olan veya `--login` seçeneği ile çağrılan bir kabuktur.

Bir etkileşimli kabuk `-c` seçeneği belirtilmeksizin, `-s` verilmedikçe seçenek olmayan argümanlar olmaksızın başlatılan ve hem girdisi hem de çıktısı uçbirimlere bağlı olan (**isatty**(3) ile saptanır) veya `-i` seçeneği ile başlatılan bir kabuktur. Daha fazla bilgi için *Etkileşimli Kabuklar* (sayfa: 70) bölümüne bakınız.

Seçeneklerin işlenmesinden sonra argümanlar kalıyorsa ve ne `-c` ne de `-s` seçenekleri verilmişse, ilk argüman kabuk komutlarını içeren bir dosya ismi olarak kabul edilir (*Kabuk Betikleri* (sayfa: 35) bölümüne bakınız). Bu durumda, `$0` dosyanın ismi ve kalan argümanlar da konumsal parametreleridir. Bash komutları bu dosyadan okur, çalıştırır ve çıkar. Bash'in çıkış durumu betikte son çalıştırılan komutun çıkış durumudur. Hiçbir komut çalıştırılmamışsa çıkış durumu sıfırdır.

2. Bash Başlatma Dosyaları

Bu bölümde Bash'in kendi başlatma dosyalarını nasıl çalıştırdığı açıklanmıştır. Herhangi bir dosya varsa ama okunamıyorsa Bash bir hata raporlar. Yaklaşık karakterleri dosya isimlerinde *Yaklaşık (~) Yorumlaması* (sayfa: 23) bölümünde açıklandığı gibi yorumlanır.

Etkileşimli kabuklar *Etkileşimli Kabuklar* (sayfa: 70) bölümünde açıklanmıştır.

Bir etkileşimli giriş kabuğu olarak veya `--login` seçeneği ile çağrıldığında
: Bash bir etkileşimli giriş kabuğu olarak veya `--login` seçeneği ile bir etkileşimsiz kabuk olarak çağrıldığında, `/etc/profile` dosyası varsa dosyayı okur ve komutlarını çalıştırır. Bu dosya okunduktan sonra sırayla `~/.bash_profile`, `~/.bash_login`, ve `~/.profile` dosyalarının varlığına bakar, var ve okunabilir olanlarını okur ve komutlarını çalıştırır. `--noprofile` seçeneği ile başlatarak Bash'in bu davranışı engellenebilir.

  Bir giriş kabuğu çıkarken, `~/.bash_logout` dosyasına bakar, varsa ve okunabilirse dosyayı okur ve komutlarını çalıştırır.

Etkileşimli ancak giriş kabuğu olmayan bir kabuk olarak çağrıldığında
: Giriş kabuğu olmayan bir etkileşimli kabuk başlatıldığında Bash, varsa `~/.bashrc` dosyasını okur ve komutlarını çalıştırır. Bu işlem `--norc` seçeneği ile engellenebilir. `--rcfile` *dosyaismi* seçeneği ile `~/.bashrc` dosyası yerine *dosyaismi* dosyasının okunması ve komutlarının çalıştırılması sağlanabilir.

  Öyleyse, her girişe özel ilklendirme sonrası (ya da öncesi) için

```
if [ -f ~/.bashrc ]; then . ~/.bashrc; fi
```

  satırı `~/.bash_profile` dosyanızda bulunmalıdır.

Etkileşimsiz olarak çağrıldığında
: Bash, örneğin bir betiği çalıştırmak için etkileşimsiz olarak başlatıldığında, ortamdaki `BASH_ENV` değişkenine bakar, varsa değerini okunup çalıştırılacak bir dosya ismi olarak yorumlar. Bash burada,

```
if [ -n "$BASH_ENV" ]; then . "$BASH_ENV"; fi
```

satırı çalıştırılmış gibi davranır, ancak PATH değişkeninin değeri dosyayı aramak için kullanılmaz.

Yukarıda da bahsedildiği gibi, --login seçeneği ile başlatılan bir etkileşimsiz kabuk, komutları giriş kabuğunun başlatma dosyalarından okuyup çalıştırmaya uğraşır.

sh ismiyle çağrıldığında

Bash, **sh** ismiyle çağrılırsa, **sh**'ın tarihsel sürümlerinin başlatma davranışlarını POSIX standardına da uydurmaya çalışarak mümkün olduğunca taklit etmeyi dener.

Bir etkileşimli giriş kabuğu olarak veya --login seçeneği ile bir etkileşimsiz kabuk olarak çağrıldığında, komutları önce /etc/profile dosyasından okumayı ve çalıştırmayı dener ve sırasıyla ~/.bash_profile, ~/.bash_login, ve ~/.profile dosyalarının varlığına bakar, var ve okunabilir olanlarını okur ve komutlarını çalıştırır. --noprofile seçeneği ile başlatarak bu davranış engellenebilir. Bir etkileşimli kabuk olarak **sh** ismiyle çağrıldığında Bash, ortamdaki ENV değişkenine bakar, varsa değerini okunup çalıştırılacak bir dosya ismi olarak yorumlar.

sh olarak çağrılan bir kabuk başka bir başlatma dosyasını okuyup komutlarını çalıştırmayı denemez, zaten --rcfile seçeneğinin de bir etkisi yoktur.

sh olarak çağrıldığında, Bash başlangıç dosyalarını okuduktan sonra POSIX kipine girer.

POSIX kipinde çağrıldığında

Bash --posix komut satırı seçeneği ile POSIX kipinde başlatıldığında, başlangıç dosyaları için POSIX standardını izler. Bu kipte etkileşimli kabuklar ENV değişkenini yorumlayarak bir dosya ismi elde eder ve komutları bu dosyadan okuyup çalıştırır. Bundan başka başlatma dosyasına bakılmaz.

Uzaktan erişilebilir kabuk süreci olarak çağrıldığında

Bash **rshd** gibi bir uzak kabuk süreci tarafından mı çalıştırılmak istendiğini saptamaya çalışır. Bash, **rshd** tarafından çalıştırılmak istendiğini saptadığında, komutları ~/.bashrc dosyası var ve okunabilirse oradan okuyup çalıştırır. **sh** olarak çağrıldığında bunu yapmaz. --norc seçeneği bu davranışı engellemekte kullanılabilir ve --rcfile seçeneği ile başka bir dosyadan komutların okunup çalıştırılması sağlanabilir, ancak **rshd** genellikle kabuğu bu seçeneklerle çağırmaz.

Farklı Etkin ve Gerçek UID/GID ile çağrıldığında

Bash başlatıldığında etkin kullanıcı (grup) kimliği ile gerçek kullanıcı (grup) kimliği aynı değilse, hiçbir başlatma dosyası okunmaz, kabuk işlevleri ortamdan miras alınmaz, ortamda görünse bile SHELLOPTS değişkeni yoksayılır ve etkin kullanıcı kimliğine gerçek kullanıcı kimliği atanır. Çağrı sırasında -p seçeneği verilmişse etkin kullanıcı kimliğinin gerçek kullanıcı kimliğine ayarlanması dışında başlatma davranışı aynıdır.

3. Etkileşimli Kabuklar

3.1. Etkileşimli Kabuk Nedir?

Bir etkileşimli kabuk -c seçeneği belirtilmeksizin, -s verilmedikçe seçenek olmayan argümanlar olmaksızın başlatılan ve hem girdisi hem de hata çıktısı uçbirimlere bağlı olan (**isatty**(3) ile saptanır) veya -i seçeneği ile başlatılan bir kabuktur.

Bir etkileşimli kabuk genellikle kulanıcının uçbirimini okur ve oraya yazar.

Bir etkileşimli kabuk başlatılırken, konumsal parametrelerin atanması -s çağrı seçeneği kullanılarak sağlanabilir.

3.2. Bu Kabuk Etkileşimli mi?

Bash'in etkileşimli olup olmadığını bir başlatma betiği içinde saptamak için - özel parametresinin değerine bakılır. `$- i` harfini içeriyorsa kabuk etkileşimlidir. Örneğin:

```
case "$-" in
*i*)    echo Bu kabuk etkileşimli ;;
*)      echo Bu kabuk etkileşimli değil ;;
esac
```

Ayrıca, başlatma betiklerinde `PS1` değişkeninin varlığına bakarak da bu saptama yapılabilir. `PS1` atanmamışsa kabuk etkileşimsizdir, atanmışsa etkileşimlidir. Örnek:

```
if [ -z "$PS1" ]; then
        echo Bu kabuk etkileşimli değil
else
        echo Bu kabuk etkileşimli
fi
```

3.3. Etkileşimli Kabuk Davranışı

Bir kabuk etkileşimli çalışıyorsa davranışını bir kaç yolla değiştirir.

1. Başlatma dosyaları *Bash Başlatma Dosyaları* (sayfa: 69) bölümünde açıklandığı gibi okunur ve çalıştırılır.
2. *İş denetimi* (sayfa: 83) öntanımlı olarak etkinleştirilir. İş denetimi etkin olduğunda, Bash klavyeden üretilen `SIGTTIN`, `SIGTTOU` ve `SIGTSTP` sinyallerini yoksayar.
3. Bash bir komutun ilk satırını okumadan önce `PS1` değişkenini yorumlar ve gösterir. Çok satırlı komutlarda, ikinci ve müteakip satırları okumadan önce `PS2` değişkenini yorumlar ve gösterir.
4. Bash birincil komut istemi `$PS1` (sayfa: 59)i göstermeden önce `PROMPT_COMMAND` (sayfa: 65) değişkenini bir komut olarak yorumlar ve çalıştırır.
5. Kullanıcının uçbiriminde komutları okumak için *Readline* (sayfa: 87) kullanılır.
6. Bash bir komutu okurken standart girdisinden bir `EOF` aldığında, çıkmak yerine **`set`** (sayfa: 49) yerleşiğinin `-o` seçeneği ile etkinleştirilen `ignoreeof` seçeneğinin değerine bakar.
7. *Komut geçmişi* (sayfa: 111) ve *geçmiş yorumlaması* (sayfa: 113) öntanımlı olarak etkindir. Bir etkileşimli kabuk çıkarken, Bash komut geçmişini `$HISTFILE` değişkenindeki dosyaya kaydeder.
8. *Takma ad yorumlaması* (sayfa: 75) öntanımlı olarak uygulanır.
9. Bir *sinyal kapanının* (sayfa: 35) yokluğunda, Bash `SIGTERM` sinyalini yoksayar.
10. Bir *sinyal kapanının* (sayfa: 35) yokluğunda `SIGINT` yakalanır ve gereği yapılır. `SIGINT` bazı kabuk yerleşiklerine *kesme uygular* (sayfa: 142).
11. `hupoxexit` kabuk seçeneği etkinse, bir etkileşimli kabuk çıkarken tüm işlere bir `SIGHUP` *sinyali gönderir* (sayfa: 35).
12. `-n` çağrı seçeneği yoksayılır yani **`set`** `-n` (sayfa: 49) etkisizdir.
13. Bash postayı `MAIL`, `MAILPATH` (sayfa: 59) ve `MAILCHECK` (sayfa: 64) değişkenlerinin değerlerine bağlı olarak belli aralıklarla kontrol eder.
14. **`set`** `-u` (sayfa: 49) etkinleştirildikten sonra atanmamış kabuk değişkenlerinden dolayı oluşan yorumlama hataları kabuğun çıkmasına sebep olmaz.
15. `${`*parametre*`:?`*sözcük*`}` (sayfa: 24) yorumlaması içindeki *parametre*'nin atanmamış ya da boş olmasından dolayı oluşan yorumlama hatalarında kabuk çıkmayacaktır.
16. Kabuk değişkenleri tarafından saptanan *yönlendirme* (sayfa: 29) hataları kabuğun çıkmasına sebep olmaz.
17. *POSIX kipinde* (sayfa: 80) çalışırken bir *özel yerleşiğin* (sayfa: 58) bir hata durumu döndürmesi kabuğun çıkmasına sebep olmayacaktır.
18. Bir başarısız **`exec`** (sayfa: 39) kabuğun çıkmasına sebep olmayacaktır.

19. Çözümleyici sözdizimi hataları kabuğun çıkmasına sebep olmayacaktır.
20. **cd** yerleşiğine verilen dizinler için *basit yazım düzeltmeleri* (sayfa: 53) öntanımlı olarak etkindir.
21. Kabuk `$PS1` komut istemini bastıktan sonra `TMOUT` (sayfa: 66) değişkenin değerinde belirtilen süre kadar komut bekler, bu süre sonunda bir komut okunmamışsa çıkar.

4. Bash Koşullu İfadeleri

Koşulu ifadeler **[[** *birleşik deyimi* (sayfa: 18) ile **test** *ve* **[** *yerleşikleri* (sayfa: 41) tarafından kullanılır.

İfadeler tek terimli ya da iki terimli olabilir. Tek terimli ifadeler çoğunlukla bir dosyanın durumunu saptamakta kullanılır. Dizge işleçleri ve sayısal karşılaştırma işleçleri de vardır. Dosya argümanı ilklerden birine `/dev/fd/`*N* biçeminde verilmişse *N* dosya tanıtıcısı kontrol edilir, `/dev/stdin`, `/dev/stdout` veya `/dev/stderr` olarak verilmişse dosya tanıtıcı olarak sırayla 0, 1, 2 kontrol edilir.

Aksi belirtilmedikçe, dosyalar üzerinde işlem yapan ilkler sembolik bağları izler ve bağın kendisinde değil hedefi üzerinde işlem yapar.

`-a` *dosya*
: *dosya* varsa doğrudur.

`-b` *dosya*
: *dosya* varsa ve bloka özelse doğrudur.

`-c` *dosya*
: *dosya* varsa ve karaktere özelse doğrudur.

`-d` *dosya*
: *dosya* varsa ve bir dizinse doğrudur..

`-e` *dosya*
: *dosya* varsa doğrudur.

`-f` *dosya*
: *dosya* varsa ve normal bir dosyaysa doğrudur.

`-g` *dosya*
: *dosya* varsa ve grup kimliği biti 1 ise doğrudur.

`-h` *dosya*
: *dosya* varsa ve bir sembolik bağ ise doğrudur.

`-k` *dosya*
: *dosya* varsa ve yapışkan biti 1 ise doğrudur.

`-p` *dosya*
: *dosya* varsa ve bir isimli boruhattı (FIFO) ise doğrudur.

`-r` *dosya*
: *dosya* varsa ve okunabilir ise doğrudur.

`-s` *dosya*
: *dosya* varsa ve uzunluğu sıfırdan büyükse doğrudur.

`-t` *fd*
: *fd* dosya tanıtıcısı açık ve bir uçbirime karşılıksa doğrudur.

`-u` *dosya*
: *dosya* varsa ve kullanıcı kimliği biti 1 ise doğrudur.

`-w` *dosya*
: *dosya* varsa ve yazılabilir ise doğrudur.

`-x` *dosya*
: *dosya* varsa ve çalıştırılabilir ise doğrudur.

`-O` *dosya*
: *dosya* varsa ve sahibi etkin kullanıcı kimlik ise doğrudur.

`-G` *dosya*
: *dosya* varsa ve sahibi etkin grup kimlik ise doğrudur.

`-L` *dosya*
: *dosya* varsa ve bir sembolik bağ ise doğrudur.

`-S` *dosya*
: *dosya* varsa ve bir soket ise doğrudur.

`-N` *dosya*
: *dosya* varsa ve son okunduğundan beri değiştirilmişse doğrudur.

dosya1 `-nt` *dosya2*
: *dosya1*, *dosya2* den değişiklik tarihine göre daha yeni ise ya da *dosya1* mevcutken *dosya2* yoksa doğrudur.

dosya1 `-ot` *dosya2*
: *dosya1*, *dosya2* den daha eski ise ya da *dosya2* mevcutken *dosya1* yoksa doğrudur.

dosya1 `-ef` *dosya2*
: *dosya1* ile *dosya2* aynı aygıt ve aynı dosya düğümünü gösteriyorsa doğrudur.

`-o` *sçn–ismi*
: *Kabuk seçeneği* (sayfa: 49) *sçn–ismi* etkinse doğrudur.

`-z` *dizge*
: *dizge* uzunluğu sıfırsa doğrudur.

`-n` *dizge*
dizge
: *dizge* sıfırdan farklıysa doğrudur.

dizge1 `==` *dizge2*
: *dizge*'ler eşitse doğrudur. Katı POSIX uyumluluğu için `==` yerine `=` kullanılabilir.

dizge1 `!=` *dizge2*
: *dizge*'ler aynı değilse doğrudur.

dizge1 `<` *dizge2*
: *dizge1* yerele göre sıralamada *dizge2* den önceyse doğrudur.

dizge1 `>` *dizge2*
: *dizge1* yerele göre sıralamada *dizge2* den sonraysa doğrudur.

arg1 `-eq` *arg2*

arg1 ile *arg2* eşitse doğrudur. Argümanlar tamsayı olmalıdır.

arg1 `-ne` *arg2*
arg1 ile *arg2* farklıysa doğrudur. Argümanlar tamsayı olmalıdır.

arg1 `-lt` *arg2*
arg1, *arg2* den küçükse doğrudur. Argümanlar tamsayı olmalıdır.

arg1 `-le` *arg2*
arg1, *arg2* den küçük ya da ona eşitse doğrudur. Argümanlar tamsayı olmalıdır.

arg1 `-gt` *arg2*
arg1, *arg2* den büyükse doğrudur. Argümanlar tamsayı olmalıdır.

arg1 `-ge` *arg2*
arg1, *arg2* den büyük ya da ona eşitse doğrudur. Argümanlar tamsayı olmalıdır.

5. Kabuk Aritmetiği

Kabuk, *kabuk yorumlarından biri olarak* (sayfa: 22) veya **`let`** (sayfa: 47) yerleşiğinde ve **`declare`** yerleşiğinin **`-i`** seçeneğiyle aritmetik ifadelerin değerlendirilmesini mümkün kılar.

Değerlendirme sabit genişlikli tamsayılarla, taşma denetimi uygulanmaksızın yapılır, sıfırla bölme bir hata olarak bayraklanır ve onun için bir sinyal kapanı vardır. İşleçler için öncelikler ve çağrışımsallık ve değerler C dilindeki ile aynıdır. Aşağıdaki işleçlerin listesi eşit öncelikli işleç düzeylerine göre öbeklenmiştir.

`id++ id--`
değişken son–arttırma ve son–eksiltme

`++id --id`
değişken ön–arttırma ve ön–eksiltme

`- +`
tek terimli eksi ve artı

`! ~`
mantıksal ve bitdüzeyi zıtlık

`**`
üstel

`* / %`
çarpma, bölme, kalan

`+ -`
toplama, çıkartma

`<< >>`
sol ve sağ bitdüzeyi kaydırmalar

`<= >= < >`
karşılaştırma

`== !=`
eşitlik ve farklılık

`&`

bitdüzeyi VE

`^`
bitdüzeyi ayrıcalıklı VEYA (XOR)

`|`
bitdüzeyi VEYA

`&&`
mantıksal VE

`||`
mantıksal VEYA

`ifade ? ifade : ifade`
koşullu değerlendirme

`= *= /= %= += -= <<= >>= &= ^= |=`
atama

`ifade1 , ifade2`
virgül

Kabuk değişkenlerinin terimler olarak kullanılması sağlanmıştır. Parametre yorumlaması ifade değerlendirilmeden önce uygulanır. Bir ifade içinde, kabuk değişkenleri parametre yorumlamasının sözdizimini kullanmaksızın doğrudan ismiyle yer alabilir. boş değerli ya da **unset** ile kaldırılmış bir kabuk değişkenine parametre yorumlamasının sözdizimini kullanmaksızın doğrudan ismiyle başvurulduğunda değeri 0 olarak değerlendirmeye sokulur. Başvurulduğunda bir değişkenin değeri bir aritmetik ifade olarak değerlendirilir ya da tamsayı niteliği verilmiş bir değişkene **declare -i** kullanılarak bir değer atanır. Boş değer 0 olarak değerlendirilir. Bir kabuk değişkeni ifade içinde kullanılırken tamsayı niteliğinin etkinleştirilmesini gerektirmez.

0 ile başlayan sabitler sekizlik sayılar olarak, `0x` veya `0X` ile başlayanlar onaltılık sayılar olarak ele alınır. Bunlar dışında kalan sayılar tabanları ile birlikte `[`*taban*`#]`*n* biçeminde gösterilebilir. Burada *taban* 0 ile 64 arasında bir tamsayı olabilir. *n* ise tabandaki sayıdır. 10 tabanındaki sayılar için *base#* kısmı verilmeyebilir. 9 dan büyük sayılar için sırasıyla küçük harfler, büyük harfler, `@` ve _ kullanılır. Eğer *taban* küçük ya da eşit 36 ise küçük ve büyük harfler 10 ile 35 arasındaki rakamlar için birbirinin yerine kullanılabilir.

İşleçler öncelik sırasına göre değerlendirilir. Parantez içine alınmış alt ifadeler öncelikle değerlendirilir. Bu nedenle parantez içine alma işleçlerin önceliklerini arttırmak amacıyla kullanılabilir.

6. Takma Adlar

Takma adlar bir basit komutun ilk sözcüğü olarak kullanıldığında, bir sözcüğün yerini bir dizgenin almasını mümkün kılar. Kabuk, **alias** (sayfa: 43) ve **unalias** (sayfa: 58) yerleşikleri ile atanan ve kaldırılan takma adları bir listede tutar.

Her basit komutun ilk sözcüğü, tırnak içine alınmamışsa, bir takma ada sahip mi diye bakılır. Sahipse, bu sözcük takma adın değeri ile değiştirilir. **/**, **$**, , **=**, kabuk meta karakterleri veya tırnak karakterleri, önceleme karakteri bir takma adın isim parçasında bulunamaz. Değer parçasında ise kabuk meta karakterleri dahil kabuk girdisi olarak geçerli tüm karakterler bulunabilir. Değerin ilk sözcüğü takma adlar için denenir, ama yorumlanacak bir takma isimle aynı olan bir sözcük ikinci kez yorumlanmaz. Bunu bir örnekle açıklayacak olursak,

```
ls='ls -F'
```

ls hem bir takma ad ismi hem de takma adın değerindeki ilk sözcüktür. Değerin ilk sözcüğü ikinci kez bir takma ad ismi olarak tekrar yorumlanmaz, yani takma adlar iç içe olamazlar. Takma ad değerinin son karakteri bir

boşluk ya da sekme karakteri ise, sonraki komut sözcüğünü izleyen takma ad ayrıca takma ad yorumlaması için kontrol edilir.

Takma adlar **alias** komutu ile oluşturulur ve listelenir; **unalias** komutu ile kaldırılır.

csh'daki gibi takma ad değerinde argümanları kullanmak için bir mekanizma yoktur. Argümanlar gerekliyse bir *kabuk işlevi* (sayfa: 20) kullanılmalıdır.

Kabuk etkileşimsizken `expand_aliases` (sayfa: 54) kabuk seçeneği etkin olmadıkça takma adlar yorumlanmaz.

Takma adların tanımlanması ve kullanımı ile ilgili kurallar bir parça kafa karıştırıcıdır. Bash komut satırındaki bir komutu çalıştırmadan önce daima en azından bir tam satır okur. Takma adlar bir komut çalıştırıldığı zaman değil, okunduğu zaman yorumlanır. Öte yandan, diğer komut olarak aynı satırda görünen bir takma ad tanımı girdinin sonraki satırı okunana kadar etkisizdir. Bu satırdaki komutlardan sonraki takma ad tanımları yeni takma addan etkilenmez. Bu davranış ayrıca işlevler çalıştırılırken de görülür. Takma adlar bir işlev çalıştırıldığı zaman değil, okunduğu zaman yorumlanır, çünkü bir işlev tanımının kendisi bir birleşik komuttur. Bir sonuç olarak, bir işlev içinde tanımlanmış olan takma adlar işlevin çalıştırılmasının sonrasına kadar kullanışsızdır. Emin olmak için, daima takma ad tanımlarını ayrı bir satıra koyun ve **alias**'ı birleşik komut içinde kullanmayın.

Hemen her maksat için, kabuk işlevleri takma adlara tercih edilir.

7. Diziler

Bash tek boyutlu dizi değişkenleri sağlar. Her değişken dizi olarak kullanılabilir; **declare** (sayfa: 45) yerleşiği bir diziyi doğrudan bildirecektir. Bir dizi için ne bir azami boyut vardır ne de üyelerinin indislenmesi ya da peşpeşe atanması gereklidir. Dizilerin ilk indisi sıfırdır.

Bir değişken aşağıdaki sözdizimi kullanılarak atanırsa, bir dizi otomatikman oluşturulmuş olur:

isim`[`*indis*`]`=*değer*

indis değerlendirildiğinde sıfıra eşit ya da büyük bir sayı olması gereken bir aritmetik ifade olarak ele alınır. Bir dizi doğrudan aşağıdaki gibi bildirilebilir:

declare `-a` *isim*

Ayrıca bu söz dizimi de kabul edilir:

declare `-a` *isim*`[`*indis*`]`

indis yoksayılır. Bir dizi değişkeni için öznitelikler **declare** ve **readonly** (sayfa: 40) yerleşikleri ile belirtilebilir. Her öznitelik dizinin tüm üyelerine uygulanır.

Diziler aşağıdaki gibi birleşik atamalar kullanılarak atanabilir.

isim=`(`*değer1 ... değerN*`)`

Burada her *değer*, `[`*indis*`]=`*dizge* biçemindedir. İsteğe bağlı olan *indis* verilirse, indis atanmış olur; aksi takdirde atanan elemanın indisi deyim tarafından atanan son indis artı birdir. İndisleme sıfırdan başlar. Bu sözdizimi ayrıca **declare** yerleşiği tarafından da kabul edilir. Tek tek dizi elemanları *isim*`[`*indis*`]`=*değer* sözdizimi kullanılarak atanabilir.

Dizideki her eleman `${`*isim*`[`*indis*`]}` biçeminde bir ifade içinde kullanılabilir. Kaşlı ayraçlar, kabuğun dosyaismi yorumlama işleci ile çelişmemesi için gereklidir. *indis* olarak `@` veya `*` verilirse, dizinin tüm üyeleri anlamındadır. Bu indisler sadece çift tırnak içine alındığında farklıdır. Sözcük çift tırnaklar arasındaysa, `${`*isim*`[*]}` sözcüğü, her dizi üyesi `IFS` değişkeninin ilk karakteri ile ayrılarak tek sözcük olarak yorumlanır. `${`*isim*`[@]}` sözcüğünde ise dizinin her elemanı ayrı bir sözcük olarak yorumlanır. Dizinin hiç elemanı yoksa, `${`*isim*`[@]}` hiçbir şey

olarak yorumlanır. Bir sözcük içinde çift tırnaklı yorumlama olursa ilk parametrenin yorumu özgün sözcüğün başlangıç parçası ile birleşir ve son parametrenin yorumu da özgün sözcüğün son parçası ile birleşir. Bu, `@` ve `*` özel parametrelerinin yorumlanmasına benzerdir. `${#`*isim*`[`*indis*`]}`, `${`*isim*`[`*indis*`]}` dizisinin uzunluğu olarak yorumlanır. *indis*, `@` veya `*` ise yorumlama dizideki elemanların sayısıdır. Bir dizi değişkeni bir ifade içinde kullanılırken bir indis belirtilmezse sıfırıncı elemana karşılıktır.

Dizileri kaldırmak için **unset** (sayfa: 43) yerleşiği kullanılır. **unset** *isim*`[`*indis*`]` komutu *indis* indisli elemanı diziden kaldırır. Dosyaismi üretiminin sebep olacağı istenmeyen yan etkilerden kaçınılmaya çalışılmalıdır. **unset** *isim* komutunda *isim* bir dizi ismi ise dizinin tamamı kaldırılır. Ayrıca *indis* olarak `*` veya `@` verildiğinde de dizinin tamamı kaldırılır.

declare (sayfa: 45), **local** (sayfa: 47) ve **readonly** (sayfa: 40) yerleşiklerinin herbirinde bir dizi belirtmek için `-a` seçeneği bulunur. **read** yerleşiğinin `-a` seçeneği ile standart girdiden okunan sözcüklerin bir listesi bir diziye atanabilir ve değerler standart girdiden okunup tek tek dizi elemanlarına atanabilir. **set** (sayfa: 49) ve **declare** (sayfa: 45) yerleşikleri girdi olarak yeniden kullanılabilir şekilde dizi değerlerini gösterebilir.

8. Dizin Yığını Yerleşikleri

Dizin yığını son ziyaret edilen dizinlerin bir listesidir. Dizinleri yığına eklemek için **pushd** yerleşiği kullanılır ve eklenen dizine geçilmesini sağlar. Dizinleri yığından kaldırmak için **popd** yerleşiği kullanılır ve yığının tepesindeki dizine geçilmesini sağlar. Dizin yığınındaki dizinleri göstermek için de **dirs** yerleşiği kullanılır.

Dizin yığınının içeriğine ayrıca bir dizi değişkeni olan `DIRSTACK` kabuk değişkeninden de erişilebilir.

8.1. Dirs Yerleşiği

dirs `[`**+***N* `|` **-***N*`]` `[`**-clpv**`]`

O an hatırlanan dizinlerin listesini gösterir. Dizinler **pushd** komutu ile listeye eklenir ve **popd** komutu ile listeden kaldırılır.

+*N*
: Seçeneksiz çağrılan **dirs** tarafından basılan listenin solundaki kayıt 0 kabul edilerek *N*. dizini gösterir.

-*N*
: Seçeneksiz çağrılan **dirs** tarafından basılan listenin sağındaki kayıt 0 kabul edilerek *N*. dizini gösterir

`-c`
: Tüm elemanlarını silerek dizin yığınını temizler.

`-l`
: Daha uzun liste üretir; öntanımlı liste biçeminde ev dizini bir yaklaşık karakteri ile gösterilir.

`-p`
: Dizin yığınını her satırda tek girdiyle gösterir.

`-v`
: `-p` gibidir, farklı olarak her satırın başına girdinin yığındaki indisi konur.

8.2. Popd Yerleşiği

popd `[`**+***N* `|` **-***N*`]` `[`**-n**`]`

Argümansız verildiğinde dizin yığınının tepesindeki girdiyi kaldırdıktan sonra yığının tepesindeki dizine **cd** uygular. **dirs** ile listelenen ilk dizin 0 olmak üzere tüm elemanlar 0 dan başlayarak numaralanır. Bu durumda, **popd** komutu **popd** `+0` komutuna eşdeğerdir.

+*N*
: Seçeneksiz çağrılan **dirs** tarafından basılan listenin solundaki kayıt 0 kabul edilerek *N*. dizini siler.

-*N*
: Seçeneksiz çağrılan **dirs** tarafından basılan listenin sağındaki kayıt 0 kabul edilerek *N*. dizini siler.

-n
: Dizinler yığından kaldırılırken, dizin değiştirilmesini engeller. Böylece sadece yığın değişmiş olur.

8.3. Pushd Yerleşiği

pushd [*dizin* | **+***N* | **-***N*] [**-n**]

İçinde bulunulan dizini dizin yığınının tepesine kaydettikten sonra belirtilen *dizin*'e geçilir. Argümansız kullanıldığında yığının tepesindeki iki dizin yer değiştirir.

+*N*
: Seçeneksiz çağrılan **dirs** tarafından basılan listenin solundaki kayıt 0 kabul edilerek *N*. dizini, listeyi döndürerek yığının tepesine getirir.

-*N*
: Seçeneksiz çağrılan **dirs** tarafından basılan listenin sağındaki kayıt 0 kabul edilerek *N*. dizini, listeyi döndürerek yığının tepesine getirir.

-n
: Dizinler yığına eklerken, dizin değiştirilmesini engeller. Böylece sadece yığın değişmiş olur.

dizin
: İçinde bulunulan dizini yığının tepesi yapar ve **cd** *dizin* komutuna eşdeğer bir işlemle *dizin*'e geçilmesini sağlar.

9. Komut İsteminin Kontrol Edilmesi

PROMPT_COMMAND değişkeninin değeri Bash birincil komut istemini basmadan hemen önce saptanır. PROMPT_COMMAND atanmış ve boş olmayan bir değere sahipse, bu değeri, komut satırında yazıldığı gibi çalıştırılır.

Komut istemi değişkenlerinde görünen özel karakterler:

\a
: Sesli uyarı (bell) karakteri.

\d
: Tarih, İng. olarak "Günismi Ayismi AyınGünü" biçemindedir (örn, "Thu Apr 24").

\D{*biçim*}
: *biçim* **strftime**(3)'a aktarılır ve sonuç istem dizgesine yerleştirilir; *biçim* boş ise sonuç yerele özel zaman gösteriminde olacaktır. *biçim* boş bile olsa kaşlı ayraçlar gereklidir.

\e
: Önceleme (esc) karakteri.

\h
: İlk . ya kadar, konak ismi.

`\H`
: Tam konak ismi.

`\j`
: Kabuk tarafından o an yönetilmekte olan işlerin sayısı.

`\l`
: Kabuğun uçbirim aygıtının dosya ismi (örn, `/dev/tty2` için `tty2`).

`\n`
: Satırsonu (LF) karakteri.

`\r`
: Satırbaşı (CR) karakteri.

`\s`
: Kabuğun ismi ya da başka bir deyişle `$0`'ın değeri (son `/` den sonraki kısım).

`\t`
: 24 saatlik gösterimle SS:MM:ss biçeminde saat.

`\T`
: 12 saatlik gösterimle SS:MM:ss biçeminde saat.

`\@`
: 12 saatlik gösterimle öö/ös biçeminde saat.

`\A`
: 24 saatlik gösterimle SS:MM biçeminde saat.

`\u`
: O anki kullanıcının kullanıcı ismi.

`\v`
: Bash'in sürümü (örn, 2.00)

`\V`
: Bash'in "dağıtım.sürüm.yamaseviyesi" olarak sürümü (örn, 2.05.9)

`\w`
: O anki dizin; `$HOME` kısaltılmış olarak ~ ile gösterilir.

`\W`
: `$PWD` değişkenindeki dizinin son / dan sonraki kısmı; `$HOME` kısaltılmış olarak ~ ile gösterilir.

`\!`
: Bu komutun geçmiş numarası.

`\#`
: Bu komutun komut numarası.

`\$`
: Etkin kullanıcı kimliği 0 ise `#`, değilse `$`.

`\`*nnn*
: Sekizlik ASCII değeri *nnn* olan karakter.

`\\`

Tersbölü.

`\[`
: Basılamayan karakterler dizisi başlatır. Bu, komut istemine bir uçbirim denetim dizgesi girmek için yararlıdır.

`\]`
: Basılamayan karakterler dizisini sonlandırır. (Örneğin `PS1=$'\[\\033[1;31m\]\\w$ '` kırmızı renkte bir komut satırı ayarlar)

Komut numarası ile geçmiş numarası farklı kavramlardır: Bir komutun geçmiş numarası onun geçmiş listesindeki konumudur (*Bash'in Geçmişsel Yetenekleri* (sayfa: 111) bölümüne bakınız). Komut numarası ise o anki kabuk oturumunun başlangıcından itibaren çalıştırılan komut sayısıdır.

Dizgenin kodu çözüldükten sonra `promptvars` (sayfa: 56) kabuk seçeneğinin değerine konu, *parametre yorumlaması* (sayfa: 24), *komut ikamesi* (sayfa: 26), *aritmetik yorumlama* (sayfa: 27) ve *tırnak kaldırma* (sayfa: 29) uygulanır.

10. Sınırlı Kabuk

Bash **rbash** ile başlatılırsa ya da çağrı sırasında `--restricted` veya `-r` seçeneği verilirse, kabuk sınırlı duruma gelir. Bir sınırlı kabuk, standart kabuktan daha kontrollü bir ortam sağlamak için kullanılır. Bir sınırlı kabuk aşağıdaki **yasaklar** (belirtilse de uygulanmazlar) dışında **bash**'e özdeş davranır:

- **cd** yerleşiği ile dizin değiştirmek.
- `SHELL`, `PATH`, `ENV` veya `BASH_ENV` değişkenlerini kaldırmak ya da değer atamak.
- `/` içeren komut isimleri yazmak.
- `.` yerleşiğine bir argüman olarak `/` içeren bir dosya ismi belirtmek.
- **hash** yerleşiğine `-p` seçeneğinin argümanı olarak `/` içeren bir dosya ismi belirtmek.
- Başlangıçta kabuk ortamından işlev tanımlarını alıntılamak.
- Başlangıçta kabuk ortamındaki `SHELLOPTS` değişkeninin değerini çözümlemek.
- Çıktıyı `>`, `>|`, `<>`, `>&`, `&>` ve `>>` yönlendirme işleçleri ile yönlendirmek.
- **exec** yerleşiğini kullanarak kabuğu başka bir komutla değiştirmek.
- **enable** yerleşiğinin `-f` ve `-d` seçenekleri ile yerleşik komutları eklemek ya da silmek.
- Etkisiz olan kabuk yerleşiklerini **enable** yerleşiği ile etkinleştirmek.
- **command** yerleşiğini `-p` seçeneği ile çalıştırmak.
- **set** `+r` veya **set** `+o restricted` komutları ile sınırlı kipi kapatmak.

Bu sınırlamalar başlangıç dosyalarından herhangi biri okunur okunmaz etkin olur.

Bir komut *kabuk betiği* (sayfa: 35) olarak varsa çalıştırılır, **rbash** bir kabuk betiğini çalıştırmak üzere kabuğu çatalladığında sınırlamaları kapatır.

11. Bash POSIX Kipi

Bash'i `--posix` komut satırı seçeneği ile başlatarak ya da Bash çalışırken **set** `-o posix` komutunu çalıştırarak, Bash'in POSIX standardındakinden farklı olan öntanımlı davranışlarını değiştirerek standarda daha yakın davranması sağlanır.

sh olarak çağrıldığında, Bash başlangıç dosyalarını okuduktan sonra POSIX kipine girer

Aşağıdaki liste 'POSIX kipi' etkinleştirildiğinde değişenleri gösterir:

1. Bir komut komut tablosunda artık yoksa, Bash komutun yeni yerini bulmak için `$PATH` dizinlerini tekrar arar. Bu ayrıca **shopt** `-s checkhash` ile etkinleştirilebilir.
2. Bir iş sıfırdan farklı bir durumla çıktığında iş denetim kodu ve yerleşikleri 'Bitti(durum)' benzeri bir ileti basar.
3. Bir iş durduğunda iş denetim kodu ve yerleşikleri 'Durdu(sinyalismi)' benzeri bir ileti basar. Burada geçen *sinyalismi* örneğin `SIGTSTP` olabilir.
4. **bg** yerleşiği artalana yerleştirilmeyi açıklamayı gerektiren ama işin o anki mi önceki mi olduğunun belirtilmesini içermeyen bir biçim kullanır.
5. Anahtar sözcüklere takma ad verilemez.
6. POSIX `PS1` ve `PS2` geçmiş numarasına `!` yorumlaması ve `!!` için de `!` etkinleştirilir ve `promptvars` kabuk seçeneğinin durumuna aldırmayıp `PS1` ve `PS2` değişkenlerinin değerlerine parametre yorumlaması uygulanır.
7. Etkileşimli açıklamalar öntanımlı olarak etkinleştirilir (Bash zaten onları öntanımlı olarak etkin yapar).
8. Normal Bash dosyaları yerine POSIX başlatma dosyaları çalıştırılır (`$ENV`).
9. Yaklaşık yorumlaması satırdaki tüm atama deyimleri yerine bir komut ismini önceleyen atamalar üzerinde uygulanır.
10. Öntanımlı geçmiş dosyası `~/.sh_history`'dir (bu, `$HISTFILE` değişkeninin öntanımlı değeridir).
11. **kill** `-l` çıktısı tüm sinyal isimlerini boşluklarla ayırarak ve `SIG` önekini kullanılmaksızın tek satıra basar.
12. **kill** yerleşiği `SIG` önekli sinyal isimlerini kabul etmez.
13. **.** *dosyaismi* komutundaki *dosyaismi* bulunamazsa, etkileşimsiz kabuklar çıkar.
14. Bir aritmetik yorumlamasının içindeki bir sözdizimi hatası, bir geçersiz ifadenin içinde sonuçlanıyorsa etkileşimsiz kabuklar çıkar.
15. Yönlendirme işleçleri, kabuk etkileşimli olmadıkça yönlendirmedeki sözcük üstünde dosyaismi yorumlaması uygulamaz.
16. Yönlendirme işleçleri, yönlendirmedeki sözcük üstünde sözcük ayrımlaması uygulamaz.
17. İşlev isimleri geçerli kabuk isimleri olmalıdır. Böylece bu isimler, bir rakam ile başlayamaz ve harfler, rakamlar ile altçizgi karakteri dışında karakterleri içeremez. Bir işlev isminin bir geçersiz isimle bildirilmesi etkileşimsiz kabuklarda bir ölümcül sözdizimi hatası oluşturur.
18. *POSIX'e özel yerleşiklere* (sayfa: 58) komut araması sırasında kabuk işlevlerinden önce bakılır.
19. Bir POSIX özel yerleşiği bir hata durumu ile dönerse bir etkileşimsiz kabuk çıkar. Ölümcül hataların da listelendiği POSIX standardı yanlış seçeneklerin aktarılması gibi şeyler, yönlendirme hataları, komut ismini önceleyen atamalar için değişken atama hataları ve benzerlerini içerir.
20. `CDPATH` atanmışsa, **cd** yerleşiği bulunulan dizini ona doğrudan eklemeyecektir. Bunun anlamı: **cd** yerleşiğine bir argüman olarak verilen isimle aynı isimdeki bir dizin bulunulan dizinde mevcut olsa bile, `$CDPATH` içindeki girdilerden birinden oluşturulabilen geçerli dizin ismi yoksa **cd** yerleşiği başarısız olacaktır.
21. Atama deyimlerinden sonra gelen komut ismi olmadığında, bir değişken atama hatası oluşursa, bir etkileşimsiz kabuk bir hata durumu ile çıkar. Örneğin, bir salt–okunur değişkene bir değer atanmaya çalışılırsa, bir değişken atama hatası oluşur.
22. Bir **for** deyiminin yineleme değişkeni veya bir **select** deyiminin seçim değişkeni bir salt–okunur değişkense bir etkileşimsiz kabuk bir hata durumu ile çıkar.
23. Süreç ikamesi yoktur.
24. POSIX özel yerleşiklerinden önce verilen atama deyimleri komut işini tamamladıktan sonra da kabukta etkin olarak kalır.

25. Kabuk işlevi çağrılarından önce verilen atama deyimleri işlev döndükten sonra da bir önceki maddede olduğu gibi kabukta etkin olarak kalır.
26. **export** ve **readonly** yerleşik komutları çıktılarını POSIX'in gerektirdiği biçemde gösterir.
27. **trap** yerleşiği sinyal isimlerini SIG ile öncelemeden gösterir.
28. **trap** yerleşiği ilk argümanına olası bir sinyal belirtimi var mı acaba diye bakmaz ve eğer varsa argüman yalnızca rakamlardan oluşmadıkça ve geçerli bir sinyal numarası olmadıkça sinyal işlemeyi özgün mecrasına geri döndürür. Eğer kullanıcı sinyal işlemeyi özgün mecrasına geri döndürmek isterse ilk argüman olarak **–** kullanmalıdır.
29. **.** ve **source** yerleşikleri bir dosyaismi argümanınını, PATH aramasında bulamazlarsa, bulunulan dizinde aramazlar.
30. Komut ikamelerini çalıştırmak için çatallanan altkabuklar -e seçeneğinin değerini kendilerini çalıştıran kabuktan miras alırlar. POSIX kipinde değilken, Bash -e seçeneğinin değerini bu altkabuklarda temizler.
31. Etkileşimsiz kabuklarda bile takma ad yorumlaması daima etkindir.
32. **alias** yerleşiği takma ad tanımlarını listelemek amacıyla kullanıldığında -p seçeneği belirtilmedikçe tanımların başlarında '**alias** gösterilmez.
33. **set** yerleşiği seçeneksiz olarak çağrıldığında, kabuk işlevlerinin isimlerini ve tanımlarını göstermez.
34. **set** yerleşiği seçeneksiz olarak çağrıldığında, sonuçlar basılamayan karakterleri içerse bile, kabuk metakarakterlerini içermedikçe, değişken değerlerini tırnaksız olarak gösterir.
35. **cd** yerleşiği mantıksal kipte çağrıldığında ve dosya yolu $PWD'den oluştuğunda ve de bir argüman olarak verilen dizin ismi içinde bulununlan dizini göstermediğinde **cd** fiziksel kipe dönmek yerine başarısız olacaktır.
36. **pwd** yerleşiği -P seçeneği ile çağrıldığında $PWD'nin içeriğini sembolik bağ içermeyen dizinlerle yazar.
37. **pwd** yerleşiği -P seçeneği belirtmek suretiyle dosya sistemini sınaması istenmese bile bastığı değerin, içinde bulunulan dizin ile aynı olup olmadığına bakacaktır.
38. Geçmiş listelenirken **fc** yerleşiği geçmiş girdisinin değiştirilmiş olup olmadığına dair bir belirti içermez.
39. **fc** tarafından öntanımlı metin düzenleyici olarak **ed** kullanılır.
40. **type** ve **command** yerleşikleri çalıştırılabilir olmayan bir dosyayı kabuk çalıştırmaya çalışacak olsa hatta $PATH içinde bu isimde sadece bu dosya varsa bile böyle birşey bulduk diye raporlamazlar.
41. **v** komutu (*Ç.N.:v değil vi olacaktı sanırım*) çalıştırılmak istendiğinde **vi** kipindeyken $FCEDIT ve $EDITOR değişkenlerinin içeriklerine bakılmaksızın doğrudan **vi** metin düzenleyici çalıştırılır.
42. xpg_echo seçeneği etkinken Bash argümanları **echo** seçenekeri olarak yorumlamaya çalışmaz. Her argüman gösterildikten sonra öncelenmiş karakterler dönüştürülür.

POSIX kipindeyken öntanımlı olan ancak Bash'in gerçeklemediği bazı POSIX davranışları vardır. Özellikle:

1. **fc** yerleşiği, geçmiş girdileri düzenleneceği zaman **FCEDIT** ortamda tanımlı değilken öntanımlı metin düzenleyici olarak **ed**'i kullanacağına gidip $EDITOR değişkeninin içeriğine bakar; $EDITOR tanımlı değilse **ed**'i kullanır.
2. Yukarıda dikkat çekildiği gibi, Bash, **echo** yerleşiğinin tam uyumlu olabilmesi için xpg_echo seçeneğinin etkin olmasını gerektirir.
3. Bash kurulum sırasında **configure** betiğine --enable-strict-posix-default seçeneği verilmek suretiyle öntanımlı olarak POSIX uyumlu olmak üzere yapılandırılabilir (bkz, *İsteğe Bağlı Özellikler* (sayfa: 118)).

VII. İş Denetimi

İçindekiler

Bu oylumda, iş denetiminin ne olduğu, nasıl çalıştığı ve onun yeteneklerine Bash ile nasıl erişeceğiniz anlatılmıştır.

1. İş Denetiminin Temelleri

İş denetimi, bir sürecin çalışmasını seçerek durdurma (bekletme) ve daha sonra çalışmasını sürdürme (yeniden başlatma) yeteneği demektir. Bir kullanıcı bu araçla genellikle Bash ve sistemin uçbirim sürücüsü tarafından ortak olarak sağlanan bir etkileşimli arayüz üzerinden çalışır.

Kabuk bir iş ile her boruhattını ilişkilendirir. Halen çalışmakta olan ve **jobs** komutu ile listelenebilen işlerin listesini bir tabloda saklar. Bash bir işi eşzamansız olarak başlattığında şöyle bir ileti gösterir:

```
[1] 25647
```

Bu ileti, işin iş numarasının 1 ve bu işle ilişkilendirilen boruhattındaki son sürecin süreç kimliğinin (PID) 25647 olduğunu ifade eder. Bir tek boruhattındaki süreçlerin tümü aynı işin üyeleridir. Bash iş denetimi için esas olarak iş soyutlaması kullanır.

İş denetiminin kullanıcı arayüzünün gerçeklenmesini kolaylaştırmada, işletim sistemi, kullanılan uçbirim sürecinin grup kimliği diye bir kavram sağlar. Bu süreç grubunun üyeleri (süreçlerin süreç grubu kimliği, kullanılan uçbirimin süreç grubu kimliğine eşittir) klavyeden üretilen `SIGINT` gibi sinyalleri alır. Bu süreçler önalanda olduklarını söylerler. Artalan süreçleri, süreç grubu kimliği uçbiriminkinden farklı olan süreçlerdir. Bu süreçler klavyeden üretilen sinyallerden bağışıktır. Sadece önalan süreçleri uçbirime yazar ve uçbirimi okuyabilir. Uçbirimi okumaya (ya da yazmaya) çalışan artalan süreçleri bir `SIGTTIN` (`SIGTTOU`) sinyali yakalanmadıkça, süreci bekleten uçbirim sürücüsü üzerinden gönderir.

Bash'in çalıştığı işletim sistemi üzerinde iş denetimi destekleniyorsa, Bash onu kullanacak araçları içerir. Bir süreç çalışırken *bekletme karakteri* nin tuşlanması (genelde `^Z`, Control–Z'dir) sürecin durdurulmasına ve denetimin Bash'e geçmesine sebep olur. Gecikmeli bekletme karakterinin (genelde `^Y`, Control–Y'dir) tuşlanması, uçbirimden girdi okumaya çalıştığı zaman sürecin durdurulmasına sebep olur ve denetim Bash'e döndürülür. Bundan sonra kullanıcı bu işin durumunu değiştirebilir: **bg** komutunu kullanarak artalanda, **fg** komutunu kullanark önalanda işi devam ettirebilir ya da **kill** komutu ile süreci öldürebilir. Bir `^Z` hemen etkisini gösterir ve çıktının askıda kalmasına ve baştanyazmanın iptal edilmesine sebep olan bir yan etkisi vardır.

Kabuktaki bir işi ifade etmede kullanılan bir kaç yol vardır. `%` karakteri bir işin ismini ifade eder.

İş numarası `n`'e, `%n` olarak başvurulabilir. `%+` ve `%%` sembolleri, artalanda başlatılan ya da önalanda olan durmuş son işin yani o anki iş kavramının kabuktaki karşılığıdır. Tek bir `%` (iş belirtimine eşlik etmeksizin) ayrıca o anki

işi belirtmek için kullanılabilir. Önceki iş ise `%-` sembolü ile ifade edilir. İşlerle ilgili çıktılarda ise (örn. **jobs** komutunun çıktısı), o anki iş daima bir `+` ile önceki iş ise `-` ile imlenir.

Bir işe onu başlatan ismi bir önek olarak kullanarak ya da komut satırının içinde geçen bir dizge kullanılarak erişilebilir. Örneğin, `%ce` durmuş olan `ce` işini ifade eder. `%?ce` şeklinde bir kullanımda ise komut satırında `ce` dizgesi geçen bir iş ifade edilebilir. Eğer önek ve dizge birden fazla işle eşleşirse, Bash bir hata uyarısı verir.

Bir işin isimlendirilmesi onu önalana almak için kullanılabilir: `%1`, **fg**`%1` komutuna eşdeğerdir ve 1 numaralı işi artalandan önalana almak içindir. Benzer olarak, `%1 &`, **bg**`%1` komutuna eşdeğerdir ve 1 numaralı işin artalanda sürdürülmesini sağlar.

Kabuk herhangi bir yerde bir iş durumunu değiştirdiğinde anında öğrenir. Normalde Bash bir işi, durumundaki değişiklikleri raporlamadan önce bir komut istemi basmaya hazır hale gelene kadar herhangi bir çıktısına kesme uygulamadan bekler. **set** yerleşiği `-b` seçeneği ile çalıştırılmışsa, Bash bu değişiklikleri anında raporlar. Her çocuk sürecin çıkışında varsa `SIGCHLD` sinyal kapanı çalıştırılır.

Durmuş işler varken Bash çıkmaya çalışırsa, kabuk durmuş işlerin varlığını belirten bir uyarı iletisi basar. Bundan sonra onların durumunu öğrenmek için **jobs** komutu kullanılabilir. Bir ara komut olmaksızın çıkmak için ikinci bir istek daha gelirse, kabuk bir ileti daha göstermez ve durmuş işler sonlandırılır.

2. İş Denetim Yerleşikleri

2.1. Bg Yerleşiği

bg [*iş–belirtimi*] ...

`&` ile başlatılmışçasına artalanda bekleyen *iş–belirtimi* işini sürdürür. *iş–belirtimi* verilmemişse, o anki iş kullanılır. İş denetimi etkin değilken çalıştırılmadıkça veya iş denetimi etkinleştirilerek çalıştırıldığında belirtilen *iş–belirtimi*nin bulunamaması dışında ya da *iş–belirtimi* iş denetimsiz başlatılan bir iş olmadıkça dönüş durumu sıfırdır.

2.2. Fg Yerleşiği

fg [*iş–belirtimi*]

Önalandaki *iş–belirtimi* işini sürdürür ve onu o anki iş yapar. Eğer *iş–belirtimi* verilmeksizin çalıştırılırsa o anki iş kullanılır. Normalde dönüş durumu önalana yerleştirilen komutun çıkış durumudur. İş denetimi etkin değilse veya iş denetimini etkinleştirerek çalıştırıldığında, *iş–belirtimi* geçerli bir işi belirtmiyorsa veya *iş–belirtimi* iş denetimsiz başlatılmış bir işi belirtiyorsa dönüş durumu sıfırdan farklıdır.

2.3. Jobs Yerleşiği

jobs [**-lnprs**] [*iş–belirtimi*]

jobs -x *komut* [*argümanlar*]

İlk satır etkin işleri listeler. Seçeneklerin anlamları:

-l
: Normal bilgilere ek olarak süreç kimliklerini de (PID) listeler.

-n
: Sadece kullanıcının durumları hakkında aldığı son uyarıdan beri durumları değişen işler hakkında bilgi gösterir.

-p
Sadece işlerin süreç grup liderlerinin süreç kimliklerini listeler.

-r
Sadece çalışan işler çıktılanır.

-s
Sadece durmuş işler çıktılanır.

iş–belirtimi verilmişse, çıktı bu iş ile ilgili bilgilerle sınırlıdır, verilmemişse tüm işlerin durumları listelenir.

`-x` seçeneği verilmişse, *argümanlar* ile belirtilen tüm iş belirtimlerini, işlerin süreç grup liderinin süreç grup kimliğine yerleştirilip *komut* çalıştırılır. Dönüş durumu komutun çıkış durumudur.

2.4. Kill Yerleşiği

```
kill [-s sinyal–belirtimi] [-n sinyal–numarası] [-sinyal–belirtimi] iş–belirtimi
kill [-s sinyal–belirtimi] [-n sinyal–numarası] [-sinyal–belirtimi] pid

kill -l [çıkış–durumu]
```

pid ile belirtilen süreç kimliğine veya *iş–belirtimi* ile belirtilen işin sürecine *sinyal–belirtimi* veya *sinyal–numarası* ile belirtilen sinyali gönderir. *sinyal–belirtimi* ya harf büyüklüğüne duyarsız olarak `SIGINT` gibi bir sinyal ismi (`SIG` öneki olmadan da verilebilir) ya da bir sinyal numarası olabilir. *sinyal–numarası* ise bir sinyal numarası olmalıdır. *sinyal–belirtimi* veya *sinyal–numarası* verilmezse öntanımlı olarak `SIGTERM` kullanılır. `-l` seçeneği ile sinyal isimleri listelenir. Bu seçenekle birlikte bir argüman verilmişse, bu argümanlara karşı düşen sinyal isimleri listelenir ve dönüş durumu sıfırdır. *çıkış–durumu* ile bir sinyal numarası ya da bir sinyal ile sonlandırılan bir sürecin çıkış durumu belirtilebilir. En azından bir sinyal başarıyla gönderilmişse çıkış durumu sıfırdır. Bir hata oluşur veya geçersiz bir seçenek saptanırsa sıfırdan farklıdır.

İş denetimi etkin değilken, **kill** yerleşiği *iş–belirtimi* argümanını kabul etmez. Bunlar süreç kimlikleri olarak verilmelidir.

2.5. Wait Yerleşiği

```
wait [ iş–belirtimi | pid … ]
```

İş belirtimi *iş–belirtimi* ile veya süreç kimliği *pid* ile belirtilen bir çocuk süreçlerin her birinin çıkmasını bekler. Dönüş durumu çıkmasını beklediği sürecin çıkış durumudur. Hiçbir argüman verilmeden çalıştırılırsa etkin tüm çocuk süreçler için bekler, bu durumda çıkış durumu sıfırdır. Ne *iş–belirtimi* ne de *pid* kabuğun bir çocuk sürecini belirtiyorsa 127 durumu ile döner.

İş denetimi etkin değilken, **wait** yerleşiği *iş–belirtimi* argümanını kabul etmez. Bunlar süreç kimlikleri olarak verilmelidir.

2.6. Disown Yerleşiği

```
disown [-ar] [-h] [iş–belirtimi ...]
```

Seçeneksiz kullanımda her *iş–belirtimi* etkin işler tablosundan kaldırılır. `-h` seçeneği ile iş tablodan kaldırılmaz ama imlenir böylece kabuk bir `SIGHUP` alırsa bunu işe göndermez. *iş–belirtimi* ve `-a` ile `-r` seçenekleri verilmezse, o anki iş kullanılır. *iş–belirtimi* verilmeden `-a` seçeneğinin kullanılması tüm işlerin imlenmesi ya da kaldırılması demektir. *iş–belirtimi* verilmeden `-r` seçeneğinin kullanılması ise işlemi işlerin çalıştırılması ile sınırlar.

2.7. Suspend Yerleşiği

```
suspend [-f]
```

Bu kabuğun çalışmasını bir SIGCONT sinyali alana kadar askıya alır. -f seçeneği kullanıldığında kabuk bir giriş kabuğu olsa bile askıya alınır.

3. İş Denetim Değişkenleri

auto_resume

Bu değişken kabuğun kullanıcı ve iş denetimi ile nasıl etkileşeceğini kontrol eder. Bu değişken varsa, yönlendirmesiz tek sözcüklük basit komutlar bir mevcut işin yeniden başlatılması için aday kabul edilir. Bir belirsizliğe izin verilmez; girilen dizge ile başlayan birden fazla iş varsa en son erişilen iş seçilmiş olacaktır. Bu bağlamda, durmuş işin ismi onu başlatmakta kullanılan komut satırıdır. Bu değişkene exact değeri atanırsa, verilen dizge tıpatıp bir durmuş işin ismi olmalıdır. Değişkene substring değeri atanırsa, verilen dizge bir durmuş işin isminin bir altdizgesi olması gerekir. substring, %? iş kimliği ile benzer işlevsellik sağlar (*İş Denetiminin Temelleri* (sayfa: 83) bölümüne bakınız). Bunların dışında bir dizge atanırsa, verilen dizge bir durmuş işin isminin bir öneki olmalıdır. Bu, % iş kimliği ile benzer işlevsellik sağlar.

VIII. Komut Satırının Düzenlenmesi

İçindekiler

Bu oylumda GNU komut satırı düzenleme arayüzünün temel özelliklerinden söz edilecektir. Komut satırı düzenlemesi, Bash'in de aralarında olduğu pek çok uygulamanın kullandığı, Readline kütüphanesi tarafından sağlanır.

1. Satır Düzenlemeye Giriş

Tuşlarla ilgili olarak kullanılan nitelemeler aşağıda açıklanmıştır.

`C-k` gördüğünüz yerde bu dizgeyi 'Kontrol–K' olarak okuyacaksınız ve `Ctrl` tuşu basılıyken `k` tuşuna da basmanız gerektiğini anlayacaksınız.

`M-k` gördüğünüz yerde ise bu dizgeyi 'Meta–K' olarak okuyacaksınız ve boşluk tuşunun solunda bulunan `Alt` tuşu basılı iken `k` tuşuna da basmanız gerektiğini anlayacaksınız.

Boşluk tuşunun solundaki `Alt` tuşu 'Meta' tuşu olarak da bilinir. Boşluk tuşunun sağındaki `AltGr` tuşu ise bazı klavyelerde Meta tuşu olabildiği gibi Türk klavyelerinde olduğu gibi tuşlara atanan ek karakterler için de kullanılır. Bu bakımdan Meta tuşu denilince soldaki `Alt` tuşundan bahsedildiğini anlayacağız.

Bazı klavyelerde bu iki tuş bulunmayabilir ya da onlara farklı görevler atanmış olabilir. Bu gibi durumlar için başvurulabilecek bir tuş daha vardır: `Esc` tuşu. Yalnız `Esc` tuşu `Alt` tuşu gibi kullanılmaz. `M-k` için `Esc`

tuşuna bir kere basıp bıraktıktan sonra k tuşuna basmalısınız. Her iki işleme k tuşunun ötelenmesi de denir (metafying).

M-C-k gördüğünüz yerde bu dizgeyi 'Meta–Kontrol–k' olarak okuyacaksınız ve Alt ile Ctrl tuşlarına basılı iken k tuşuna da basmanız gerektiğini anlayacaksınız. Bu işleme C-k ötelemesi de denir.

Bunlara ek olarak bazı tuşların kendi isimleri kullanılmıştır. Özellikle DEL (silme tuşu), ESC (öteleme tuşu), LFD, SPC (boşluk tuşu), RET (return veya enter tuşu) ve TAB (sekme tuşu) tuşları ilerleyen bölümlerde ve init dosyasında (*Readline İlklendirme Dosyası* (sayfa: 90) bölümüne bakınız) bu isimlerle geçer. Klavyenizde LFD tuşu yoksa C-j tuşlayarak bu tuşun işlevselliğine erişebilirsiniz.

2. Readline Etkileşimi

Bir etkileşimli oturumda satıra uzunca bir metin girdiğinizde çoğunlukla sadece ilk sözcüğü yanlış yazdığınızda uyarılırsınız. Readline kütüphanesi bu gibi durumlarda hatalı yazdığınız metni yeniden yazmadan kolayca değiştirebilmenizi sağlayan bazı komutlara sahiptir. Bu düzenleme komutlarını kullanrak imleci düzeltme yapılacak yere götürebilir, hatalı kısmı silebilir ve silinen yere doğrusunu yazabilirsiniz. Düzeltmeyi bitirdiğinizde metnin neresinde olursanız olun basitçe RET tuşuna bastığınızda satırın tümü kabul edilir.

2.1. Readline'ın Yalın Özü

Satıra karakterleri girmek isterseniz basitçe onları yazarsınız. Yazılan karakterler imlecin olduğu yerde görünür ve siz yazdıkça imleç hep sağdaki boşluğa kayar. Bir karakteri yanlış yazarsanız, silme tuşlarını kullanarak onu yerinde ya da geriye doğru silebilirsiniz.

Bazan yanlış yazılmış bir karakteri geç farkedersiniz ya da başka karakterleri yazdıktan sonra bir hata uyarısı alırsınız. Bu durumda C-b tuşlayarak satır üzerinde sola ilerleyip hatalı yeri düzelttikten sonra C-f tuşlayarak kaldığınız yere geri dönebilirsiniz.

Bir satırın ortasına bir kaç sözcük eklemek isterseniz, imleç sağa ilerledikçe imlecin sağındaki karakterlerde sağa doğru ötelenerek gireceğiniz metin için yer açılır. Tersine olarak, arada kalan bir metni sildiğinizde, açılan boşluk sağdaki karakterlerin sola ötelenmesiyle doldurulur.

Girdi satırındaki metni düzenlemek için kullanılan temel tuşlar:

C-b
: İmleç bir karakter sola gider.

C-f
: İmleç bir karakter sağa gider.

DEL veya Gerisilme
: İmlecin solundaki karakteri siler.

C-d
: İmlecin altındaki karakteri siler.

basılan karakterler
: İmlecin olduğu yerde satıra eklenir.

C-_ veya C-x C-u
: Son yapılan düzeltmeleri geri alır. Geri ala ala boş satıra kadar gidilebilir.

Klavye yapılandırmanıza bağlı olarak, imlecin altındaki karakteri silen C-d yerine DEL tuşu kullanılabilir (Türk klavye yapılandırması böyledir).

2.2. Readline Hareket Tuşları

Bir önceki bölümde bahsedilen tuşlar, girdi satırı üzerinde yapacağınız düzenlemelerde kullanabileceğiniz en temel tuşlardı. Rahatınız için C-b, C-f, C-d ve DEL tuşlarına ek olarak başka komutlar da bulunur.

C-a
: İmleç satırın başına gider.

C-e
: İmleç satırın sonuna gider.

M-f
: İmleç bir sözcük ileri gider. Harf ya da rakam olmayan karakterler sözcükten sayılmaz, onlara boşluk gibi davranılır.

M-b
: İmleç bir sözcük ileri gider. Harf ya da rakam olmayan karakterler sözcükten sayılmaz, onlara boşluk gibi davranılır.

C-l
: Son satır tepede kalacak şekilde ekran yukarı kaydırılır (ekran temizlenir).

Dikkat ettiyseniz C-f imleci bir karakter sağa kaydırırken, M-f imleci bir sözcük sağa kaydırmaktadır. Bir teamül olarak Ctrl ile basılan tuşlar karakterlerle Alt ile basılan tuşlar sözcüklerle ilgilidir.

2.3. Readline Kes ve Yapıştır Komutları

Kesmek satırdaki metni silmek ama silerken daha sonra kullanmak üzere saklamaktır. *Yapıştırmak* ise kesilen metni satıra yerleştirmektir.

Bir komut açıklamasında 'kesmek'ten bahsediliyorsa, bu metnin satırda aynı yerde ya da farklı bir yerde tekrar kulanabileceğiniz anlamına gelir.

Bir metni kestiğinizde metin *pano*'ya kopyalanır. Metin içinde ardışık yapılan kesmeler, satır üzerinde kesildiği sırada birlikte tutulur. Örneğin "aaa bbb ccc ddd" satırında sırayla "aaa", "bbb", "ddd" kesilmişse metni panodan istediğinizde "aaa bbb" ile "ddd" şeklinde iki kesme (üç değil) bulunduğunu görürüz. Ancak pano satıra özel değildir. Bir satırdan kesilen bir metin başka bir satıra yapıştırılabilir.

Metinleri kesmek için kullanılan komutlar:

C-k
: Metni imleç konumundan satırın sonuna kadar keser.

M-d
: Metni imleç konumundan sözcüğün sonuna kadar keser. İmleç iki sözcüğün arasındaysa, metin sonraki sözcüğün sonuna kadar kesilir. Sözcük sınırları M-f ile aynıdır.

M-DEL
: Metni sözcük başlangıcından imlece kadar keser. İmleç iki sözcüğün arasındaysa, metin önceki sözcüğün başına kadar kesilir. Sözcük sınırları M-f ile aynıdır.

C-w

Metni imleç konumundan önceki boşluğa kadar keser. Buradaki sözcük sınırları M-DEL'deki sınırlardan farklıdır.

Burada metnin satıra nasıl yapıştırılacağından bahsedilecektir. Yapıştırmak, panodaki son metni satıra kopyalamaktır.

C-y
: Son kesilen metni imleç konumunda satıra yerleştirir.

M-y
: Her M-y pano içeriğindeki kesmelerden bir öncekinin seçilmesini sağlar. Pano içeriği M-y ile sürekli döndürülür. Bu işlem sadece önceki komut C-y ya da M-y ise uygulanır.

2.4. Readline Argümanları

Readline komutlarına sayısal argümanlar aktarılabilir. Bu argümanlar bazan bir yineleme, bazan bir argümanın önem derecesini gösteren bir *işaret* olarak ele alınır. Negatif bir argüman, normalde ileri yönde bir eylem oluşturan bir komutta ters yönde bir eyleme sebep olacaktır. Örneğin bir metni imleç konumundan satırın başına kadar kesmek isterseniz M-- C-k komutunu verebilirsiniz.

Bir komuta sayısal argümanları aktarmanın genel yolu komuttan önce rakam ötelemesi yapmaktır. Bir sayıdan önce bir eksi işareti (-) koyarsanız, argümanın işareti negatif olur. Bir rakam ötelemesinden sonra sayıyı tamamlamak için kalan rakamlar tek başına tuşlanır ve ardından komut tuşlanır. Örneğin C-d komutuna 10 argümanını vermek için M-1 0 C-d tuşlanır ve girdi satırında imleç konumundan itibaren 10 karakter silinir.

2.5. Geçmiş içinde Komutların Aranması

Readline komut geçmişinde belirtilen bir dizgeyi aramak için komutlara sahiptir (*Bash'in Geçmişsel Yetenekleri* (sayfa: 111) bölümüne bakınız). İki arama kipi vardır: *arttırımlı* ve *arttırımsız*.

Arttırımlı arama, kullanıcı aranacak dizgeyi yazmaya başladığı anda başlar. Karakterler yazıldıkça, Readline yazılan dizge ile eşleşen bir sonraki geçmiş girdisini gösterir. Bir arttırımlı arama, istenen geçmiş girdisini bulmayı sağlayacak kadar karakterin yazılmasını gerektirir. Bir dizgeyi geçmişte geriye doğru aramayı başlatmak için C-r tuşlayın. C-s aramayı ileri yönde başlatır. Arttırımlı aramayı sonlandıracak karakterler isearch-terminators değişkeninde tutulur. Değişkene bir değer atanmamışsa, ESC ve C-J bir arttrımlı aramayı sonlandıracaktır. C-g aramayı iptal ederek özgün satıra dönülmesini sağlar. Arama sonlandırıldığında aramanın sonucu olan girdi komut satırı haline gelir.

Geçmiş listesinde diğer eşleşmeleri bulmak için aramayı hangisiyle başlattığınıza bağlı olarak C-r ya da C-s tuşlayabilirsiniz. Bu, yazdığınız arama dizgesine bağlı olarak ileri veya geri yönde eşleşen diğer geçmiş girdilerini taramanızı sağlar. Readline komutu olan herhangi bir tuşlama aramayı sonlandırır ve komutu çalıştırır. Örneğin, RET tuşu aramayı sonlandırıp satırın kabul edilmesini ve geçmiş listesindeki komutun çalıştırılmasını sağlar. Bir hareket komutu aramayı sonlandıracak, bulunan son satırı komut satırı yapacak ve satır düzenlemesini başlatacaktır.

Readline son arttırımlı arama dizgesini hatırlar. Peşpeşe iki C-r girerseniz, hatırdaki arama dizgelerinden biri kullanılır.

Arttırımsız arama, eşleşen geçmiş satırını aramaya başlamadan önce arama dizgesinin tamamını okur. Arama dizgesi kullanıcı tarafından yazılmış ya da o anki satırın içindeki bir parça olabilir.

3. Readline İlklendirme Dosyası

Readline kütüphanesi kurulumda öntanımlı olarak Emacs benzeri tuş kısayolları ile gelmekle birlikte, farklı tuş kısayolları ile de kullanmak mümkündür. Bir kullanıcı, teamül olarak ev dizininde bulunan bir `inputrc` dosyasına yerleştirdiği komutlarla Readline'ın kullanıldığı uygulamaları kişiselleştirebilir. Bu dosyanın ismi bir kabuk değişkeni olan `INPUTRC` değişkeninden alınır. Bu değişken atanmamışsa dosyanın öntanımlı ismi `~/.inputrc`'dir. Bu dosya mevcut değilse vela okunamıyorsa, eninde sonunda bakılacak dosya `/etc/inputrc` dosyasıdır.

Readline kütüphanesini kullanan bir uygulama başlatıldığında ilklendirme dosyası okunur ve tuş kısayolları atanır.

Ek olarak, `C-x C-r` komutu bu ilklendirme dosyasının tekrar okunmasını sağlar, böylece dosyanın okunmasından sonra dosyada değişiklik yapıldığında bu değişikliklerin etkinleştirilmesi sağlanmıştır.

3.1. Readline İlklendirme Dosyasının Sözdizimi

Readline ilklendirme dosyasında izin verilen bir kaç temel yapı vardır. Boş satırlar yoksayılır. `#` ile başlayan satırlarda açıklamalar verilebilir. `$` ile başlayan satırlar koşullu yapılardır (*Koşullu İlklendirme Yapıları* (sayfa: 95) bölümüne bakınız). Diğer satırlar değişken atamaları ve tuş kısayolları olarak ele alınır.

3.1.1. Değişken Atamaları

Readline'ın çalışma anı davranışlarını, Readline değişkenlerine ilklendirme dosyasında **`set`** komutuyla değer atayarak değiştirebilirsiniz. Sözdizimi basittir:

`set` *değişken değer*

Burada, örneğin öntanımlı Emacs benzeri tuş kısayollarını **`vi`** satır düzenleme komutları ile değiştirmek için komutu şöyle yazabilirsiniz:

```
set editing-mode vi
```

Değişken isimleri ve değerleri uygun kullanıldığında harf büyüklüklerine bakılmaksızın tanınır. Bilinmeyen değişken isimleri yoksayılır.

Mantıksal değişkenlerden (açık veya kapalı olabilenler) değerleri boş, null, `on` (harf büyüklüğü önemsiz) veya 1 olanlar açık (`on`) kabul edilir; bunların dışındaki herhangi bir değere sahip olanlar kapalı (`off`) kabul edilir.

`bind` `-V` (sayfa: 43) komutu o anki Readline değişken isimlerini ve değerlerini listeler.

Çalışma anı davranışlarının büyük kısmı aşağıdaki değişkenlerle değiştirilebilir:

`bell-style`
: Readline uçbirim çanını çaldırmak istediğinde neler olacağını kontrol eder. Değer olarak `none` atanırsa, çan hiç çalmaz. `visible` atanırsa ve bir görünür çan kullanılabilir durumdaysa Readline kullanır. `audible` atanırsa (öntanımlıdır), Readline uçbirim çanını çalmaya çalışır.

`bind-tty-special-chars`
: Değeri **`on`** ise Readline, çekirdeğin uçbirim sürücüsü tarafından özel olarak ele alınan kontrol karakterlerini Readline eşdeğerlerine bağlamaya çalışır.

`comment-begin`
: **`insert-comment`** komutu çalıştırıldığında satırın başına yerleştirilecek dizge. Öntanımlı değeri: "`#`".

`completion-ignore-case`
: `on` değeri atanırsa, Readline harf büyüklüğüne duyarlı dosyaismi eşleştirmesi ve tamamlaması uygular. Öntanımlı değeri: `off`.

`completion-query-items`
: Kullanıcıya olasılıkların listesini görmek isteyip istemediği sorulurken saptanan olası tamamlamaların sayısı. Olası tamamlamaların sayısı bu değerden yüksekse Readline kullanıcıya onları görmek isteyip istemediğini soracaktır; aksi takdirde, basitçe listelenecektir. Bu değişkene sıfırdan büyük ya da eşit bir tamsayı atanabilir. Negatif bir değer asla sorulmayacak anlamına gelir. Öntanımlı değeri: 100.

`convert-meta`
: `on` değeri atanırsa, Readline karakterlerin sekizinci bitini ayırıp sekizinci bite bir ASCII tuş dizisi atayarak ve onları meta önekli tuş dizisine çeviren `ESC` karakteri ile önceleyerek karakterleri dönüştürecektir. Öntanımlı değeri: `on`.

`disable-completion`
: `On` değeri atandığında, Readline sözcük tamamlamasını iptal eder. Tamamlama karakterleri nasıl girildiyse o şekilde satıra yerleştirilir. Öntanımlı değeri: `off`.

`editing-mode`
: `editing-mode` değişkeni kullanılan öntanımlı tuş kısayollarının kipini kontrol eder. Öntanımlı olarak Readline tuş kısayolları Emacs'ınkine çok benzeyen Emacs düzenleme kipinde başlar. Bu değişkene ya `emacs` ya da `vi` atanabilir.

`enable-keypad`
: `on` değeri atandığında, Readline uygulama çağrıldığında sayısal tuş takımını etkinleştirmeye çalışacaktır. Bazı sistemler ok tuşlarının kullanılabilmesi için sayısal tuş takımının etkinleştirilmesine ihtiyaç duyar. Öntanımlı değeri: `off`.

`expand-tilde`
: `on` değeri atanırsa, Readline sözcük tamamlaması yapmaya çalışırken yaklaşık yorumlaması uygular. Öntanımlı değeri: `off`.

`history-preserve-point`
: `on` değeri atanırsa, komut geçmişi kodu imleci `previous-history` veya `next-history` ile alınan her geçmiş satırında aynı konuma yerleştirmeye çalışır. Öntanımlı değeri: `off`.

`horizontal-scroll-mode`
: `on` değeri atanırsa, düzenlenen metnin satırlarının uzunluğu ekran genişliğinden fazlaysa imleci bir alt satıra indirmek yerine, satır kaydırılır. Öntanımlı değeri: `off`.

`input-meta`
: `on` değeri atanırsa, uçbirimin destek için ne gerektirdiğine bakılmaksızın, Readline sekiz bitlik girdileri etkinleştirir (okunurken sekizinci bit temizlenmeyecektir). `meta-flag` bu değişkenle eşanlamlıdır. Öntanımlı değeri: `off`.

`isearch-terminators`
: Bir arttırımlı aramayı sonlandıracak karakter dizisi (*Geçmiş içinde Komutların Aranması* (sayfa: 90) bölümüne bakınız). Bu değiken bir değerle atanmamışsa bir arttırımlı arama `ESC` ve `C-J` ile sonlandırılır.

`keymap`
: Tuş kısayolları için kullanılacak tuşeşlemi atanır. Kabul edilen tuşeşlemi isimleri `emacs`, `emacs-standard`, `emacs-meta`, `emacs-ctlx`, `vi`, `vi-move`, `vi-command` ve `vi-insert`. `vi` ile `vi-command` ve `emacs` ile `emacs-standard` eşdeğerdir. Öntanımlı değeri `emacs`'dır. Ayrıca `editing-mode` değişkeni de öntanımlı tuşeşlemini etkiler.

`mark-directories`
: `on` değeri atanırsa„ tamamlanmış dizin isimlerine bir `/` eklenir. Öntanımlı değeri: `on`.

`mark-modified-lines`
: `on` değeri atanırsa, değiştirilmiş geçmiş satırlarını Readline başına bir `*` ekleyerek gösterir. Öntanımlı değeri: `off`.

`mark-symlinked-directories`
: **`on`** yapılırsa, dizinlere sembolik bağ olan tamamlanmış isimlere bir / eklenmiş olur (`mark-directories` değerine konu). Öntanımlı değeri: `off`.

`match-hidden-files`
: `on` değeri atanırsa, dosyaismi tamamlaması uygularken, dosyaisminin tamamlanmasında baştaki **`.`** kullanıcı tarafından verilmedikçe, Readline'ın **`.`** ile başlayan dosya isimlerini eşleştirmesini sağlar. Öntanımlı değeri: `on`.

`output-meta`
: `on` değeri atanırsa Readline, karakterleri bir meta önekli öteleme dizileri olarak değil sekizinci biti etkin olarak gösterecektir. Öntanımlı değeri: `off`.

`page-completions`
: **`on`** yapılırsa, Readline olası tamamlamaları bir defada bir tam ekran göstermek için **`more`** benzeri bir dahili sayfalayıcı kullanır. Öntanımlı değeri: `on`.

`print-completions-horizontally`
: `on` değeri atanırsa, Readline tamamlamalarla eşleşmeleri ekranda yukardan aşağı dizmek yerine alfabetik sırada yanyana gösterecektir. Öntanımlı değeri: `off`.

`show-all-if-ambiguous`
: Tamamlama işlevlerinin öntanımlı davranışını değiştirir. `on` değeri atanırsa, birden fazla olası tamamlaması olan sözcüklerin eşleşmelerinin çanı çaldırmak yerine listelenmesini sağlar. Öntanımlı değeri: `off`.

`show-all-if-unmodified`
: Tamamlama işlevlerinin öntanımlı davranışlarını *show–all–if–ambiguous*'a benzer bir şekilde değiştirir. Değeri **`on`** yapılırsa herhangi bir kısmi tamamlama olmaksızın (olası tamamlamalar ortak bir önek paylaşmazlar) tek olası tamamlamadan fazlasına sahip sözcükler çanı çaldırmak yerine eşleşenlerin hemen listelenmesine sebep olur. Öntanımlı değeri: `off`.

`visible-stats`
: `on` değeri atanırsa, dosya türünü gösteren bir karakter, olası tamamlamalar listelenirken dosya ismine eklenir. Öntanımlı değeri: `off`.

3.1.2. Tuş Kısayolları

İlklendirme dosyasında tuş kısayollarının kontrol edilmesi için kullanılan sözdizimi basittir. Önce değiştirmek istediğiniz komutun ismini bulmanız gerekir. Aşağıda komut isimlerini, öntanımlı tuş kısayollarını ve varsa komutun ne yaptığını anlatan bir kısa açıklamanın da bulunduğu bir liste bulacaksınız.

Komutun ismini öğrendikten sonra, ilklendirme dosyasında bir satıra, önce komuta kısayol olarak atayacağınız tuşun ismini yazın, bir **`:`** işareti koyun ve komutun ismini yazın. Tuşun ismi ile ikinokta imi arasında boşluk olmayabilir, bu durumda ikinokta imi tuş isminin bir parçası olarak yorumlanır. Tuşun ismini kolayınıza gelen farklı yollarla ifade edebilirsiniz.

Komut isimlerine ek olarak, Readline tuşlara basıldığında bir dizge basmalarını sağlamak üzere bir dizge atanmasına da izin verir (bir *makro*).

bind `-p` *komutu* (sayfa: 43) Readline işlev isimleri ve kısayollarını bir ilklendirme dosyasına doğrudan konulabilecek biçemde gösterir.

tuşismi: işlev–ismi veya makro
: *tuşismi*, bir tuşun İngilizce ismidir. Örnek:

```
Control-u: universal-argument
Meta-Rubout: backward-kill-word
Control-o: "> çıktı"
```

Bu örnekte, `C-u` tuşlarına `universal-argument` işlevi, `M-DEL` tuşlarına `backward-kill-word` işlevi ve `C-o` tuşlarına çalıştırılacak bir makro (satıra `> çıktı` dizgesini girer) atanmıştır.

Tuş kısayollarını tanımlarken kullanılabilecek sembolik karakter isimlerinin sözdizimleri: *DEL*, *ESC*, *ESCAPE*, *LFD*, *NEWLINE*, *RET*, *RETURN*, *RUBOUT*, *SPACE*, *SPC* ve *TAB*.

"*tuşdizisi*": *işlev–ismi veya makro*
: *tuşdizisi tuşismi*'nden dizgenin çift tırnak içine alınmış olması ile farklıdır. Bazı GNU Emacs tarzı tuş ötelemeleri aşağıdaki örnekteki gibi kullanılabilmesine karşın özel karakter isimleri kullanılamaz.

```
"\C-u": universal-argument
"\C-x\C-r": re-read-init-file
"\e[11~": "İşlev tuşu 1"
```

Bu örnekte, `C-u` tuşlarına `universal-argument` işlevini (ilk örnekteki gibi), `C-x C-r` tuşlarına `re-read-init-file` işlevini ve `ESC [ 1 1 ~` ise `İşlev tuşu 1` metnini girmek üzere atanmıştır.

Aşağıdaki GNU Emacs tarzı tuş ötelemeleri, tuş dizilerini belirtirken kullanılabilir:

`\C-`
: Kontrol öneki

`\M-`
: Meta öneki

`\e`
: öteleme karakteri

`\\`
: tersbölü

`\"`
: ", bir çift tırnak işareti

GNU Emacs tarzı tuş ötelemelerine ek olarak, tersbölü öncelemeli karakterler de mevcuttur:

`\a`
: sesli uyarı (bell)

`\b`
: gerisilme

`\d`
: silme

`\f`

sayfa ilerletme

`\n`
satırsonu

`\r`
satırbaşı

`\t`
yatay sekme

`\v`
düşey sekme

`\nnn`
sekizlik değeri `nnn` olan sekiz bitlik karakter (1, 2 ya da 3 haneli olabilir).

`\xHH`
onaltılık değeri `HH` olan sekiz bitlik karakter (1 ya da 2 haneli olabilir).

Bir makro metnini girerken, bir makro tanımını belirtmek için çift ya da tek tırnaklar arasına alınmalıdır. Tırnaksız metinler bir işlev ismi olarak kabul edilir. Makro gövdesinde tersbölü öncelemeli karakterler yukarıda açıklandığı gibi yorumlanır. Makro metni içindeki tırnak içine alınmış yerlerde tırnak işaretleri `"` ve `'` tersbölü ile öncelenmelidir. Örneğin aşağıdaki atama `C-x \` tuşlarıyla bir tek `\` basar:

```
"\C-x\\": "\\"
```

3.2. Koşullu İlklendirme Yapıları

Readline, C önişlemcisinin koşullu derleme özelliklerine benzer testlerin sonucu olarak uygulanacak tuş kısayolları ve değişken atamalarına izin veren bir araç sağlar. Kullanılan dört çözümleyici yönerge vardır.

`$if`
`$if` yapısı, Readline kullanan uygulamalar, kullanılan uçbirim ve düzenleme kipi üzerine tabanlanmış kısayollar yapılmasına izin verir. Test metni satır sonuna kadar uzar; onu izole etmek için bir karaker gerekmez.

`mode`
`$if` yönergesinin `mode=` sınaması Readline'ın `emacs` kipinde mi yoksa `vi` kipinde mi olduğunu test etmekte kullanılır. Bu, **`set`** `tuşeşlemi` komutu ile birlikte kullanılabilir, örneğin, Readline sadece `emacs` kipinde başlatıldığında kısayolların `emacs-standard` ve `emacs-ctlx` tuşeşlemlerine ayarlanması gibi.

`term`
`term=` sınaması uçbirime özel tuş kısayollarını içermekte, belki de tuş dizilerinin çıktısını uçbirimin işlev tuşlarına göre bağlamakta kullanılabilir. `=` işaretinin sağındaki sözcük hem uçbirimin tam adı hem de ilk `-`' den önceki parçasında test edilir. Bu, örneğin `sun` sözcüğünün hem `sun` hem de `sun-cmd` ile eşleşmesine izin verir.

`uygulama`
`uygulama` sınaması uygulamaya özel ayarların içerilmesinde kullanılır. Readline kütüphanesini kullanan her program *uygulama ismi*'ni belirler ve bir kısmi değer için test yapabilirsiniz. Bu, tuş dizilerini belli bir program için kullanışlı olan işlevlere bağlamakta kullanılmalıdır. Örneğin, aşağıdaki komut Bash'da o anki ya da önceki sözcüğü tırnak içine alan bir tuş dizisi ekler:

```
$if Bash
# Şimdiki ya da önceki sözcüğü tırnak içine al
"\C-xq": "\eb\"\ef\""          $endif
```

`$endif`
Bu yönerge yukardaki örnekte de görüldüğü gibi bir **`$if`** yönergesini sonlandırmak için kullanılır.

`$else`
`$if` yönergesi başarısız olduğunda bu yönerge altındaki komutlar çalıştırılır.

`$include`
Bu yönerge argüman olarak tek bir dosyaismi alır ve komutları ve tuş kısayollarını dosyadan okur. Örneğin aşağıdaki yönerge `/etc/inputrc` dosyasını okur:

```
$include /etc/inputrc
```

3.3. Örnek İlklendirme Dosyası

Buradaki `inputrc` dosyası örneği tuş kısayolları, değişken atamaları ve koşullu sözdizimlerini örneklerle açıklamaktadır.

```
# Bu dosya Gnu Readline Kütüphanesini kullanan uygulamalar için
# satır girdi düzenlemesi davranışlarını kontrol eder.
# Bu uygulamalar FTP, Bash ve Gdb'dir.
#
# inputrc dosyasını C-x C-r ile yeniden okutabilirsiniz.
# '#' ile başlayan satırlar açıklamadır.
#
# Önce, sistem genelinde etkin olacak kısayollar ve değişken atamaları
# /etc/inputrc dosyası ile dosya içeriğine dahil edilir.
$include /etc/inputrc

#
# Çeşitli kısayollar emacs kipinde atanır
set editing-mode emacs

$if mode=emacs

Meta-Control-h: backward-kill-word    İşlev isminden sonraki metin yoksayılır

#
# Sayısal tuştakımı kipinde ok tuşları
#
#"\M-OD":        backward-char
#"\M-OC":        forward-char
#"\M-OA":        previous-history
#"\M-OB":        next-history
#
# ANSI kipinde ok tuşları
#
"\M-[D":        backward-char
"\M-[C":        forward-char
"\M-[A":        previous-history
"\M-[B":        next-history
#
# 8 bit sayısal tuştakımı kipinde ok tuşları
```

```
#
#"\M-\C-OD":       backward-char
#"\M-\C-OC":       forward-char
#"\M-\C-OA":       previous-history
#"\M-\C-OB":       next-history
#
# 8 bit ANSI kipinde ok tuşları
#
#"\M-\C-[D":       backward-char
#"\M-\C-[C":       forward-char
#"\M-\C-[A":       previous-history
#"\M-\C-[B":       next-history
C-q: quoted-insert

$endif
# Eski tarz bir kısayol.  Bunun öntanımlı olması umulur.
TAB: complete

# Kabuk etkileşimini kolaylaştıran makrolar
$if Bash
# yolu düzenle
"\C-xp": "PATH=${PATH}\e\C-e\C-a\ef\C-f"
# tırnak içine almak için hazırla --
# ardarda iki çift tırnak yazar
# ve bir karakter geri gider
"\C-x\"": "\"\"\C-b"
# bir tersbölü girer
# (tuş dizilerindeki tersbölü öncelemelerini ve makroları test etmek için)
"\C-x\\": "\\"
# Şimdiki ya da önceki sözcüğü tırnak içine alır
"\C-xq": "\eb\"\ef\""
# Satırı tazeler
"\C-xr": redraw-current-line
# Satırdaki değişkeni düzenler
"\M-\C-v": "\C-a\C-k$\C-y\M-\C-e\C-a\C-y="
$endif

# varsa bir görünür çan kullanılır
set bell-style visible

# okurken karakterlerden 8. biti ayırmaz
set input-meta on

# iso-* karakterlerin girilmesine izin verir
set convert-meta off

# karakterleri meta önekli karakterler olarak değil
# sekizinci bitleri doğrudan 1 yapılmış olarak gösterir
set output-meta on

# bir sözcüğün tamamlanabilmesi için 150 den fazla olasılık varsa
# kullanıcıya onları görmek isteyip istemediğini sor
set completion-query-items 150

# FTP için
$if Ftp
"\C-xg": "get \M-?"
```

```
"\C-xt": "put \M-?"
"\M-.": yank-last-arg
$endif
```

4. Kısayollar için Readline Komutları

Bu kısımda, tuş dizilerine bağlanabilecek Readline komutları anlatılmıştır. Tuş kısayollarını **bind** (sayfa: 43)`-P` komutunu çalıştırarak ya da bir `inputrc` dosyasında kullanılabilir biçemde **bind**`-p` komutu ile listeleyebilirsiniz. Bir tuş dizisinin eşlik etmediği komut isimleri öntanımlı olarak bağlantısızdır.

Aşağıdaki açıklamalarda `nokta` o anki imleç konumunu, `mark` ise **set-mark** komutu ile kaydedilmiş imleç konumunu ifade eder. `nokta` ile `mark` arasındaki dizge ise `bölge` ile ifade edilmiştir.

4.1. Hareket Komutları

beginning-of-line (`C-a`)
İmleç satırın başına gider.

end-of-line (`C-e`)
İmleç satırın sonuna gider.

forward-char (`C-f`)
İmleç bir karakter ileri gider.

backward-char (`C-b`)
İmleç bir karakter geri gider.

forward-word (`M-f`)
İmleç bir sözcük ileri gider. Harf ya da rakam olmayan karakterler sözcükten sayılmaz, onlara boşluk gibi davranılır.

backward-word (`M-b`)
İmleç bir sözcük geri gider. Harf ya da rakam olmayan karakterler sözcükten sayılmaz, onlara boşluk gibi davranılır.

clear-screen (`C-l`)
Son satır tepede kalacak şekilde ekran yukarı kaydırılır (ekran temizlenir).

redraw-current-line ()
Satırı tazeler. Öntanımlı olarak kısayol atanmamıştır.

4.2. Geçmişi Yöneten Komutlar

accept-line (`Newline veya Return`)
İmlecin yerine bakmaksızın satırı kabul eder. Bu bir boş satır değilse, `HISTCONTROL` ve `HISTIGNORE` değişkenlerinin değerlerine bağlı olarak geçmiş listesine eklenir. Satır değiştirilmiş bir geçmiş satırı ise, geçmiş satırı orjinal durumuna getirilir.

previous-history (`C-p`)
Önceki komut alınarak, geçmiş listesinde ‘geri’ gidilir.

next-history (`C-n`)

Sonraki komut alınarak, geçmiş listesinde 'ileri' gidilir.

beginning-of-history (M-<)
Geçmişteki ilk satıra gidilir.

end-of-history (M->)
Geçmişin son satırına gidilir, örneğin, o an girilen satır.

reverse-search-history (C-r)
O anki satırdan geriye doğru ve gerekliyse geçmişte 'yukarı' hareketle arama yapılır. Bu bir arttırımlı aramadır.

forward-search-history (C-s)
O anki satırdan ileriye doğru ve gerekliyse geçmişte 'aşağı' hareketle arama yapılır. Bu bir arttırımlı aramadır.

non-incremental-reverse-search-history (M-p)
O anki satırdan geriye doğru ve gerekliyse kullanıcı tarafından verilen bir dizge için bir arttırımsız arama kullanılarak geçmişte 'yukarı' hareketle arama yapılır.

non-incremental-forward-search-history (M-n)
O anki satırdan ileriye doğru ve gerekliyse kullanıcı tarafından verilen bir dizge için bir arttırımsız arama kullanılarak geçmişte 'aşağı' hareketle arama yapılır.

history-search-forward ()
O anki satırın başından imleç konumuna kadar olan metin için geçmişte ileri doğru arama yapılır. Bu bir arttırımsız aramadır. Öntanımlı olarak kısayol atanmamıştır.

history-search-backward ()
O anki satırın başından imleç konumuna kadar olan metin için geçmişte geriye doğru arama yapılır. Bu bir arttırımsız aramadır. Öntanımlı olarak kısayol atanmamıştır.

yank-nth-arg (M-C-y)
İlk argümanı önceki komuta imleç konumunda yerleştirir (genellikle önceki satırın ikinci sözcüğü). Bir n argümanı ile, önceki komutun n. sözcüğü olarak yerleştirilir (önceki satırdaki sözcükler ilk sözcük 0 kabul edilerek sayılır). Bir negatif argüman ile, önceki komutun sonundan n. sözcük olarak yerleştirilir. n. argüman bir kere hesaplandı mı !n geçmiş yorumlaması belirtilmiş gibi argüman elde edilir,

yank-last-arg (M-. veya M-_)
Son argümanı önceki komuta yerleştirir (önceki geçmiş girdisindeki son sözcük). Bir argüman ile **yank-nth-arg** açıklamasındaki gibi davranır. Peşpeşe yapılan çağrılarla geçmiş listesinde geriye doğru gidilirken her satıra son argüman yerleştirilir. Geçmiş yorumlama oluşumları son argümanı elde etmek için !$ geçmiş yorumlaması belirtilmiş gibi kullanılır.

4.3. Metni Değiştirmek için Komutlar

delete-char (C-d)
İmleç konumunda karakteri siler. İmleç satırın başlangıcında ise, satırda hiç karakter yoksa ve son karakter **delete-char**'a bağlı değilse EOF döner.

backward-delete-char (Rubout)

İmlecin arkasındaki karakteri siler. Bir sayısal argüman karakterlerin silinmesini değil kesilmesini sağlar.

forward-backward-delete-char ()
İmleç satırın sonunda olmadıkça, imlecin altındaki karakteri siler. İmleç satırın sonundaysa imlecin arkasındaki karakter silinir. Öntanımlı olarak bir tuşa atanmamıştır.

quoted-insert (C-q veya C-v)
Sonraki karakteri yazılan satıra aynen ekler. Bu C-q gibi bir tuş dizilerinin girilmesi gibidir.

self-insert (a, b, A, 1, !, ...)
Kendini yerleştirir.

transpose-chars (C-t)
İmleç arkasındaki karakteri peşinden ileri doğru bir karakter sürükler. İmleç satırın sonundaysa, son iki karakter yer değiştirir. Negatif argümanlar etkisizdir.

transpose-words (M-t)
İmleç öncesindeki sözcüğü peşinden bir sözcük ileri sürükler. İmleç satırın sonundaysa son iki sözcük yer değiştirir.

upcase-word (M-u)
İmlecin üzerindeki (ya da sonrasındaki) sözcüğün harfleri büyük harfe çevrilir. Bir negatif argüman ile, önceki sözcük büyük harfli yapılır ama imleç hareket etmez.

downcase-word (M-l)
İmlecin üzerindeki (ya da sonrasındaki) sözcüğün harfleri küçük harfe çevrilir. Bir negatif argüman ile, önceki sözcük küçük harfli yapılır ama imleç hareket etmez.

capitalize-word (M-c)
İmlecin üzerindeki (ya da sonrasındaki) sözcüğün başharfini büyük harfe çevrilir. Bir negatif argüman ile, önceki sözcüğün başharfi büyük harfe çevrilir ama imleç hareket etmez.

overwrite-mode ()
Üsteyazma kipini açar kapar. Doğrudan bir pozitif sayısal argüman ile üsteyazma kipine geçer. Doğrudan bir pozitif olmayan sayısal argümanla ekleme kipine geçer. Bu komut sadece emacs kipini etkiler; vi kipinde bu işlem farklı yapılır. Her **readline()** çağrısı ekleme kipini başlatır.

Üsteyazma kipinde, **self-insert**'e bağlı karakterler metni sağa itmek yerine aynı noktadaki metinle değiştirilir. **backward-delete-char**'a bağlı karakterler imlecin öncesindeki karakteri boşlukla değiştirir.

Öntanımlı olarak bu komut bir tuşa bağlanmamıştır.

4.4. Kesme ve Yapıştırma Komutları

kill-line (C-k)
İmleç konumundan satırın sonuna kadar keser.

backward-kill-line (C-x Rubout)
İmleç konumundan satırın başına kadar keser.

unix-line-discard (C-u)

İmleç konumundan satırın başına kadar keser.

kill-whole-line ()
Satırın tamamını keser, imleç konumundan bağımsızdır. Öntanımlı olarak kısayol atanmamıştır.

kill-word (M-d)
İmleç konumundan sözcüğün sonuna kadar keser, imleç iki sözcüğün arasındaysa sonraki sözcüğün sonuna kadar keser. Sözcük sınırları **forward-word** ile aynıdır.

backward-kill-word (M-DEL)
Metni sözcük başlangıcından imlece kadar keser. İmleç iki sözcüğün arasındaysa, metin önceki sözcüğün başına kadar kesilir. Sözcük sınırları **backward-word** ile aynıdır.

unix-word-rubout (C-w)
Metni imleç konumundan önceki boşluğa kadar keser. Buradaki sözcük sınırları **backward-kill-word**'deki sınırlardan farklıdır.

unix-filename-rubout ()
Sözcük sınırları olarak boşluk ve / karakterini kullanarak imlecin arkasındaki sözcüğü keser. Kesilen metin kesme zincirine kaydedilir.

delete-horizontal-space ()
İmlecin çevresindeki tüm boşlukları ve sekmeleri siler. Öntanımlı olarak kısayol atanmamıştır.

kill-region ()
'Bölge'deki metni keser. Öntanımlı olarak kısayol atanmamıştır.

copy-region-as-kill ()
'Bölge'deki metni keser. Öntanımlı olarak kısayol atanmamıştır.

copy-backward-word ()
İmleçten önceki sözcüğü keser. Sözcük sınırları **backward-word** ile aynıdır. Öntanımlı olarak kısayol atanmamıştır.

copy-forward-word ()
İmleçten sonraki sözcüğü keser. Sözcük sınırları **forward-word** ile aynıdır. Öntanımlı olarak kısayol atanmamıştır.

yank (C-y)
Son kesilen metni imleç konumunda satıra yerleştirir.

yank-pop (M-y)
Her **yank-pop** pano içeriğindeki kesmelerden bir öncekinin seçilmesini sağlar. Pano içeriği **yank-pop** ile sürekli döndürülür. Bu işlem sadece önceki komut **yank** ya da **yank-pop** ise uygulanır.

4.5. Sayısal Argümanların Belirtilmesi

digit-argument (M-0, M-1, ... M--)
Rakamı evvelce başlatılmış argümana ekler ya da bir yeni argüman başlatır. M-- bir negatif argüman başlatır.

universal-argument ()

Bu bir argüman belirtmenin diğer yoludur. Bu komuttan sonra bir veya daha fazla rakam verilirse, istemlik bir eksi işaretiyle, bu rakamlar argümanı tanımlar. Komuttan sonra rakamlar verilmişse **universal-argument**'ın tekrar çalıştırılması sayısal argümanla biter, ama başka türlü yoksayılır. Özel bir durum olarak bu komuttan sonra bir rakam ya da eksi işareti olarak bir karakter verilirse sonraki komut için argüman sayısı dört ile çarpılır. Argüman sayısı dahili olarak birdir, yani bu işlevin ilk defa çalıştırılması argüman sayısını dört yapar, bir ikincisi argüman sayısını 16 yapar ve böyle gider. Öntanımlı olarak kısayol atanmamıştır.

4.6. Readline Sizin Yerinize Yazsın

complete (TAB)
İmlecin arkasındaki metne tamamlama uygulamaya çalışır. Uygulanan asıl tamamlama uygulamaya özeldir. Bash, metni döndürülen bir değişken (metin `$` ile başlıyorsa), kullanıcı ismi (metin `~` ile başlıyorsa), konak ismi (metin `@` ile başlıyorsa) ya da komut (takma adlar ve işlevler) olarak ele alıp tamamlamaya çalışır. Eğer bunlardan hiçbiri bir eşleşme sağlamazsa, dosyaismi tamamlaması yapılmaya çalışılır.

possible-completions (M-?)
İmlecin arkasındaki metnin olası tamamlamalarını listeler.

insert-completions (M-*)
İmlecin arkasındaki metnin **possible-completions** tarafından üretilmiş olan tüm tamamlamalarını yerleştirir.

menu-complete ()
complete'e benzer ancak, tamamlanacak sözcük ile olası tamamlamalar listesindeki tek bir eşleşme yer değiştirilir. **menu-complete** peşpeşe çalıştırılırsa olası tamamlamalar listesi üzerinde adım adım dolaşılarak her eşleşme tek tek satıra yerleştirilir. Tamamlamalar listesinin sonuna gelindiğinde çan çalar (`bell-style` atamasına bağlı olarak) ve orjinal metne dönülür. Bir `n` argümanı eşleşenler listesinde `n` adım ileri gidilmesini sağlar; argüman negatifse listede `n` adım geri gidilir. Bu komut TAB için düşünülmüştür ancak öntanımlı olarak atanmamıştır.

delete-char-or-list ()
İmleç satırın başında ya da sonunda değilse imlecin altındaki karakteri siler (**delete-char** gibi). Satırın sonunda **possible-completions**'a özgü davranır. Bu komuta öntanımlı olarak bir kısayol atanmamıştır.

complete-filename (M-/)
İmlecin arkasındaki metne dosyaismi tamamlaması yapmaya çalışır.

possible-filename-completions (C-x /)
İmlecin arkasındaki metni bir dosyaismi olarak ele alıp, olası tamamlamalarını listeler.

complete-username (M-~)
İmlecin arkasındaki metni bir kullanıcı ismi olarak ele alıp, tamamlamaya çalışır.

possible-username-completions (C-x ~)
İmlecin arkasındaki metni bir kullanıcı ismi olarak ele alıp, olası tamamlamalarını listeler.

complete-variable (M-$)
İmlecin arkasındaki metni bir kabuk değişkeni olarak ele alıp, tamamlamaya çalışır.

possible-variable-completions (C-x $)

İmlecin arkasındaki metni bir kabuk değişkeni olarak ele alıp, olası tamamlamalarını listeler.

complete-hostname (M-@)
İmlecin arkasındaki metni bir konak ismi olarak ele alıp, tamamlamaya çalışır.

possible-hostname-completions (C-x @)
İmlecin arkasındaki metni bir konak ismi olarak ele alıp, olası tamamlamalarını listeler.

complete-command (M-!)
İmlecin arkasındaki metni bir komut ismi olarak ele alıp, tamamlamaya çalışır. Komut tamamlaması metni sırasıyla takma adlar, anahtar sözcükler, kabuk işlevleri, kabuk yerleşikleri ve son olarak çalıştırılabilir dosya isimleri ile tamalamaya çalışır.

possible-command-completions (C-x !)
İmlecin arkasındaki metni bir komut ismi olarak ele alıp, olası tamamlamalarını listeler.

dynamic-complete-history (M-TAB)
İmlecin arkasındaki metni geçmiş listesinin satırları olan olası tamamlamalarla tamamlamaya çalışır.

complete-into-braces (M-{)
Dosyaismi tamamlaması uygular ve olası tamamlamaların listesini kaşlı ayraçlar arasına yerleştirir, böylece liste kabuk tarafından kullanılabilir hale gelir (*Kaşlı Ayraçların Yorumlanması* (sayfa: 23) bölümüne bakınız).

4.7. Klavye Makroları

start-kbd-macro (C-x ()
Yazılan karakterlerin o anki klavye makrosuna kaydını başlatır.

end-kbd-macro (C-x))
Yazılan karakterlerin o anki klavye makrosuna kaydını durdurur ve tanımı kaydeder.

call-last-kbd-macro (C-x e)
Atanan klavye makrosunu, makrodaki karakterler klavyeden yazılmışçasına basarak yeniden çalıştırır.

4.8. Çeşitli Komutlar

re-read-init-file (C-x C-r)
`inputrc` dosyasının içeriğini okur ve burada bulunan değişken atamaları ve tuş kısayolları ile Readline'ı yeniden ilklendirir.

abort (C-g)
o anki düznleme komutundan çıkar ve uçbirim çanını çalar (`bell-style` atamasına bağlı olarak).

do-uppercase-version (M-a, M-b, M-x, ...)
Ötelenmiş karakter x küçük harf ise, bu karakterin büyük harfli karşılığını içeren komutu çalıştırır.

prefix-meta (ESC)
Ardından yazılan karakteri öteler. Bu, bir meta tuşu olmayan klavyeler içindir. `ESC f` ile `M-f` eşdeğerdir.

undo (C-_ or C-x C-u)
Arttırımlı geri alma, her satır için ayrı ayrı hatırlanır.

revert-line (M-r)
Bu satırda yapılan tüm değişiklikleri geri alır. Bu, satır üstünde peşpeşe **undo** komutu girilip tüm değişikliklerin geri alındığı duruma bir kerede erişmeyi sağlar.

tilde-expand (M-&)
O anki sözcük üstünde yaklaşık yorumlaması uygular.

set-mark (C-@)
Noktayı mark yapar. (İmleç konumunu kaydeder). Bir sayısal argüman verilirse, belirttiği konum mark yapılır.

exchange-point-and-mark (C-x C-x)
Nokta ile markı yer değiştirir. O anki imleç kaydedilmiş konuma kaydırılır ve eski imleç konumu mark olarak kaydedilir.

character-search (C-])
Bir karakter okunur ve imleç bu karakterin bulunduğu sonraki konuma gider. Bir negatif argüman karakteri satırın başına doğru arar.

character-search-backward (M-C-])
Bir karakter okunur ve imleç bu karakterin bulunduğu önceki konuma gider. Bir negatif argüman karakteri satırın sonuna doğru arar.

insert-comment (M-#)
Bir sayısal argüman olmaksızın comment-begin değişkeninin değeri o anki satırın başına yerleştirilir. Bir sayısal argüman sağlanmışsa bu komutun değiştirilmesine sebep olur: eğer satırın başındaki karakter comment-begin değeri ile eşleşmiyorsa değer yerleştirilir, aksi takdirde comment-begin'deki karakter satırın başından silinir. Her durumda, satır bir satırsonu tuşlanmış gibi kabul edilir. comment-begin değişkeninin öntanımlı değeri bu komutun o anki satırı bir kabuk açıklaması yapmasına sebep olur. Eğer bir sayısal argüman açıklama karakterinin silinmesine sebep olursa satır kabuk tarafından çalıştırılacaktır.

dump-functions ()
Tüm işlevleri ve bunlarla ilişkili tuş kısayollarını Readline çıktı hattına basar. Bir sayısal argüman verilirse, çıktı inputrc dosyasında girdi olarak kullanılabilecek biçemde basılır. Bu komuta öntanımlı olarak bir kısayol atanmamıştır.

dump-variables ()
Tüm atanabilir değişkenleri ve değerlerini Readline çıktı hattına basar. Bir sayısal argüman verilirse, çıktı inputrc dosyasında girdi olarak kullanılabilecek biçemde basılır. Bu komuta öntanımlı olarak bir kısayol atanmamıştır.

dump-macros ()
Readline tuş dizilerini ve bunların bağlandığı makrolarla dizgeleri Readline çıktı hattına basar. Bir sayısal argüman verilirse, çıktı inputrc dosyasında girdi olarak kullanılabilecek biçemde basılır. Bu komuta öntanımlı olarak bir kısayol atanmamıştır.

glob-complete-word (M-g)
İmlecin öncesindeki sözcük bir * örtük olarak eklenerek dosyayolu yorumlaması için bir kalıp olarak ele alınır. Bu kalıp olası tamamlamalar için eşleşen dosya isimlerinin bir listesini üretmek için kullanılır.

glob-expand-word (C-x *)

İmlecin öncesindeki sözcük dosyayolu yorumlaması için bir kalıp olarak ele alınarak eşleşen dosya isimlerinin listesi sözcükle değiştirilir. Bir sayısal argüman sağlanmışsa dosya yorumlamasının öncesine bir * eklenir.

glob-list-expansions (C-x g)
glob-expand-word tarafından üretilen yorumlama listesi gösterilir ve satır yeniden oluşturulur. Bir sayısal argüman sağlanmışsa dosya yorumlamasının öncesine bir * eklenir.

display-shell-version (C-x C-v)
Bash'in çalışan kopyası hakkında sürüm bilgisi gösterir.

shell-expand-line (M-C-e)
Satırı kabuğun yaptığı gibi yorumlar. Bu takma ad ve geçmiş yorumlaması ile kabuk sözcük yorumlamalarının hemen tamamını uygular (*Kabuk Yorumları* (sayfa: 22) bölümüne bakınız).

history-expand-line (M-^)
O anki satıra geçmiş yorumlaması uygular.

magic-space ()
O anki satıra geçmiş yorumlaması uygular ve bir boşluk girer (*Geçmiş Yorumlaması* (sayfa: 113) bölümüne bakınız).

alias-expand-line ()
O anki satıra takma ad yorumlaması uygular (*Takma Adlar* (sayfa: 75) bölümüne bakınız).

history-and-alias-expand-line ()
O anki satıra geçmiş ve takma ad yorumlaması uygular.

insert-last-argument (M-. or M-_)
yank-last-arg ile eşanlamlıdır.

operate-and-get-next (C-o)
O anki satırı çalıştırmak için kabul eder ve o anki satıra göreli sonraki satırı düzenlemek için geçmişten alır. Her argüman yoksayılır.

edit-and-execute-command (C-xC-e)
O anki komut satırı üstünde bir metin düzenleyici çağırır ve sonucu bir kabuk komutu olarak çalıştırır. Bash metin düzenleyici olarak sırasıyla $VISUAL, $EDITOR ve **emacs** çağırmaya çalışır.

5. Readline vi Kipi

Readline kütüphanesi vi'nin düzenleme işlevlerinin tamamına sahip olmasa da bir satırın düzenlenmesinin basitliği nedeniyle yeterlidir. Readline vi kipinde POSIX standardında belirtildiği gibi davranır.

emacs ve vi düzenleme kipleri arasında etkileşimli olarak geçiş yapmak için **set** (sayfa: 49)-o emacs ve **set**-o vi komutları kullanılır. Readline öntanımlı olarak emacs kipindedir.

vi kipindeyken bir satır girerseniz, kendinizi i tuşuna basıldığında girilen 'üzerine yazma' kipinde bulursunuz. ESC tuşuna basarak 'komut' kipine geçersiniz. Böylece standart vi hareket tuşları ile satırı düzenleyebilirsiniz. Örneğin k ile geçmiş listesindeki önceki satırlara, j ile müteakip satırlara hareket edebilirsiniz.

6. Programlanabilir Tamamlama

complete (sayfa: 107) yerleşiği kullanılarak etkinleştirilen bir tamamlama belirtiminin (compspec) bir komutun bir argümanı için sözcük tamamlamasında kullanılmaya çalışılması halinde, programlanabilir tamamlama araçları çağrılmış olur.

Önce komut ismi saptanır. Bu komut için bir tamamlama belirtimi tanımlanmışsa, bu, sözcüğün olası tamamlamalarını üretir. Komut tam dosyayolu belirtilerek verilmişse ilk olarak bu dosyayoluna göre bir tamamlama belirtimi aranır. Tam dosyayolu için bir tamamlama belirtimi bulunmuyorsa, son / işaretinden sonraki kısım için bir tamamlama belirtimi bulunmaya çalışılır.

Bir tamamlama belirtimi bir kere bulunduktan sonra, onunla, uygun sözcüklerin bir listesi üretilir. Bir tamamlama belirtimi yoksa, *Readline Sizin Yerinize Yazsın* (sayfa: 102) bölümünde bahsedilen öntanımlı Bash tamamlaması uygulanır.

İlk olarak, tamamlama belirtimi tarafından belirtilen eylemler kullanılır ve sadece sözcük ile başlayan tamamlamalar döndürülür. Dosyaismi veya dizin ismi tamamlaması için `-f` ya da `-d` seçeneği kullanıldığında, `FIGNORE` *kabuk değişkeni* (sayfa: 62) uyuşanları süzmek için kullanılır.

Sonra, `-G` seçeneği ile bir dosyaismi yorumlama kalıbı verilerek tamamlamalar üretilir. Sözcüklerin üretildiği kalıbın tamamlanması istenen sözcükle eşleşmemesi gerekir. Eşleşenleri süzmekte `GLOBIGNORE` *kabuk değişkeni* (sayfa: 62) değil, `FIGNORE` kabuk değişkeni kullanılır.

Sonra da `-W` seçeneğine argüman olarak dizgenin belirtildiği varsayılır. `IFS` özel değişkenindeki karakterler ayraç olarak kullanılarak dizge önce ayrıştırılır. Kabuk tarafından tırnaklama uygulanır. Her sözcüğe *kaşlı ayraç yorumlaması* (sayfa: 23), *yaklaşık yorumlaması* (sayfa: 23), *parametre yorumlaması* (sayfa: 24), *komut ikamesi* (sayfa: 26) ve *aritmetik yorumlama* (sayfa: 27) uygulanır. Sonuçlara *sözcüklere ayırma* (sayfa: 27) uygulanır. Yorumlama sonuçları tamamlanması istenen sözcüğün önek olduğu sözcüklerdir ve eşleşen sözcükler olası tamamlamalardır.

Bu eşleşmeler üretildikten sonra, `-F` ve `-C` seçenekleri ile bir kabuk işlevi ya da bir komut çağrılır ve `COMP_LINE` *ile* `COMP_POINT` (sayfa: 61) değişkenlerine değerleri atanır. Bir kabuk işlevi çağrılmışsa ayrıca `COMP_CWORD` (sayfa: 61) ve `COMP_WORDS` (sayfa: 61) değişkenleri de atanır. İşlev veya komut çağrıldığında, ilk argüman argümanları tamamlanacak olan komutun ismidir, ikinci argüman tamamlanan sözcüktür ve üçüncü argüman komut satırındaki tamamlanması istenen sözcüğü önceleyen sözcüktür. Tamamlanması istenen sözcüğe karşılık üretilen tamamlamalara süzme uygulanmaz; işlev ya da komut üretilen eşleşmelerle serbestçe tamamlanır.

Bir işlev ilk olarak‘ `-F` seçeneğine belirtilerek çağrılır. İşlev, eşleşmeleri üretmek için **compgen** *yerleşiği* (sayfa: 107) dahil, her kabuk aracını kullanabilir. Olası tamamlamaları `COMPREPLY` (sayfa: 61) dizi değişkenine yerleştirmelidir.

Sonra, komut ikamesine eşdeğer bir ortam içinde bir komut `-C` seçeneğine belirtilerek çağrılır. Bu, komut tamamlamalarının listesini her biri bir satırda standart çıktıya basmalıdır. Gerekliyse, tersbölü bir satırsonunu öncelemekte kullanılabilir.

Tüm olası tamamlamalar üretildikten sonra `-X` seçeneğine belirtilen bir süzgeç listeye uygulanır. Süzgeç dosyayolu yorumlaması için kullanılan bir kalıptır; kalıptaki bir `&` karakteri tamamlanması istenen sözcük metni ile değiştirilir. Kalıba uyan tamamlamalar listeden kaldırılacaktır. `!` ile başlayan bir kalıp zıt eşleme yapar; bu durumda, kalıba uymayan tamamlamalar listeden kaldırılacaktır.

Son olarak, `-P` ve `-S` seçenekleri ile belirtilen bir önek ve bir sonek tamamlama listesinin her üyesine eklenir ve sonuç Readline tamamlama koduna olası tamamlamaların listesi olarak döndürülür.

Eğer öncelikle uygulanan eylemler bir eşleşme üretmezse ve tamamlama belirtimi tanımlanırken **complete** (sayfa: 107) ile `-o dirnames` seçeneği verilmişse, dizin ismi tamamlanmaya çalışılır.

Eğer tamamlama belirtimi tanımlandığında `-o plusdirs` seçeneği **complete** komutuna sağlanmışsa dizin ismi tamamlanmaya çalışılır ve eşleşenler diğer eylemlerin sonuçlarına eklenir.

Öntanımlı olarak, bir tamamlama belirtimi varsa, onun ürettiği her ne varsa olası tamamlamaların bir tam kümesi olarak tamamlama koduna döner. Öntanımlı Bash tamamlamaları uygulanmaz ve Readline'ın öntanımlısı olan dosyaismi tamamlaması kapatılır. Eğer tamamlama belirtimi tanımlıyken **complete** yerleşiği `-o bashdefault` seçeneği ile çalıştırılmışsa ve tamamlama balirtimi bir eşleşme üretmiyosa öntanımlı Bash tamamlamaları uygulanmaya çalışılır. Tamamlama belirtimi tanımlıyken `-o default` seçeneği ile çalıştırılmışsa ve tamamlama belirtimi (ve eğer uygulanmışsa öntanımlı Bash tamamlamaları) hiç eşleşme üretememişse, Readline'ın öntanımlı tamamlaması uygulanacaktır.

Bir tamamlama belirtimi dizin ismi tamamlaması istendiğini belirtiyorsa programlanabilir tamamlama işlevleri Readline'ı dizinlere sembolik bağ olan ve `mark-symlinked-directories` Readline değişkeninin değerine bakmaksızın `mark-directories` Readline değişkeninin değerine konu olan isimleri tamamlamak için / eklemeye zorlar.

7. Programlanabilir Tamamlama Yerleşikleri

Programlanabilir tamamlama araçlarını yönetmek için iki yerleşik komut vardır.

7.1. Compgen Yerleşiği

compgen [*seçenek*] [*sözcük*]

sözcük ile eşleşen olası tamamlamaları, `-p` ve `-r` seçenekleri dışında **complete** yerleşiği tarafından kabul edilen herhangi bir seçenek olabilen *seçenek*lere göre üretir ve eşleşenleri standart çıktıya yazar. `-F` veya `-C` seçenekleri kullanıldığında, çeşitli kabuk değişkenleri programlanabilir tamamlama araçları tarafından kullanılana kadar yararlı olmayacak değerlerle atanır.

Programlanabilir tamamlama kodunun, bir tamamlama belirtiminden aynı seçeneklerle doğrudan ürettiği gibi eşleşmeler üretilmiş olacaktır. *sözcük* belirtilmişse bu eşleşmelerin sadece, *sözcük* ile eşleşenleri gösterilmiş olacaktır.

Geçersiz bir seçenek verilmedikçe ya da hiç eşleşme üretilmemesi dışında dönüş durumu sıfırdır.

7.2. Complete Yerleşiği

complete [**-abcdefgjksuv**] [**-o** *tamamlama–seçeneği*] [**-A** *eylem*] [**-G** *genel–kalıp*]
[**-W** *sözcük–listesi*] [**-P** *önek*] [**-S** *sonek*] [**-X** *süzgeç–kalıbı*]
[**-F** *işlev*] [**-C** *komut*] *isim* [*isim* ...]

complete -pr [*isim* ...]

Argümanların her *isim*e nasıl tamamlanması gerektiği belirtilir. `-p` seçeneği verilirse veya hiç seçenek verilmezse, mevcut tamamlama belirtimleri girdi olarak yeniden kullanılabilir şekilde basılır. `-r` seçeneği her *isim* için tamamlama belirtimini kaldırır, hiç *isim* belirtilmezse tümü kaldırılır.

Sözcük tamamlaması yapılmaya çalışıldığında, bu tamamlama belirtimlerinin uygulanma süreci *Programlanabilir Tamamlama* (sayfa: 105) bölümünde açıklanmıştır.

Diğer seçenekler belirtildiklerinde aşağıdaki anlamlara gelirler. `-G`, `-W` ve `-X` (ve lazımsa `-P` ve `-S`) seçeneklerinin argümanları, **complete** yerleşiği çağrılmadan önce yorumlanmaması için tırnak içine alınmış olmalıdır.

-o *tamamlama–seçeneği*
: *tamamlama–seçeneği* tamamlamaların basit üretimi dışında tamamlama belirtiminin çeşitli davranış biçimlerini kontrol eder. *tamamlama–seçeneği* aşağıdakilerden biri olabilir:

  `bashdefault`

Tamamlama belirtimi hiç eşleşme üretemezse öntanımlı Bash tamamlamalarının kalanı uygulanır.

`default`
Tamamlama belirtimi hiç eşleşme üretemezse Readline'ın öntanımlı dosyaismitamamlaması kullanılır.

`dirnames`
Tamamlama belirtimi hiç eşleşme üretemezse, dizin ismi tamamlaması uygulanır.

`filenames`
Readline'a tamamlama belirtiminin dosyaisimlerini üretmesini söyler, böylece dosyaismine özel işlemleri uygulayabilir (dizin isimlerine / eklemek veya sondaki boşlukların kaldırılması gibi işlemler). Bu seçenek, `-F` seçeneği ile belirtilen kabuk işlevleriyle kullanmak için düşünülmüştür.

`nospace`
Readline'a bir sözcüğü tamamlamak için satır sonundayken bir boşluk eklememesini söyler.

`plusdirs`
Tamamlama belirtimi tarafından tanımlanmış eşleşmeler üretildikten sonra dizin ismi tamamlanmaya çalışılır ve eşleşenler diğer eylemlerin sonuçlarına eklenir.

`-A` *eylem*
eylem, olası tamamlamaların bir listesinin üretilmesi için kullanılan aşağıdaki seçeneklerden biridir:

`alias`
Takma adlar. `-a` seçeneği ile aynıdır.

`arrayvar`
Dizi değişkeni isimleri.

`binding`
Readline tuş kısayollarının isimleri (*Kısayollar için Readline Komutları* (sayfa: 98) bölümüne bakınız).

`builtin`
Kabuk yerleşik komutlarının isimleri. `-b` seçeneği ile aynıdır.

`command`
Komut isimleri. `-c` seçeneği ile aynıdır.

`directory`
Dizin isimleri. `-d` seçeneği ile aynıdır.

`disabled`
İptal edilmiş kabuk yerleşikleri.

`enabled`
Etkin kabuk yerleşiklerinin isimleri.

`export`
Ortama aktarılmış kabuk değişkenlerinin isimleri. `-e` seçeneği ile aynıdır.

`file`
Dosya ismi. `-f` seçeneği ile aynıdır.

`function`
Kabuk işlevlerinin isimleri.

`group`
Grup isimleri. `-g` seçeneği ile aynıdır.

`helptopic`
`help` *yerleşik komutu* (sayfa: 47) tarafından kabul edilen yardım konuları.

`hostname`
`HOSTFILE` *kabuk değişkeni* (sayfa: 63) tarafından belirtilen dosyadan alınan konak isimleri.

`job`
İş denetimi etkinse, iş isimleri. `-j` seçeneği ile aynıdır.

`keyword`
Kabuk anahtar sözcükleri. `-k` seçeneği ile aynıdır.

`running`
İş denetimi etkinse, çalışan işlerin isimleri.

`service`
Servis isimleri. Ayrıca **`-s`** olarak da belirtilebilir.

`setopt`
`set` *yerleşik komutunun* (sayfa: 49) `-o` seçeneğinin geçerli argümanları.

`shopt`
`shopt` *yerleşik komutunun* (sayfa: 52) kabul ettiği kabuk seçeneklerinin isimleri.

`signal`
Sinyal isimleri

`stopped`
İş denetimi etkinse, durmuş işlerin isimleri.

`user`
Kullanıcı isimleri. `-u` seçeneği ile aynıdır.

`variable`
Tüm kabuk değişkenlerinin isimleri. `-v` seçeneği ile aynıdır.

`-G` *genel–kalıp*
Dosyaismi yorumlama kalıbı *genel–kalıp*, olası tamamlamaları üretmek için yorumlanır.

`-W` *sözcük–listesi*
sözcük–listesi `IFS` özel değişkenindeki karakterler ayraç olarak kullanılarak ayrıştırılır ve elde edilen sözcük yorumlanır. Olası tamamlamalar, tamamlanması istenen sözcükle eşleşen sonuç listesinin üyeleridir.

`-C` *komut*
komut bir altkabuk ortamında çalıştırılır ve çıktısı olası tamamlamalar olarak kullanılır.

`-F` *işlev*
Kabuk işlevi *işlev* bulunulan kabuk ortamında çalıştırılır. Bittiğinde olası tamamlamalar `COMPREPLY` (sayfa: 61) dizi değişkeninden alınır.

`-X` *süzgeç–kalıbı*

süzgeç–kalıbı dosyaismi yorumlaması için kullanılan bir kalıptır. Önceki seçenek ve argümanlar tarafından üretilen olası tamamlamaların listesine uygulanır ve *süzgeç–kalıbı* ile eşleşen her tamamlama listeden kaldırılır. Bir `!` ile öncelenirse kalıp zıt işlem yapar yani, eşleşmeyenler listeden kaldırılır.

-P *önek*
Tüm seçenekler uygulandıktan sonra her olası tamamlamanın başlangıcına *önek* eklenir.

-S *sonek*
Tüm seçenekler uygulandıktan sonra her olası tamamlamanın sonuna *sonek* eklenir.

Bir geçersiz seçenek verilmedikçe, `-p` veya `-r` seçenekleri dışında bir seçenek bir isim argümanı ile verilmedikçe, hiç belirtimi olmayan bir isim için bir tamamlama belirtimi kaldırılmaya çalışılmadıkça veya bir tamamlama belirtimine eklemenin bir hata oluşturması dışında dönüş durumu sıfırdır.

IX. Geçmişin Etkileşimli Kullanımı

İçindekiler

Bu oylumda GNU Geçmiş Kitaplığının etkileşimli olarak nasıl kullanılacağı anlatılmış, bir kullanıcı kılavuzu olarak ele alınmıştır. GNU Geçmiş Kitaplığının diğer uygulamalarla kullanımı konuları için GNU Readline Kitaplığı Kılavuzuna bakınız.

1. Bash'in Geçmişsel Yetenekleri

set (sayfa: 49) yerleşiğini `-o history` seçeneği ile çalıştırıp geçmişi etkinleştirdiğinizde, kabuk önceden kullandığınız komutların bir listesi olan *komut geçmişi*ne erişim sağlar. `HISTSIZE` kabuk değişkeninin değeri geçmiş listesindeki kayıtların sayısı olarak kullanılır. Daima son `$HISTSIZE` komut (öntanımlı 500) saklanır. Kabuk her komutu, `HISTIGNORE` ve `HISTCONTROL` kabuk değişkenlerinin değerlerine bağlı olarak geçmiş yorumlaması uyguladıktan sonra, parametre ve değişken yorumlamalarını uygulamadan önce geçmiş listesine alır.

Kabuk başlatıldığında `HISTFILE` değişkeninde ismi kayıtlı olan dosyadan (öntanımlı değeri `~/.bash_history`) geçmişi ilklendirir. Bu dosyanın satır sayısı `HISTFILESIZE` değişkenindeki değerden fazla ise, dosya boyutu fazlalıklar atılarak ayarlanır. Bir etkileşimli kabuk çıkarken geçmiş listesindeki son `$HISTSIZE` komutu `$HISTFILE` isimli dosyaya kopyalar. `histappend` *kabuk seçeneği* (sayfa: 54) etkinse geçmiş listesindeki komutlar dosyaya eklenir, değilse dosyanın üzerine yazılır. `HISTFILE` atanmamışsa veya geçmiş dosyası yazılabilir değilse geçmiş kaydedilmez. Geçmiş kaydedildikten sonra, geçmiş dosyası `$HISTFILESIZE` satırdan fazla satır içeriyorsa fazla satırlar dosyadan kaldırılır. `$HISTFILESIZE` atanmamışsa fazla satırlar kaldırılmaz.

HISTTIMEFORMAT atanmışsa her geçmiş girdisi ile ilgili zaman damgası bilgisi geçmiş dosyasına yazılır.

fc *yerleşik komutu* (sayfa: 111) ile geçmiş listesi, listelenebilir, düzenlenebilir ve listenin bir bölümü yeniden çalıştırılabilir. **history** *yerleşik komutu* (sayfa: 112) ile geçmiş listesini görebilir, değiştirebilir ve geçmiş dosyasını değiştirebilirsiniz. Komut satırı düzenlemesi yaparken, geçmiş listesine erişim sağlayan her düzenleme kipinde arama komutları kullanılabilir (*Geçmişi Yöneten Komutlar* (sayfa: 98) bölümüne bakınız).

Kabuk, geçmiş listesinde kayıtlı komutların üzerinde denetime izin verir. `HISTCONTROL` ve `HISTIGNORE` değişkenleri kabuğun, girilen komutların belli bir bölümünün kaydedilmesini sağlamakta kullanılabilir. `cmdhist` (sayfa: 53) kabuk seçeneği etkinse, bir çok satırlı komut, sözdizimsel doğruluğun korunmasını sağlayacak yerlerine **;** işaretleri yerleştirilerek bütün satırları tek geçmiş girdisi olarak kaydedilmeye çalışılır. `lithist` (sayfa: 55) kabul seçeneği etkinse, bir çok satırlı komut, **;** işaretleri yerine kendi içindeki satırsonu karakterleri ile kaydedilir. Bu seçenekleri etkinleştirmek için **shopt** *yerleşik komutu* (sayfa: 52) kullanılır.

2. Bash Geçmiş Yerleşikleri

Geçmiş listesinde ve geçmiş dosyasında değişiklik yapmak için kullanılabilecek iki yerleşik komut vardır.

2.1. Fc Yerleşiği

```
fc [-e düzenleyici–ismi] [-nlr] [ilk] [son]
fc -s [eski=yeni] [komut]
```

Komutun ismi düzeltme komutu anlamına gelen **F**ix **C**ommand sözcüklerinin baş harflerinden türetilmiştir. İlk sözdiziminde, geçmiş listesinin *ilk* ve *son* kayıtları arasındaki komutlar seçilebilir. *ilk* ve *son* birer dizge olarak belirtilebileceği gibi birer sayı da olabilir. Dizgelerle komutların kendileri, numaralarla ise kaydın listedeki sıra numarası belirtilir. Değer negatif bir sayı ise sayma son komuttan eskiye doğrudur. *son* verilmezse, *ilk*'e eşitlenir. *ilk* verilmezse, listelemek için –16 kabul edilir ve düzenlemek için sondan bir önceki komut seçilir. `-l` seçeneği verilmişse, komutlar standart çıktıya listelenir. `-n` seçeneği de verilirse, listleme sırasında satır numaraları gösterilmez. `-r` seçeneği ile birlikte, listelemenin ters sıralamayla yapılmasını sağlar. `-l` seçeneği verilmezse, seçilen aralıktaki komutlar *düzenleyici–ismi* ile verilen metin düzenleyicide gösterilir. *düzenleyici–ismi* verilmemişse, `${FCEDIT:-${EDITOR:-vi}}` değişken yorumlaması kullanılır. Bu yorumlama sonucunda önce etkin bir `FCEDIT` değişkeninin varlığına bakılır, yoksa `EDITOR` değişkenine bakılır, o da yoksa **vi** çalıştırılır. Düzenleme bittiğinde düzenlenen komutlar standart çıktıya yansıtılır ve çalıştırılır.

İkinci sözdiziminde, *komut*'un içinde *eski*'lerin yerine *yeni*'ler yerleştirilerek *komut* tekrar çalıştırılır.

fc komutu yerine kullanılabilecek en faydalı *takma ad* (sayfa: 75) `r='fc -s'` olurdu. Örneğin sadece `r` yazarak son komutu tekrar çalıştırabilirsiniz, ya da `r cc` yazarak **cc** ile başlayan son komutu tekrar çalıştırabilirsiniz.

2.2. History Yerleşiği

```
history [-n]
history [-c]
history -d konum
history [-anrw] [dosyaismi]
history -ps argüman
```

Seçeneksiz kullanıldığında geçmiş listesini satır numaraları ile gösterir. Değişiklik yapılmış satırlarda numaralardan önce bir `*` vardır. *argüman* olarak bir sayı verildiğinde, son *argüman* sayıda komut listelenir. **HISTTIMEFORMAT** kabuk değişkeni tanımlanmış ve null değilse gösterilen her geçmiş girdisi ile ilişkili zaman damgasını gösterecek `strftime` için biçim dizgesi olarak kullanılır. Biçimli zaman damgası ile geçmiş satırı arasına birşey basılmaz.

Seçeneklerin anlamları:

`-c`
: Geçmiş listesini temizler. Geçmiş listesini toptan değiştirmek için bu seçenek diğer seçeneklerle birlikte kullanılabilir.

`-d` *konum*
: Satır numarası *konum* olan geçmiş girdisini siler. *konum* silinecek satırın geçmiş listesinde gösterilen numarası olmalıdır.

`-a`
: Bash oturumunun başlangıcından itibaren girilen geçmiş satırları geçmiş dosyasına eklenir.

`-n`
: Geçmiş dosyasından henüz okunmamış olan geçmiş satırları, Bash oturumunun başlangıcından beri oluşan geçmiş listesine eklenir.

`-r`

Geçmiş dosyası okunur ve içeriği geçmiş listesine eklenir.

`-w`
Geçmiş listesini geçmiş dosyasına yazar.

`-p`
argümanlar üzerinde geçmiş yorumlaması uygular ve sonuçları standart çıktıda gösterir. Sonuçlar geçmiş listesinde saklanmaz.

`-s`
argümanlar geçmiş listesinin sonuna tek girdi olarak eklenir.

`-w`, `-r`, `-a` veya `-n` seçenekleri ile verilen *dosyaismi*, geçmiş dosyası olarak kullanılır. *dosyaismi* verilmezse, `HISTFILE` değişkenindeki değer kullanılır.

3. Geçmiş Yorumlaması

Geçmiş kütüphanesi, **csh** tarafından kullanılana benzer bir geçmiş yorumlaması özelliğine sahiptir. Bu kısımda geçmiş bilgilerinin yönetiminde kullanılan sözdizimi açıklanacaktır.

Geçmiş yorumlaması geçmiş listesindeki girdilerin standart girdiye aktarılmasıyla, komutların tekrarlanabilmesini, önceki komutların girdi satırına alınıp değiştirilerek tekrar kullanımını veya önceki komutlarda yapılan hataların düzeltilmesini kolaylaştırır.

Geçmiş yorumlaması iki bölümde ele alınabilir. İlki, yorumlama sırasında geçmiş listesindeki hangi satırın kullanılacağının saptanması, ikincisi de alıntılamak için satırın bazı parçalarının seçilmesidir. Geçmişten seçilen satıra *eylem* ve satırdan seçilen parçalara da *sözcükler* denir. Seçilen sözcükleri yönetmek için çeşitli *değiştiriciler* bulunur. Bash'in bazı sözcükleri tırnak içine alarak tek sözcük haline getirmesine benzer bir yolla satır sözcüklere ayrılır. Geçmiş yorumlaması, *geçmiş yorumlama karakteri* adı verilen bir karakterinin varlığı ile anlaşılır. Bu karakter öntanımlı olarak `!`'dir. Geçmiş yorumlama karakteri sadece `\` ve `'` karakterleri ile öncelenebilir.

shopt (sayfa: 52) yerleşiği ile etkinleştirilen kabuk seçeneklerinden bazıları geçmiş yorumlamasının davranışlarını değiştirmekte kullanılabilir. `histverify` (sayfa: 55) kabuk seçeneği etkinleştirildiğinde, Readline kullanılıyorsa geçmiş yorumlaması doğrudan kabuk çözümleyicisine aktarılmaz. Yorumlanan satır, değişiklik yapılabilsin diye önce Readline düzenleme tamponuna yüklenir. `histreedit` (sayfa: 55) kabuk seçeneği etkinleştirildiğinde, Readline kullanılıyorsa başarısız olmuş bir geçmiş yorumlaması düzeltilebilmesi için Readline düzenleme tamponuna tekrar yüklenir. **history** (sayfa: 112) yerleşik komutunun `-p` seçeneği, uygulanmadan önce geçmiş yorumlamasının ne yapacağını görmek için, `-s` seçeneği de komutların çalıştırılmaksızın geçmiş listesinin sonuna eklenmesinde kullanılabilir, böylece ilerde bu komutların geçmiş listesinden yeniden çağrılması mümkün olabilir. Bu, Readline ile bağlantılı olarak oldukça kullanışlıdır.

`histchars` (sayfa: 62) kabuk değişkeni ile kabuk, çeşitli karakterlerin geçmiş yorumlama mekanizması tarafından kullanılmasına izin verir.

3.1. Eylem Belirticiler

Bir eylem belirtici, geçmiş listesindeki bir komut satırı girdisine bir göstergedir.

`!`
Kendinden sonra boşluk, sekme, satırsonu, `=` veya `(` karakteri gelmedikçe bir geçmiş yorumlamasını başlatır (**shopt** yerleşiği kullanılarak `extglob` kabuk seçeneği etkinleştirildiğinde).

`!n`
Geçmiş listesindeki *n.* satıra karşılıktır.

`!-n`
: Geçmiş listesindeki geriye doğru *n.* satıra karşılıktır.

`!!`
: Bir önceki komuta karşılıktır. `!-1`'e eşdeğerdir.

`!dizge`
: *dizge* ile başlayan son komuta karşılıktır.

`!?dizge[?]`
: İçinde *dizge* geçen son komuta karşılıktır. *dizge*'den hemen sonra bir satırsonu geliyorsa `?` verilmeyebilir.

`^dizge1^dizge2^`
: Hızlı Yorumlama. *dizge1* ile *dizge2* yerdeğiştirilerek bir önceki komut tekrarlanır. `!!:s/dizge1/dizge2/` ile eşdeğerdir.

`!#`
: O ana kadar yazılan komut satırının tümüne karşılıktır. Örneğin, `type !#` yazıp deneyin.

3.2. Sözcük Belirticiler

Sözcük belirteçleri, eylemden istenen sözcükleri seçmek için kullanılır. Bir `:` karakteri eylem belirtimini sözcük belirticiden ayırır. Sözcük belirtici bir `^`, `$`, `*`, `-`, veya `%` karakteri ile başlıyorsa `:` verilmeyebilir. Sözcükler satırın başından itibaren numaralanır ve ilk sözcüğün numarası 0 (sıfır)dır. Sözcükler satıra birer boşlukla ayrılarak girilir.

Örneğin,

`!!`
: önceki komutu belirtir.

`!!:$`
: Önceki komutun son argümanına karşılıktır. Kısaltması olarak `!$` da yazılabilir.

`!fi:2`
: `fi` harfleriyle başlayan son komutun ikinci argümanını belirtir.

Bunlar sözcük belirticidir:

`0 (sıfır)`
: 0. sözcük. Çoğu durumda bu komut sözcüğüdür.

`n`
: *n.* sözcük

`^`
: İlk argüman; yani 1. sözcük.

`$`
: Son argüman.

`%`
: En son *?dizge?* aramasındaki sözcük.

`x-y`
: Sözcük aralığı; *x*`-`*y*, `0`*-y* ile aynıdır.

`*`
: 0. hariç, sözcükleri tümü. `1-$` ile eşdeğerdir. Eylem sadece bir sözcük içerse bile `*` hata vermez, sadece boş dizge döner.

`x*`
: `x-$` için kısaltma.

`x-`
: `x-$`ın `x*`'a benzer kısaltmasıdır, ancak son sözcük atlanır.

Bir sözcük belirtici, bir eylem belirtimi olmaksızın verilirse önceki komut eylem olarak kullanılır. Örneğin, `type` komutunu girdikten sonra yeni satır olarak `!! !!` yazın. İlki eylem, ikincisi sözcük belirtir.

3.3. Değiştiriciler

İsteğe bağlı olarak sözcük belirticiden sonra, herbiri bir `:` ile öncelenerek aşağıdaki değiştiricilerden bir veya birkaçını ekleyebilirsiniz.

`h`
: Bir dosya yolundan son dizini kaldırır. Örneğin, `cd /home/nilgun/belgeler` komutundan sonra `cd !$:h` yazarsanız `/home/nilgun` dizinine geçilir.

`t`
: Bir dosya yolundan geriye son dizin kalacak şekilde baştakileri kaldırır. Yukardaki örneğe devamla `cd !$:t` yazarsanız `cd nilgun`'e karşılık olur ki, "böyle bir dosya ya da dizin yok" iletisi alınabilir.

`r`
: *.sonek* şeklindeki bir soneki kaldırır, sadece esas dosya ismi (basename) kalır.

`e`
: sonek hariç herşeyi kaldırır.

`p`
: Son komutu basar ama çalıştırmaz. Örnek: `!:p`

`q`
: Yerini aldığı sözcükleri, gerekirse önceleyerek, tırnak içine alır,

`x`
: `q` gibidir ancak sözcüklerin arasındaki boşlukları, sekmeleri ve satırsonlarını koruyarak sözcükleri tek tek tırnak içine alır.

`s/eski/yeni/`
: Eylem satırındaki ilk *eski*'nin yerine *yeni*'yi koyar (ikame). `/` yerine herhangi bir ayraç kullanılabilir. *eski* ile *yeni* arasındaki ayraç bir tek tersbölü ile tırnaklanabilir. *yeni* içinde `&` varsa *eski* ile değiştirilir. Bir tek tersbölü `&` karakterini önceleyecektir. Girdi satırındaki son karakter olduğunda son ayraç isteğe bağlıdır.

`&`
: Önceki ikameyi tekrarlar.

`g`
`a`
: Değişikliklerin eylem satırının tümü özerinde uygulanmasını sağlar. `s` ile birleşik olarak `gs/eski/yeni/`'deki gibi veya `&` ile birleşik kullanılır.

`G`
: Olaydaki her sözcüğe izleyen **`s`** değiştiricisini bir kere uygular.

X. Bash Kurulumu

İçindekiler

Bu oylumda desteklendiği platformlarda Bash kurulumu için temel talimatlar bulunmaktadır. Dağıtım, GNU işletim sistemlerini, Unix'in hemen her sürümünü, BeOS ve Interix gibi çeşitli Unix olmayan sistemleri destekler. MS–DOS, OS/2, Windows platformları için başka bağımsız dağıtımlar da vardır.

1. Temel Kurulum

Bash için kurulum talimatları.

Bash'i derlemenin en basit yolu:

1. Kaynak kodun bulunduğu dizine **cd** yapın ve Bash'i sisteminize göre yapılandırmak için `./configure` yazın. Bir eski sürüm System V üzerinde **csh** kullanıyorsanız, **csh**'ın `configure` dosyasının kendisini çalıştırmayı denemesinden korunmak yerine **sh.**`/configure` yazmalısınız. `configure` dosyasının çalışması biraz zaman alır. Çalışırken hangi özellikleri denetlediğini belirten iletiler basar.
2. Bash'ı derlemek için **make** yazın ve **bashbug** hata raporlama betiğini kurgulayın.
3. İstemlik olarak, testleri çalıştırmak için **make tests** yazın.
4. **bash** ve **bashbug**'ı kurmak için **make install** yazın. Bu ayrıca kılavuz sayfalarını ve Info dosyasını da kuracaktır.

configure kabuk betiği derleme sırasında çeşitli sistem bağımlı değişkenler için doğru değerleri tahmin etmeye çalışır. Bu değerleri paketin her dizinindeki `Makefile` dosyalarını oluşturmakta kullanır (paket içinde tepe dizin, `builtins`, `doc` ve `support` dizinleri ve bu dizinlerin altındaki `lib` ve çeşitli başka dizinlerden oluşur). Betik ayrıca sistem bağımlı tanımları içeren `config.h` dosyasını oluşturur. Son olarak, o anki yapılandırmayı tekrar oluşturmak için çalıştırabileceğiniz `config.status` isimli bir kabuk betiği, yeniden yapılandırmayı hızlandıracak test sonuçlarının kaydedildiği `config.cache` diye bir dosya ve derleyici çıktısını içeren `config.log` (yapılandırma hatalarını ayıklamak için faydalıdır) dosyasını oluşturur. Bazı noktalarda istemediğiniz sonuçları içeriyorsa, `config.cache` dosyasını silebilir ya da düzenleyebilirsiniz.

configure betiğinin anladığı seçenekler ve argümanlar hakkında daha fazla bilgi edinmek için Bash kaynak dizininizde Bash komut istemine

```
bash-2.04$ ./configure --help
```

yazın.

Bash derlerken olağandışı şeyler yapmaya ihtiyacınız varsa, lütfen "onların yapılıp yapılamayacağını **configure** nasıl denetlemeliydi" meselesini halletmeyi deneyin ve farklılıkları veya talimatları <bash–maintainers (at) gnu.org> adresine postalayın ki, gelecek dağıtım için onlar hesaba katılabilsin.

`configure.in` dosyası Autoconf diye bir program tarafından **`configure`** betiğini oluşturmakta kullanılır. `configure.in` dosyasını **`configure`** betiğini daha yeni bir Autoconf sürümünü kullanarak yeniden üretmek veya değiştirmek isterseniz **`Autoconf`** sürüm 2.50 veya daha yeni bir sürümünü kullanmalısınız.

Program çalıştırılabilirlerini ve nesne dosyalarını kaynak kod dizinlerinden **`make clean`** yazarak kaldırabilirsiniz. Ayrıca **`configure`** betiğinin oluşturduğu dosyaları da kaldırmak isterseniz (Böylece Bash'i başka bir makina için derleyebilirsiniz) **`make distclean`** yazın.

2. Derleyiciler ve Seçenekler

Bazı sistemler **`configure`** betiğinin bilmediği olağandışı derleme ve ilintileme seçenekleri gerektirir. **`configure`** betiğinin iç dağişkenlerinin değerlerini ortamda atayarak bunları verebilirsiniz. Bir Bourne uyumlu kabuk kullanarak bunun gibi bir komut satırı ile bunu yapabilirsiniz:

```
CC=c89 CFLAGS=-O2 LIBS=-lposix ./configure
```

`env` programı kullanılan bazı sistemlerde ise bunu şöyle yapabilirsiniz:

```
env CPPFLAGS=-I/usr/local/include LDFLAGS=-s ./configure
```

Yapılandırma süreci Bash'i kurgulamak için varsa GCC'yi kullanır.

3. Çoklu Mimariler için Derleme

Bash'i birden fazla mimari için, her mimarinin nesne dosyalarını kendi dizinlerine yerleştirerek bir seferde derleyebilirsiniz. Bunu yapabilmek için **`make`**'in `VPATH` değişkenini destekleyen GNU **`make`** gibi bir sürümünü kullanmanız gerekir. Nesne dosyalarının ve çalıştırılabilirlerin bulunacağı dizine geçerek kaynak dizinindeki **`configure`** betiğini çalıştırabilirsiniz. Bunu yaparken **`configure`** betiğini kaynak dosyaların nerede bulunacağını belirten `--srcdir=KaynakDizini` seçeneği ile çalıştırmak gerekebilir. **`configure`** betiği kaynak dizinini öntanımlı olarak bulunduğu dizin kabul eder burada bulamazsa `..` dizinine bakar.

`VPATH` değişkenini desteklemeyen bir **`make`** sürümü ile Bash'i kaynak dizini içinde bir kerede bir mimari için derleyebilirsiniz. Bir mimari için Bash'i kurduktan sonra başka bir mimari için paketi yapılandırmadan önce bir **`make distclean`** yapmalısınız.

Alternatif olarak, sisteminiz sembolik bağları destekliyorsa, kaynak dizininden farklı bir yerde her mimari için bir dizin oluşturup **`support/mkclone`** betiği ile bu dizin altında kaynak dizindeki her dosya ve dizin için sembolik bağlar oluşturabilirsiniz. Aşağıdaki örnek kaynak dizin `/usr/gnu/src/bash-2.0` dan bulunduğunuz dizine sembolik bağları oluşturur:

```
bash /usr/gnu/src/bash-2.0/support/mkclone -s /usr/gnu/src/bash-2.0
```

`mkclone` betiği Bash ile çalıştırılabilir. Bu nedenle çok mimarili bir derlemede her mimariye ayrı bir derleme dizini oluşturabilmek için en azından bir mimari için Bash kurmuş olmanız gerekir.

4. Kurulum İsimleri

Öntanımlı olarak, **`make install`** kurulumu `/usr/local/bin`, `/usr/local/man`, vs. dizinlerine yapar. Kurulumun yapılacağı temel dizini (burada `/usr/local`) **`configure`** betiğine `--prefix=`*`DosyaYolu`* seçeneği ile kurulum dizini öneki olarak belirtebileceğiniz gibi `DESTDIR` **`make`** değişkenine bu değeri **`make install`** komutunu çalıştırmadan önce atayarak belirtebilirsiniz.

Mimariye özel ve mimari bağımlı dosyaları ayrı kurulum önekleriyle kurmak için **`configure`** betiğini `--exec-prefix=`*`DosyaYolu`* seçeneği ile çalıştırabilirsiniz. Bu yöntemle sadece çalıştırılabilirler ve kütüphane dosyaları bu önekle kurulacaktır. Belgeler ve diğer veri dosyaları normal dizin önekini kullanacaktır.

5. Sistem Türünün Belirtilmesi

Bazı **configure** özellikleri otomatik olarak halledilemeyebilir. Bash'in üzerinde çalışacağı konak türünün saptanması gerekir. Genel olarak, **configure** bunu halleder, ancak bazan bir ileti ile konak türünü saptayamadığını belirtebilir. Bu durumda `--host=`*KonakTürü* seçeneği kullanılır. *KonakTürü* ya `sun4` gibi bir kısa isim veya `İşlemci-Firma-Sistem` biçeminde üç alanlı bir kurallı isim olabilir (örn. `i386-unknown-freebsd4.2`).

Her alana girilebilecek mümkün değerler için `support/config.sub` dosyasına bakınız.

6. Öntanımlıların Paylaşımı

configure betiğinin öntanımlı değerlerini her paket kurulumu için aynı olacak şekilde paylaşılmasını isterseniz **config.site** diye bilinen bir site kabuk betiği oluşturmalısınız. Bu dosya içinde `CC`, `cache_file` ve `prefix` gibi **configure** betiğinin kullandığı değişkenlere değer atayabilirsiniz. **configure** betiği bu dosyayı sırayla `PREFIX/share/config.site`, `PREFIX/etc/config.site` dizinlerinde ve `CONFIG_SITE` ortam değişkeninde arar.

Uyarı

Bash **configure** betiği bir site betiğini arar ama **configure** betiklerinin hepsi bunu yapmaz.

7. İşlem Denetimi

configure betiğinin işlemlerini kontrol eden seçenekler:

`--cache-file=`*dosya*
: Yapılandırma sırasında yapılan sınamalar `./config.cache` dosyası yerine *dosya*'ya kaydedilir ve buradan kullanılır. Hata ayıklaması yapmak için `/dev/null` vererek kaydı engelleyebilirsiniz.

`--help`
: **configure** seçeneklerinin bir özetini basar ve çıkar.

`--quiet`
`--silent`
`-q`
: Denetimler sırasında neler yapıldığını gösteren iletileri basmaz.

`--srcdir=`*dizin*
: Bash kaynak kodu *dizin* dizininde aranır. Genelde **configure** bunu otomatik olarak saptayabilir.

`--version`
: **configure** betiğinin üretiminde kullanılan Autoconf'un sürümünü gösterir ve çıkar.

configure ayrıca geniş olarak kullanılan bazı standart hale gelmiş seçeneklere de sahiptir. **configure**`--help` komutu ile bu seçenekleri listeleyebilirsiniz.

8. İsteğe Bağlı Özellikler

Bash **configure** betiği, Bash'in isteğe bağlı *özellik*'lerinin belirtilebildiği `--enable-`*özellik* seçeneklerine sahiptir. Bundan başka `bash-malloc` veya `purify` gibi bazı *paket*'lerin belirtilebildiği `--with-`*paket* seçenekleri, öntanımlı kullanılan paketlerin kullanılmamasını sağlayan `--without-`*paket* seçenekleri, öntanımlı olarak var olan bazı *özellik*'lerinin kaldırılmasını sağlayan `--disable-`*özellik* seçenekleri vardır.

Aşağıda `--enable-` ve `--with-` ile başlayan Bash seçeneklerinin bir listesini bulacaksınız:

`--with-afs`
Transarc'ın Andrew Dosya Sistemini (Andrew File System) kullanıyorsanız atayın.

`--with-bash-malloc`
`lib/malloc` dizinindeki **malloc**'un Bash sürümü kullanılır. Bu GNU libc **malloc** ile aynı değildir. Daha eski olan 4.2 BSD **malloc**'dan türetilmiş bir sürümdür. Bu **malloc** çok hızlı ama her bellek ayırmada belleğin birazını işe yaramaz hale getiren bir sürümdür. Bu seçenek öntanımlı olarak etkindir. `NOTES` dosyasında bu seçeneğin kapatılması gereken sistemler listelenmiştir ve **configure** bu sistemler üzerinde otomatik olarak bu seçeneği iptal eder.

`--with-curses`
termcap kütüphanesi yerine **curses** kütüphanesi kullanılır. Eğer sisteminizdeki **termcap** veritabanı yetersiz ya da yoksa bu seçenek kullanılmalıdır.

`--with-gnu-malloc`
`--with-bash-malloc` ile eşanlamlıdır.

`--with-installed-readline[=`*ÖNEK*`]`
Bu seçeneği `lib/readline`'daki sürümü yerine Readline'ın makinanızda bulunan sürümü ile Bash'i ilintilemek için kullanın. Bu seçenek sadece Readline 5.0 ve sonraki sürümleri ile çalışır. *ÖNEK* `yes` olarak verilmişse ya da hiç verilmemişse ve Readline, sistemin standart `include` ve `lib` dizinlerinde bulunmuyorsa **configure** betiği **make**'in `prefix` değişkeninde tanımlı dizinin altdizinleri olarak atanmış `includedir` ve `libdir` değişkenlerinin değerlerini kullanır. *ÖNEK* `no` ise, Bash `lib/readline`'daki sürüm ile ilintilenir. *ÖNEK* için başka bir değer verilmişse, **configure** bu değeri bir dizin yolu olarak ele alır ve Readline'ın kurulu sürümünü bu dizinin alt dizinlerinde arar (başlık dosyaları için *ÖNEK*`/include` ve kütüphane dosyaları için *ÖNEK*`/lib` dizinleri).

`--with-purify`
Rational Software'in Purify bellek ayırma denetleyicisini kullanmak içindir.

`--enable-minimal-config`
Tarihsel Bourne kabuğuna çok yakın, en az özellik içeren bir kabuk üretir.

`--enable-` ile başlayan seçenekler, Bash'in çalışma anında özelliklerini değiştirmek yerine derleme ve ilintileme ile kurulurken uygun özelliklerde olmasını sağlamak içindir.

`--enable-largefile`
İşletim sistemi büyük dosyalara erişen programların derlenebilmesi için özel derleyici seçenekleri gerektiriyorsa, büyük dosya[B322] desteğini etkinleştirir. İşletim sistemi büyük dosya desteğine sahipse bu özellik öntanımlı olarak etkindir.

`--enable-profiling`
gprof tarafından işlenen profil bilgilerini üreten bir Bash çalıştırılabilirinin derlenmesini sağlar.

`--enable-static-link`
gcc kullanılıyorsa, Bash'in statik olarak ilintilenmesini sağlar. Bu seçenek `root` kullanıcısının kabuğu olarak kullanılacak bir sürüm derlemek için kullanılmalıdır.

`minimal-config` seçeneği aşağıdaki seçeneklerin tamamını iptal etmek için kullanılabilir. Ancak `minimal-config` seçeneği ilk seçenek olmalıdır. Bundan sonra aşağıdaki seçeneklerden istenenler `enable-özellik` seçeneği ile tek tek belirtilebilir.

İşletim sisteminin gerekli desteği sağlamaması dışında, `disabled-builtins` ve `xpg-echo-default` seçenekleri hariç aşağıdaki seçeneklerin tümü öntanımlı olarak etkinleştirilir.

`--enable-alias`
: **alias** ve **unalias** yerleşikleri dahil *takma ad yorumlamasına* (sayfa: 75) izin verilir.

`--enable-arith-for-command`
: C dilinin `for` deyimi gibi davranan başka bir **for** *komutu* (sayfa: 17) için destek içerir.

`--enable-array-variables`
: Tek boyutlu *dizi* (sayfa: 76) kabuk değişkenleri için destek içerir.

`--enable-bang-history`
: **csh** tarzı *geçmiş yorumlaması* (sayfa: 113) için destek içerir.

`--enable-brace-expansion`
: **csh** tarzı *kaşlı parantez yorumlaması* (sayfa: 23) için destek içerir. (`b{a,b}c ==> bac bbc`).

`--enable-command-timing`
: **time** sözcüğünün anahtar sözcük olması ve **time** ile başlayan *boruhatları* (sayfa: 15) için zamanlama istatistiklerinin gösterilmesi desteğini içerir.

`--enable-cond-command`
: **[[** *koşullu komutu* (sayfa: 18) için destek içerir.

`--enable-cond-regexp`
: **=~** iki terimlisinin **[[** *koşullu komutunda* (sayfa: 18) kullanımıyla POSIX düzenli ifadelerini kullanarak eşleşme bulmaya destek verir.

`--enable-debugger`
: Bash hata ayıklayıcı (ayrı dağıtılır) için destek içerir.

`--enable-directory-stack`
: **csh** tarzı *dizin yığını* (sayfa: 77) ile **pushd** (sayfa: 78), **popd** (sayfa: 77), ve **dirs** (sayfa: 77) yerleşikleri için destek içerir.

`--enable-disabled-builtins`
: Bir `xxx` yerleşiği ***enable* (sayfa: 47) -n xxx** komutu ile iptal edilmiş bile olsa ***builtin* (sayfa: 44) xxx** yerleşiği ile çalıştırılabilmesine izin verilir.

`--enable-dparen-arithmetic`
: **((...))** (sayfa: 18) komutu için destek içerir.

`--enable-extended-glob`
: *Kalıp Eşleme* (sayfa: 28) özellikleri için destek içerir.

`--enable-help-builtin`
: Kabuk yerleşikleri ve değişkenleri hakkında yardım almak için kullanılan **help** (sayfa: 47) yerleşiği için destek içerir.

`--enable-history`
: **history** (sayfa: 112) ve **fc** (sayfa: 111) yerleşikleri için destek içerir.

`--enable-job-control`
: İşletim sistem destekliyorsa *İş Denetimi* (sayfa: 83) özelliklerini etkinleştirir.

`--enable-multibyte`
: İşletim sistem destekliyorsa çokbaytlı karakterlere desteği etkinleştirir.

`--enable-net-redirections`

Yönlendirmelerde (sayfa: 29) kullanıldığında `/dev/tcp/konak/port` ve `/dev/udp/konak/port` biçemindeki dosyalara erişimi etkinleştirir.

`--enable-process-substitution`
İşletim sistemi gerekli desteği sağlıyorsa *süreç ikamesini* (sayfa: 27) etkinleştirir.

`--enable-progcomp`
Programlanabilir tamamlama (sayfa: 105) oluşumlarını etkinleştirir. Readline etkin değilse bu seçenek etkisizdir.

`--enable-prompt-string-decoding`
`$PS1`, `$PS2`, `$PS3` ve `$PS4` komut istemi dizgelerinde tersbölü öncelemeli karakterlerin yorumlanmasını etkinleştirir. Daha ayrıntılı bilgi için *Komut İsteminin Kontrol Edilmesi* (sayfa: 78) bölümüne bakınız.

`--enable-readline`
Readline kütüphanesinin Bash sürümü ile *Komut Satırının Düzenlenmesi* (sayfa: 87) ve *geçmiş* (sayfa: 111) desteğini etkinleştirir.

`--enable-restricted`
Sınırlı Kabuk (sayfa: 80) için destek içerir. Bu seçenek etkin olduğunda, Bash **`rbash`** olarak çağrıldığında sınırlı kipe girer.

`--enable-select`
Basit menülerin oluşturulmasına imkan veren **`select`** (sayfa: 18) komutuna destek içerir.

`--enable-separate-helpfiles`
`help` yerleşiği tarafından gösterilen belgeleme için metni dahili olarak saklamak yerine harici dosyalar kullanılır.

`--enable-single-help-strings`
`help` yerleşiği tarafından gösterilen metin her yardım konusu için ayrı bir dizge olarak saklanır. Bu metnin farklı dillere tercümesini mümkün kılar. Eğer derleyiciniz çok uzun dizge sabitlerle çalışamıyorsa bu seçeneği etkinleştirmemelisiniz.

`--enable-strict-posix-default`
Bazh'i öntanımlı olarak POSIX uyumlu yapar (bkz, *Bash POSIX Kipi* (sayfa: 80)).

`--enable-usg-echo-default`
`--enable-xpg-echo-default` ile eşanlamlıdır.

`--enable-xpg-echo-default`
`echo` (sayfa: 46) yerleşiğinin `-e` seçeneğini gerektirmeksizin tersbölü öncelemeli karakterleri yorumlamasını öntanımlı olarak etkinleştirir. Bu seçenek **`echo`** yerleşiğinin Tek Unix Belirtimi, sürüm 3'te belirtilen sürüme daha benzer davranmasını sağlayan `xpg_echo` (sayfa: 56) kabuk seçeneğini öntanımlı etkin yapar.

`config-top.h` dosyası **`configure`** tarafından belirtilemeyen seçenekler için C önişlemcisinin `#define` deyimlerini içerir. Bunların bazıları değiştirilebilir anlamında almayın, yaparsanız, sonuçlarına katlanırsınız. Etkisini anlamak için her tanım ile ilişkili açıklamaları okuyun.

A. Hataları Raporlama

Bash'de bulduğunuz hataların tümünü lütfen raporlayın. Ama önce, onun gerçekten hata olup olmadığından ve Bash'in en son sürümününde de olduğundan emin olun. Bash'in en son sürümünü daima `ftp://ftp.gnu.org/pub/bash/` adresinden temin edebilirsiniz.

Gerçekten bir hata saptarsanız, hatayı raporlamak için **bashbug** komutunu kullanın. Kaynak kodunda bir düzeltme yaptıysanız (fix) cesur olun ve onu bize postalayın. Tavsiyeler ve 'felsefi' hata raporları <bug–bash (at) gnu.org> adresine veya Usenet `news:gnu.bash.bug` haber öbeğine postalayabilirsiniz.

Tüm hata raporlarında bunların bulunmasını sağlayın:

- Bash'in sürüm numarası.
- Donanım ve işletim sistemi.
- Bash'i derlemekte kullanılan derleyici.
- Hatanın neler yaptığına dair açıklama.
- Hatayı oluşturan bir kısa betik veya bir 'reçete'.

bashbug ilk üç öğeyi otomatik olarak hata raporlamak için kullanacağınız şablona yerleştirir.

Lütfen bu kılavuzun orjinali ile ilgili tüm raporları <chet (at) po.CWRU.Edu> adresine ve çeviri ile ilgili hataları <nilgun (at) belgeler·gen·tr> adresine bildirin.

B. Bash ile Bourne Kabuğu Arasındaki Başlıca Farklar

Bash, Bourne Kabuğu ile esasen aynı dil bilgisi, parametre ve değişken yorumlaması, yönlendirme ve tırnak içine alma özelliklerin sahiptir. Bash bu özelliklerin belirtimleri olarak POSIX 1003.2 standardını kullanır. Geleneksel Bourne kabuğu ile Bash arasında bazı farklar vardır; bu bölümde önemli farklara kısaca değinilecektir. Bu farkların bir kısmı önceki bölümlerde derinliğine açıklanmıştır. Bu bölümde SVR4.2 (tarihi Bourne kabuğunun son sürümü) içindeki **sh** sürümü ile karşılaştırma yapılmıştır.

- POSIX belirtiminin geleneksel **sh** davranışından farkları olsa da, Bash POSIX uyumludur (*Bash POSIX Kipi* (sayfa: 80) bölümüne bakınız).
- Bash çok karakterli çağrı seçeneklerine sahiptir (*Bash'in Çağrılması* (sayfa: 67) bölümüne bakınız).
- Bash'de komut satırı düzenlemesi ve **bind** yerleşiği vardır (*Komut Satırının Düzenlenmesi* (sayfa: 87) bölümüne bakınız).
- Bash bir *programlanabilir sözcük tamamlama* (sayfa: 105) mekanizması ile onu yönetmek için **complete** ve **compgen** yerleşiklerini içerir.
- Bash *komut geçmişi* (sayfa: 111) ile onu yönetmek için **history** ve **fc** yerleşiklerine sahiptir. Bash geçmiş listesi zaman damgası bilgisi sağlar ve değerini göstermek için `HISTTIMEFORMAT` değişkenini kullanır.
- Bash **csh** tarzı *geçmiş yorumlamasına* (sayfa: 113) sahiptir.
- Bash tek boyutlu *dizi* (sayfa: 76) değişkenlerine ve onları kullanacak uygun değişken atama sözdizimi ve değişken yorumlamasına sahiptir. Bash'in bazı yerleşiklerinin seçenekleri dizilerle çalışmak içindir. Ayrıca Bash'in yerleşik dizi değişkenleri de vardır.
- Bash'de tek tırnak içine alınmış metinlerde *ANSI–C* (sayfa: 13) tersbölü öncelemeli karakterleri yorumlayan `$'...'` tırnaklama sözdizimi desteği vardır.
- Bash çift tırnak içine alınmış karakterlere *yerele özel çeviri* (sayfa: 14) yapmak için `$"..."` tırnaklı sözdizimini destekler. `-D`, `--dump-strings`, ve `--dump-po-strings` komut satırı seçenekleri bir betik içindeki çevrilebilir dizgeleri listeler.

- Bash bir *boruhattının* (sayfa: 15) dönüş değerinin tersini alabilen bir `!` anahtar sözcüğüne sahiptir. Bir sınama başarısız olduğunda bir **`if`** deyiminde işlem yaptırmak için faydalıdır. Bash, **`set`** yerleşiğinin `-o pipefail` seçeneği ile etkinleştirilen ve içindeki bir komutunun başarısızlığı halinde bir hata döndüren bir boruhattına sahiptir.
- Bash `time` anahtar sözcüğü ve *komut zamanlaması* (sayfa: 15) içerir. `TIMEFORMAT` değişkeni ile kontrol edilen zamanlama istatistikleri gösterir.
- Bash, C dilindekine benzer

  `for` `((` *ifade1* `;` *ifade2* `;` *ifade3* `))`

  aritmetik *for* (sayfa: 17) komutu içerir.
- Bash basit menülerin oluşturulmasına izin veren **`select`** (sayfa: 18) birleşik komutuna sahiptir.
- Bash kabuk sözdiziminde sınama yapmaya yarayan ve isteğe bağlı olarak düzenli ifade eşleşmesini de mümkün kılan **`[[`** *koşullu komutu* (sayfa: 18) için destek içerir.
- Bash **`case`** ve **`[[`** oluşumları için isteğe bağlı olarak harf büyüklüğüne duyarsız eşleşme sağlar.
- Bash *kaşlı ayraç yorumlaması* (sayfa: 23) ve *yaklaşık yorumlaması* (sayfa: 23) içerir.
- Bash *komut takma adları* (sayfa: 75) ile **`alias`** ve **`unalias`** yerleşiklerini içerir.
- Bash *kabuk aritmetiği* (sayfa: 74), **`((`** *birleşik komutu* (sayfa: 18) ve *aritmetik yorumlamasına* (sayfa: 74) sahiptir.
- Kabuğun ilk ortamında mevcut olan değişkenler ast süreçlere otomatik olarak aktarılır. Bourne kabuğu **`export`** komutu kullanılarak değişkenler imlenmedikçe normalde bunu yapmaz.
- Bash isimli değişkenin sol taraf değerine ekleme yapan **`+=`** atama işlecini destekler.
- Bash POSIX kalıp yoketme, değişken değerlerinden baştaki ve sondaki altdizgeleri kaldıracak `%`, `#`, `%%` ve `##` yorumlamaları içerir (*Kabuk Parametrelerinin Yorumlaması* (sayfa: 24) bölümüne bakınız).
- `${xx}`'in uzunluğunu döndüren `${#xx}` yorumlaması desteklenir (*Kabuk Parametrelerinin Yorumlaması* (sayfa: 24) bölümüne bakınız).
- *konum* dan başlayan *uzunluk* karakterlik *parametre* alt dizgesini yorumlayan `${parametre:konum[:uzunluk]}` *parametre yorumlaması* (sayfa: 24) vardır.
- *kalıp*'ı eşleştirdikten sonra onu `parametre` nin değerindeki *dizge* ile değiştiren `${parametre/[/]kalıp[/dizge]}`, *parametre yorumlaması* (sayfa: 24) vardır.
- İsimleri *önek* ile başlayan tüm kabuk değişkenlerinin isimleri olarak yorumlanan `${!önek}*` *parametre yorumlaması* (sayfa: 24) vardır.
- Bash `${!sözcük}` kullanarak *dolaylı değişken yorumlamasına* (sayfa: 24) sahiptir.
- Bash `$9` dan büyük konumsal paramaetreleri `${sayı}` kullanarak yorumlar.
- POSIX `$()` biçemli *komut ikamesi* (sayfa: 26), Bourne kabuğunun `` ` `` komut ikamesine tercih edilmiştir. Ancak Bourne kabuğunun komut ikamesi de desteklenmektedir.
- Bash *süreç ikamesine* (sayfa: 27) sahiptir.
- Bash o anki kullanıcı (`UID`, `EUID` ve `GROUPS`), o anki konak (`HOSTTYPE`, `OSTYPE`, `MACHTYPE` ve `HOSTNAME`) ve çalışan Bash kopyası (`BASH`, `BASH_VERSION` ve `BASH_VERSINFO`) hakkında bilgi sağlayan *değişkenleri* (sayfa: 60) otomatik olarak atar.
- `IFS` değişkeni tüm sözcüklerin değil sadece *yorumlama sonuçlarını ayrımlamakta* (sayfa: 27) kullanılmıştır. Bu uzun zamandır süren bir güvenlik açığını kapatmıştır.
- Bash POSIX 1003.2 *dosyaismi yorumlama* (sayfa: 27) işleçlerinin tamamını destekler (karakter sınıfları, eşitlik sınıfları ve birleştirme sembolleri dahil).
- Bash `extglob` kabuk seçeneği etkinken ek kalıp eşleşme işleçlerini kullanabilir (*Kalıp Eşleme* (sayfa: 28) bölümüne bakınız).

- Bir değişken ile bir işlev aynı isimde olabilir; **sh** bu iki isim alanını ayırmaz.
- Bash işlevleri **local** (sayfa: 47) yerleşiğini kullanarak yerel değişkenler kullanabilir. Böylece iç içe işlev çağrıları yazılabilmesi mümkündür.
- Komutları önceleyen değişken atamaları sadece komutları, yerleşikleri ve işlevleri etkiler. **sh**'da komutları önceleyen tüm değişken atamaları komut dosya sisteminden çalıştırılmadıkça çağrıldığı ortamı da etkiler (*Ortam* (sayfa: 34) bölümüne bakınız).
- Bash girdi ve çıktı *yönlendirme* (sayfa: 29) işleçlerine terim olarak belirtilen dosyaisimleri üstünde dosyaismi yorumlaması uygular.
- Bash bir dosyayı hem okumak hem de yazmak için açan `<>` ve standart çıktı ile standart girdiyi aynı dosyaya yönlendiren *yönlendirme* (sayfa: 29) işleçlerini içerir.
- Bash bir dizgenin bir komutun standart girdisi olarak kullanılmasını mümkün kılan `<<<` yönlendirme işlecini içerir.
- Bash bir dosya tanıtıcısını bir diğerine taşıyan `[`*n*`]<&`*sözcük* ve `[`*n*`]>&`*sözcük* yönlendirme işleçlerini içerir.
- Bash *yönlendirme* (sayfa: 29) işleçlerinde kullanıldığında dosyaisimlerini özel olarak ele alır.
- Bash *yönlendirme* (sayfa: 29) işleçleri ile isteğe bağlı olarak makinalara ve servislere ağ bağlantıları açabilir.
- Bash'de çıktı yönlendirmesi ile mevcut dosyaların üzerine yazılmasını engelleyecek `noclobber` (sayfa: 50) seçeneği vardır. `>|` yönlendirme işleci ile `noclobber` seçeneğinin engeli aşılabilir.
- Bash **cd** (sayfa: 38) ve **pwd** (sayfa: 40) yerleşiklerinin herbiri `-L` ve `-P` seçenekleri ile mantıksal ve fiziksel kiplere geçebilir.
- Bash bir yerleşikle aynı isimde işlevlere izin vererek işlev içinden yerleşiğin işlevselliğine **builtin** (sayfa: 45) ve **command** (sayfa: 45) yerleşikleri üzerinden erişilmesini sağlar.
- **command** (sayfa: 45) yerleşiği komut araması uygulanırken işlevlerin seçimli iptaline izin verir.
- Yerleşikler **enable** (sayfa: 47) yerleşiği kullanılarak tek tek etkinleştirilebilir ya da iptal edilebilir.
- Bash **exec** (sayfa: 39) yerleşiği, çalıştırılan komuta aktarılan ortamın içeriğinin ve komuta verilecek sıfırıncı argümanın ne olacağının kullanıcı tarafından kontrol edilmesini sağlayan ek seçeneklere sahiptir.
- *Kabuk işlevleri* (sayfa: 20) **export** `-f` ile ortam üzerinden alt süreçlere aktarılabilir.
- Bash **export** (sayfa: 39), **readonly** (sayfa: 40) ve **declare** (sayfa: 45) yerleşikleri kabuk işlevleri ile çalışan bir `-f` seçeneğine, bir kabuk girdisi olarak kullanılabilen bir biçemde atanan çeşitli isimlere sahip değişkenleri gösteren bir `-p` seçeneğine, çeşitli değişken isimlerini kaldıran bir `-n` seçeneğine ve değişken isimleri ile değerlerini aynı anda atayan `isim=değer` argümanlarına sahiptir.
- Bash **hash** (sayfa: 40) yerleşiği dosyaismi `$PATH` aramasında **hash** `-p` kullanılarak bulunamasa bile bir ismin bir keyfi dosyaismi ile ilişkilendirilmesine izin verir.
- Bash kabuk yerleşikleri ve değişkenleri hakkında yardım almak için kullanılan bir **help** (sayfa: 47) yerleşiğine sahiptir.
- **printf** (sayfa: 48) yerleşiği biçemli çıktıları göstermekte kullanılır.
- Bash **read** (sayfa: 48) yerleşiği `\` ile biten bir satırı `-r` seçeneği ile okuyacak ve hiç seçenek olmayan argüman verilmezse öntanımlı olarak `REPLY` değişkenini kullanacaktır. Bash **read** yerleşiği ayrıca `-p` seçeneği ile bir komut istemi dizgesi kabul eder ve `-e` seçeneği verildiğinde satırı sağlamada Readline'ı kullanır. **read** yerleşiği bunlardan başka girdiyi kontrol etmek için ek seçeneklere sahiptir: `-s` seçeneği girdi karakterlerinin okunduğu gibi yansılanmasını engelleyecek, `-t` seçeneği ile belirtilen süre içinde bir girdi alınmazsa okumanın zamanaşımına girmesi sağlanacak, `-n` seçeneği satırın tamamı yerine sadece belirtilen sayıda karakterin okunmasına izin verecek ve `-d` seçeneği satırsonu yerine bir başka karaktere kadar okuyacaktır.

- **return** (sayfa: 41) yerleşiği **.** veya **source** (sayfa: 56) yerleşikleri ile çalıştırılan betiklerin çıkmasını sağlamakta kullanılabilir.
- Bash kabuğun istemlik yeteneklerini daha iyi kontrol etmek için ve bu seçeneklerin *kabuk çağrısında* (sayfa: 67) atanmasını ve kaldırılmasını sağlayan **shopt** (sayfa: 52) yerleşiğine sahiptir.
- Bash **set** (sayfa: 49) yerleşiği ile kontrol edilebilir çok sayıda istemlik davranışa sahiptir.
- `-x` (`xtrace` (sayfa: 51)) seçeneği bir çalışmanın izlenmesi sırasında basit komutlardan başka komutlar da gösterir.
- **test** (sayfa: 41) yerleşiği argümanlarıyla davranışı değiştirilebilen bir POSIX algoritması olarak biraz farklı gerçeklenmiştir.
- Bash, etkin altyordam çağrısının (**.** veya **source** yerleşiği ile çalıştırılan bir kabuk işlevi ya da betiği) bağlamını gösteren **caller** yerleşiğini içerir. Bu, bash hata ayıklayıcısını da destekler.
- **trap** (sayfa: 42) yerleşiği `EXIT`'e benzer bir `DEBUG` sinyalimsi belirtimine izin verir. Bir `DEBUG` kapanı ile belirtilen komutlar her basit komuttan, **for** komutundan, **case** komutundan, **select** komutundan, her aritmetik **for** komutundan ve bir kabuk işlevinde çalıştırılacak ilk komuttan önce çalıştırılır. İşleve `trace` niteliği verilmiş olmadıkça ya da **shopt** yerleşiği kullanılarak `functrace` seçeneği etkinleştirilmiş olmadıkça `DEBUG` kapanı kabuk işlevleri tarafından miras alınmaz. `extdebug` kabuk seçeneğinin `DEBUG` kapanı üzerinde ek bir etkisi vardır.
- **trap** (sayfa: 42) yerleşiği `EXIT` ve `DEBUG`'a benzer bir `ERR` sinyalimsi belirtimine izin verir. Bir `ERR` kapanı ile belirtilen komutlar bir kaç durum dışında her başarısız basit komuttan sonra çalıştırılır. **set** yerleşinin `-o errtrace` seçeneği etkinleştirilmedikçe `ERR` kapanı kabuk işlevleri tarafından miras alınmaz.

  trap (sayfa: 42) yerleşiği `EXIT` ve `DEBUG`'a benzer bir `RETURN` sinyalimsi belirtimine izin verir. Bir `RETURN` kapanı ile belirtilen komutlar **.** veya **source** ile çalıştırılan bir kabuk işlevi veya betiği döndükten sonra kaldığı yerden devam etmeden önce çalıştırılırlar. İşleve `trace` niteliği verilmiş olmadıkça ya da **shopt** yerleşiği kullanılarak `functrace` seçeneği etkinleştirilmiş olmadıkça `RETURN` kapanı kabuk işlevleri tarafından miras alınmaz.

- Bash **type** (sayfa: 56) yerleşiği daha yaygındır ve bulduğu isimler hakkında daha fazla bilgi verir.
- Bash **umask** (sayfa: 43) yerleşiği kendisine girdi olarak verilebilecek biçemde gösterilen bir çıktı veren bir `-p` seçeneğine sahiptir.
- Bash bir **csh** benzeri *dizin yığınına* (sayfa: 77) sahiptir ve onu yönetmek için **pushd**, **popd** ve **dirs** yerleşiklerini içerir. Bash ayrıca dizin yığınını `DIRSTACK` (sayfa: 61) kabuk değişkeninin değerindeki gibi görünür yapabilir.
- Bash etkileşimliyken *komut isteminde* (sayfa: 78) özel tersbölü öncelemeli karakterleri yorumlar.
- Bash *sınırlı kipte* (sayfa: 80) çok kullanışlıdır; SVR4.2 kabuğu ise sınırlı kipte çok fazla sınırlıdır.
- **disown** (sayfa: 85) yerleşiği bir işi kabuğun dahili iş tablosundan kaldırabilir veya kabuk bir `SIGHUP` sinyalinin sonucu olarak çıkarken bir işe `SIGHUP` sinyalinin gönderilmesini engelleyebilir.
- Bash, kabuk betikleri için ayrı bir hata ayıklayıcıyı destekleyecek bir miktar özellik içerir.
- The SVR4.2 kabuğu ayrıcalıklarla ilgili `mldmode` ve `priv` diye Bash'de bulunmayan iki yerleşiğe sahiptir.
- Bash `stop` ve `newgrp` yerleşiklerine sahip değildir.
- Bash kabuk hesapları uygulamaz ve `SHACCT` değişkenini kullanmaz.
- SVR4.2 **sh** `TIMEOUT` değişkeni kullanırken Bash `TMOUT` değişkenini kullanır.

Bash'e özel diğer özellikler *Bash'in Özellikleri* (sayfa: 67) bölümünde bulunabilir.

B.1. SVR4.2 Kabuğunun Oluşumsal Farkları

Bash tamamen yeni bir oluşum olduğundan, SVR4.2 kabuğunun bir çok sınırlamasından muaftır. Örneğin:

- Bash bir **if** veya **while** deyiminde olduğu gibi bir kabuk denetim yapısının içine ya da dışına yapılan yönlendirmelerde bir altkabuk çatallamaz.
- Bash kapanmamış tırnaklara icin vermez. SVR4.2 kabuğu aynı durumda hata vermeyerek kapatan tırnağı `EOF` konumunda yerleştirecektir. Bu da bulunması çok zor hatalara sebep olur.
- SVR4.2 kabuğu `SIGSEGV` sinyalinin yakalanmasına bağlı bir barok bellek yönetim şeması kullanır. Kabuk `SIGSEGV` ile bloklanmış (örneğin, `system()` C kütüphane işlevi çağrısı ile) bir süreçten başlatılırsa, oldukça yanlış davranır.
- Güvenlikte şüpheli bir teşebbüs halinde SVR4.2 kabuğu `-p` seçeneği olmaksızın çağrıldığında gerçek ve etkin UID ve GID'i bir sihirli eşik değerden (genelde 100) küçükse, gerçek ve etkin UID ve GID'ini değiştirecektir. Bu beklenmeyen sonuçlara yol açabilir.
- SVR4.2 kabuğu kullanıcıların `SIGSEGV`, `SIGALRM` ve `SIGCHLD` sinyalleri ile ilgili sinyal kapanı kurmalarına izin vermez.
- SVR4.2 kabuğu `IFS`, `MAILCHECK`, `PATH`, `PS1` ve `PS2` değişkenlerinin ortamdan kaldırılmasına izin vermez.
- SVR4.2 kabuğunda `^`, `|`'ya eşdeğer olarak ele alınır.
- Bash seçeneklere çoklu argümanlar olarak (`-x -v`) izin verirken SVR4.2 kabuğu sadece tek argümanlık (`-xv`) seçenek belirtilmesine izin verir. Gerçekte, kabuğun bazı sürümleri ikinci argüman bir `-` ile başlıyorsa bellek dökümü ile çıkar (core dump).
- SVR4.2 kabuğunda bir betiğin içindeki bir yerleşik komut hata ile çıkarsa betikte çıkar. Bash'de ise sadece POSIX 1003.2 özel yerleşiklerinden biri başarısız olursa ve sadece POSIX 1003.2 standardında sıralanmış başarısızlık hallerinde betik çıkar.
- SVR4.2 kabuğu **jsh** olarak çağrıldığında farklı davranır (iş denetimini etkinleştirir).

C. Bu Kılavuzun Kopyalanması

GNU Özgür Belgeleme Lisansı ile lisanslanmış belgelerin bu lisansı içermesi gerektiğinden ve bu lisans kendisinin değiştirilmesine izin vermediğinden (buna tercüme de dahildir) lisans hiçbir değişiklik yapılmaksızın burada belgeye eklenmiştir.

(Ç.N. – GNU Özgür Belgeleme Lisansı bu özelliği sebebiyle dili İngilizce olmayan belgelerde kullanmak için uygun değildir; Türkçe belgenize İngilizce bir metin eklemek istemezsiniz, herhalde. Daha özgür –kendinin belgeye eklenmesini zorunlu kılmayan– lisanslar da var. Örneğin "Creative Commons Share Alike" kendinin belgeye eklenmesini zorunlu kılmaması dışında GNU ÖBL'ye hemen hemen eşdeğerdir.)

GNU Free Documentation License

Version 1.2, November 2002

```
Copyright © 2000,2001,2002 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA  02111-1307, USA

Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.
```

1. PREAMBLE

   The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

2. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world–wide, royalty–free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front–matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front–Cover Texts or Back–Cover Texts, in the notice that says that the Document is released under this License. A Front–Cover Text may be at most 5 words, and a Back–Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine–readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ascii without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard–conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally

available, and the machine–generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

3. VERBATIM COPYING

   You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

   You may also lend copies, under the same conditions stated above, and you may publicly display copies.

4. COPYING IN QUANTITY

   If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front–Cover Texts on the front cover, and Back–Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

   If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

   If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine–readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer–network location from which the general network–using public has access to download using public–standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

5. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front–matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front–Cover Text, and a passage of up to 25 words as a Back–Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front–Cover Text and one of Back–Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

6. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

7. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

8. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

9. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

10. TERMINATION

    You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

11. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
  Copyright (C)  year  your name.
  Permission is granted to copy, distribute and/or modify this document
```

```
  under the terms of the GNU Free Documentation License, Version 1.2
  or any later version published by the Free Software Foundation;
  with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.
  A copy of the license is included in the section entitled "GNU
  Free Documentation License".
```

If you have Invariant Sections, Front–Cover Texts and Back–Cover Texts, replace the "with...Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Kabuk Yerleşik Komutları Dizini

Semboller

A

B

C

D

E

F

G

H

J

K

L

P

R

S

T

U

W

Kabuk Seçenekleri Dizini

Kabuk Anahtar Sözcükleri Dizini

Semboller

C

D

E

F

İ

S

T

U

W

Parametreler ve Değişkenler Dizini

Semboller

A

B

C

D

E

F

G

H

I

İ

K

L

İşlev Dizini

P

Q

R

S

T

U

Y

Kavramlar Dizini

Notlar

a) Belge içinde dipnotlar ve dış bağlantılar varsa, bunlarla ilgili bilgiler bulundukları sayfanın sonunda dipnot olarak verilmeyip, hepsi toplu olarak burada listelenmiş olacaktır.

b) Konsol görüntüsünü temsil eden sarı zeminli alanlarda metin genişliğine sığmayan satırların sığmayan kısmı ¬ karakteri kullanılarak bir alt satıra indirilmiştir. Sarı zeminli alanlarda ¬ karakteri ile başlayan satırlar bir önceki satırın devamı olarak ele alınmalıdır.

(1) Bir betik çalışırken klavyeden ^C tuşladığınızda durmamasını isterseniz aşağıdaki gibi bir programcığı betiğin başına yerleştirebilirsiniz (fare kapanı kurmak gibi sinyal kapanı kuruyoruz):

```
# Ctrl + C tuşlandığında gösterilecek ileti
trap 'echo "Control-C iptal edildi."' SIGINT
```

(2)

```
# Hiçbir ileti göstermeden de ^C iptal edilebilir
trap '' SIGINT
```

(3) ^C kesmesini tekrar etkin kılmak isterseniz:

```
trap SIGINT
```

yeterlidir.

(4) Ç.N.: Tohum (seed) yeni rasgele sayıların üretilmesi için kaynaklık edecek bir sayı olarak düşünülebilir. Aynı tohumlar hep aynı sayıları üretir.

(B322) http://www.sas.com/standards/large_file/x_open.20Mar96.html

Bu dosya (bashref.pdf), belgenin XML biçiminin TEXLive ve belgeler-xsl paketlerindeki araçlar kullanılarak PDF biçimine dönüştürülmesiyle elde edilmiştir.

27 Şubat 2007