Zeki ÇIPLAK / 723624006

TC. Marmara Üniversitesi / Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği / Doktora Programı

18 Mart 2025

BASH KABUĞU

İÇİNDEKİLER

1. GİRİŞ

Bash (**B**ourne **A**gain **Sh**ell), Linux ve Unix tabanlı sistemlerde kullanılan bir komut satırı yorumlayıcısıdır. Kullanıcının işletim sistemiyle etkileşim kurmasını sağlar. Shell'in gelişmiş bir versiyonudur ve GNU tarafından geliştirilmiştir.

Bash, 1989 yılında **Brian Fox** tarafından Bourne Shell'in (sh) geliştirilmiş bir versiyonu olarak yazılmıştır. Zamanla birçok ek özellik kazanmış ve Linux'un varsayılan kabuğu haline gelmiştir. Bash, kullanım kolaylığı ve yaygınlığı nedeniyle en çok tercih edilen kabuktur. Linux kullanıcılarının sistem üzerinde tam kontrol sağlamasını kolaylaştıran güçlü bir araçtır.

Bash kabuğu, genellikle Linux ve macOS sistemlerinde varsayılan olarak bulunur. Windows'ta Bash kullanmak için **WSL** (**W**indows **S**ubsystem for **L**inux) veya Git Bash gibi araçlar gereklidir. Windows 10 ve 11 kullanıcıları genellikle WSL üzerinden Ubuntu veya başka bir Linux dağıtımını kurarak tam Bash deneyimi elde edebilir. Terminal, kullanıcı ile işletim sistemi arasındaki arayüzdür. Böylece Bash, terminal içinde çalışan bir kabuktur ve komutları yorumlar. İşletim sisteminin çekirdeğiyle doğrudan iletişim kurar.

2. TEMEL BASH KOMUTLARI

`echo`: Terminale yazı yazdırmak için çift tırnak ile kullanılır.

`clear`: Terminaldeki tüm yazıları temizler.

`ls`: Dosyaları listelemek için kullanılır.

- `ls` Bulunulan dizindeki dosyaları listeler.
- `ls -l` Bulunulan dizindeki dosyaları listeler ve dosya izinleri ile ilgili bilgiler verir.
- `ls -a` Bulunulan dizindeki dosyaları, üst ve alt dizinleri listeler.
- `ls dizin` Belirtilen dizindeki dosyaları listeler.

`cd`: Dizin değiştirmek için kullanılır.

- `cd` /home klasörüne (temel dizine) gider.
- `cd –` Bir önceki bulunulan dizine gider.
- `cd ..` Bir üst dizine gidebilmeyi sağlar.

`pwd`: Terminalde o an bulunduğumuz dizini gösterir.

`cat`: Dosya içeriğini okuma ve çeşitli dosya kaydetme işlemleri için kullanılır.

- `cat dosya` "dosya"'nın içeriğini okur ve o içeriği terminale döker.
- `cat > dosya` "dosya" isminde yeni bir dosya oluşturur ve terminalden içeriğini doldurmamızı bekler.
- `cat dosya1 dosya2 > dosya3` "dosya1" ve "dosya2"'yi okur, ikisinin içeriğini "dosya3"'e ekler.

`touch`: Yeni ve boş bir dosya oluşturur.

- `touch dosya` "dosya" isminde yeni ve boş bir dosya oluşturur.

cp: Kopyalama işlemi için kullanılır.

- **cp dosya1 dosya2** "dosya1" dosyasının içeriğini kopyalar ve "dosya2" adıyla kaydeder.

mv: Taşıma işlemi için kullanılır.

- **mv dosya dizin** "dosya" dosyasını, belirtilen "dizin"'e taşır.

mkdir: Klasör oluşturmak için kullanılır.

- **mkdir klasör** "klasör" isminde yeni bir klasör oluşturur.
- **mkdir -p klasör/altklasör** "klasör" içinde istenilen miktarda alt klasör oluşturur.

rmdir: Klasör silmek için kullanılır.

- **rmdir klasör** Boş bir klasörü silmek için kullanılır. Yalnızca boş klasörü siler.

rm: Dosya ve Klasör silme işlemleri için kullanılır.

- **rm dosya** "dosya"'yı siler.
- **rm -r klasör** "klasör"'ü ve alt dizinlerinde bulunan tüm dosya ve klasörleri siler.

head: Dosyanın ilk n satırını göstermek için kullanılır.

- **head -n 10 dosya** Belirtilen dosyanın ilk 10 satırını gösterir.

tail: Dosyanın son n satırını göstermek için kullanılır.

- **tail -n 10 dosya** Belirtilen dosyanın son 10 satırını gösterir.

chmod: Dosyanın izinlerini değiştirir.

- **chmod izinkodu dosya** "dosya"'nın izinlerini verilen izinkodu ile düzenler.

İzinkodu 3 haneli bir sayıdır ve dosya üzerinde gruplara göre okuma, yazma ve çalıştırma izinlerini düzenler. Birinci hane, dosya sahibinin yapabileceği işlemleri; İkinci hane, dosya grubunun yapabileceği işlemleri; Üçüncü hane ise diğer kullanıcıların yapabilecekleri işlemleri ifade eder.

Okuma izni: **4**, Yazma izni: **2**, Çalıştırma izni: **1** ve hiçbir işleme izin verilmeme durumu da **0** ile ifade edilir. Her bir hane, 0-7 arası rakam alabilir. Her bir rakam, söz konusu izinlerin toplamıdır. (Tüm izinler için, 4+2+1=7)

ls -l komutu ile terminalde dosyaların izinleri ile ilgili bilgiler de çıkar. Bu bilgilerde **r**, **w**, **x** ve **–** karakterleri bulunur. **r**: okuma, **w**: yazma, **x**: çalıştırma ve **–** karakteri ise izinsizlik durumunu açıklar.

Örneğin, **drwxrwxrwx** şeklindeki bir ifadede en baştaki **d** karakteri dosya tipini belirtir. Sonraki karakterleri üçer üçer alırsak, ilk üç karakter izinkodundaki birinci haneyi, ortadaki üç karakter ikinci haneyi, son üç karakter de izinkodundaki üçüncü haneyi belirtmiş olur. Böylece **drwxrwxrwx** ifadesi **777** izinkoduna karşılık gelmiş olur.

Benzer şekilde;

- `d-wx-w-rwx` ifadesi **327**,
- `drwxr-xr-x` ifadesi **755**,
- `drwx------` ifadesi de **700** olur.

Bir başka örnek olarak **755** kodu;

- dosya sahibinin okuma, yazma, çalıştırma gibi tüm izinlere sahip olduğunu (4+2+1=7),
- dosya grubunun okuma ve çalıştırma yapabileceğini (4+0+1=5),
- diğer kullanıcıların sadece okuma/çalıştırma yapabileceği (4+0+1=5) belirtilmiş olur.

`find`: Dosya aramak için kullanılır.

- **`find`** `/home -name "belge.txt"` Belirtilen dizindeki dosyayı arar.

`grep`: Dosya içinde metin araması gerçekleştirir.

- **`grep`** `"aranan" dosya` Belirtilen aranan metni dosyada arar.

`sort`: Sıralama işlemi için kullanılır.

- **`sort`** `dosya` "dosya" içerisindeki satırları a-z veya 0-9'a sıralar, dosya güncellenmez, sadece içeriği terminale dökülür.
- **`sort`** `-r dosya` "dosya" içerisindeki satırları z-a veya 9-0'a sıralar, dosya güncellenmez, sadece içeriği terminale dökülür.
- **`sort`** `dosya > yenidosya` "dosya" içeriği a-z veya 0-9'a sıralanır ve "yenidosya" dosyasına kaydedilir.

`ps aux`: Anlık olarak çalışan tüm prosesleri görüntüler.

`top`: Gerçek zamanlı olarak CPU ve bellek kullanımını gösterir.

`kill`: Belirli bir işlem kimliğine (PID) sahip işlemi öldürür.

- **`kill`** Proses_ID PID bilgisi verilen prosesi sonlandırır.

`ping`: Belirtilen adrese ICMP paketi göndererek, bağlantının sağlığını test eder.

- **`ping`** `-c 4 IP` Belirtilen IP adresine 4 adet paket göndererek, bağlantıyı test eder.

`read`: Terminale input açar. Kullanıcının veri girmesini sağlar.

- **`read`** `sifre` Kullanıcıdan sifre değişkenine bilgi girmesi sağlanır.
  Daha sonra **`$sifre`** şeklinde, kullanıcı tarafından girilmiş olan veri kullanılabilir.

3. BASH KABUĞUNDA ÖZEL KARAKTERLER

`*` Tüm karakterleri temsil eder. Örneğin, `ls *.txt` komutu, mevcut dizindeki tüm **.txt** uzantılı dosyaları listeler.

`?` Tek bir karakteri temsil eder. Örneğin, `ls dosya?.txt` komutu, **dosya1.txt**, **dosyaX.txt** gibi tek karakter farkıyla eşleşen dosyaları listeler.

`[]` Belirli karakter aralıklarını temsil eder. Örneğin, `ls dosya[1-3].txt` komutu, -varsa- **dosya1.txt**, **dosya2.txt**, **dosya3.txt** dosyalarını listeler.

`$` Değişkenin değerini döndürmek için kullanılır. **sifre** adlı bir değişken varsa, `echo $sifre` komutu çalıştırılarak, **sifre** değişkeninin sahip olduğu değer görüntülenebilir.

`${}` Karmaşık değişken ifadelerinde kullanılır. **dosya** bir değişken ise ve bu değişkenin değerine ek yapılacaksa, bu karakter kullanılır. `yenidosya = ${dosya}_2025.txt` gibi.

`$()` Metinsel ifadelerde komut gömme için kullanılır. `echo "Bugün: $(date)"` gibi.

`>` Komutun çıktısını dosyaya kaydeder (var olan içeriği siler). `ls > dosyalar.txt`

`>>` Komutun çıktısını dosyaya ekler (var olan içeriğin sonuna).
`echo "Yeni satır" >> dosyalar.txt`

`<` Dosya içeriği komut girdisi olarak kullanılır. `sort < dosyalar.txt`

`|` Pipe kullanımı ile bir işlemin çıktısı diğer bir işleme aktarılır.
`ls -l | grep ".txt"` komutu, listelenen dosyalarda **.txt** uzantılı olanları arar ve gösterir.

`&&` Birden fazla komutu şartlı çalıştırma için kullanılır.
`mkdir dizin && cd dizin` kodunda ilk komut başarılı olursa, ikincisi çalıştırılır.

`||` Alternatif komut çalıştırma işlemleri için kullanılır.

`ls /dizin || echo "Dizin bulunamadı!"` kodunda ilk komut başarısız olursa, ikincisi çalıştırılır.

4. DEĞİŞKENLER

Bash'te normal değişken tanımlamak için `degisken = "deger"` formatı kullanılır. Çevresel değişkenler ise sistem tarafından yönetilen ve tüm oturumlarda erişilebilen değişkenlerdir. Bu değişkenlerin değerlerini görüntülemek için terminalde `echo $DEĞİŞKENADI` komutunu yazmak yeterlidir.

- **SHELL**: Shell adını verir.
- **COLUMNS**: Terminal ekranının sütun sayısını verir.
- **LINES**: Terminal ekranının satır sayısını verir.
- **HOME**: /home klasörünün dizinini verir.
- **USER**: Anlık olarak terminali kullanan kullanıcının, kullanıcı adını verir.
- **OSTYPE**: İşletim sistemi tipini verir.
- **PATH**: /path klasörünün dizinini verir.
- **PWD**: Anlık olarak kullanıcının bulunduğu klasörün dizinini verir.

`printenv` veya `env`: Çevresel (Ortam) değişkenlerini listeler.

`unset`: Daha önce oluşturulan bir değişkeni silmek için kullanılır.
Kullanımı: `unset degisken`

5. METİN EDİTÖRLERİ

Yaygın olarak nano ve vim editörleri kullanılır. Bunların dışında da kullanılabilecek birçok editör vardır.

Nano, kullanımı kolay ve basit bir komut satırı metin editörüdür. Yeni başlayanlar için en uygun editörlerden biridir. `nano dosya.txt` komutu ile dosya.txt dosyası düzenlenir. Bu dosya varsa, varolan hali düzenlenir. Dosya yoksa oluşturulmuş olur.

Nano'da Temel Komutlar

Ctrl + O	*Dosyayı kaydet*
Enter	*Kaydetme işlemini onayla*
Ctrl + X	*Çıkış yap*
Ctrl + K	*Satırı kes*
Ctrl + U	*Kesilen satırı yapıştır*
Ctrl + W	*Metin içinde arama yap*
Ctrl + G	*Yardım menüsünü aç*

Vim, gelişmiş ve güçlü bir metin editörüdür. Hızlı düzenleme ve komutlarla kontrol imkanı sunduğu için daha çok profesyoneller tarafından tercih edilir. `vim dosya.txt`

6. KOŞULLU İFADELER

Koşullu ifade komutlarını, **.sh** uzantılı bir dosya içerisine yazıp, onu **bash dosya** şeklindeki bir komutla çağırarak çalıştırabiliriz. Koşullu ifadeleri yazarken özellikle boşluklara çok dikkat edilmesi gerekir. Dosya çağırırken kullanacağımız bazı değişkenler şunlardır:

- **$0**: Çalıştırılan dosyanın ismi.
- **$1**: Çalıştırılan dosyaya komut satırında (dosya isminden sonra) verilen birinci parametre.
- **$N**: Çalıştırılan dosyaya komut satırında (dosya isminden sonra) verilen N'inci parametre.
- **$#**: Çalıştırılan dosyaya komut satırında (dosya isminden sonra) verilen toplam parametre sayısı.

Örnek 1: if-else kodlarımızı yazdığımız dosyanın ismi **ornek1.sh** olsun ve **bash ornek1.sh** yazıp çalıştıralım.

```
if [ $0 == "ornek1.sh" ]
then echo "Dosyamızın ismi $0"
fi
```

Örnek 2: **ornek2.sh** dosyasına aşağıdaki kodları ekleyip **bash ornek2.sh** şeklinde çalıştırabiliriz.

```
sayi=$1
if [ $sayi -eq 10 ]
then echo "Sayı 10'a eşittir"
elif [ $sayi -lt 10 ]
then echo "Sayı 10'dan küçüktür"
elif [ $sayi -gt 10 ]
then echo "Sayı 10'dan büyüktür"
fi
```

Bash'te koşullu ifadelerde kullanılan **operatörler** aşağıdaki gibidir. Sonda verilen ifadelerin önüne if eklendiğinde koşullu ifadeye dönüşür.

Sayısal Karşılaştırmalar:

```
-eq → Eşittir [ $a -eq $b ]
-ne → Eşit değildir [ $a -ne $b ]
-lt → Küçüktür [ $a -lt $b ]
-gt → Büyüktür [ $a -gt $b ]
-le → Küçük veya eşittir [ $a -le $b ]
-ge → Büyük veya eşittir [ $a -ge $b ]
```

Dize (String) Karşılaştırmaları:

```
== → Eşittir [ "$a" == "$b" ]
!= → Eşit değildir [ "$a" != "$b" ]
-z → Değişken boş mu? [ -z "$a" ]
-n → Değişken dolu mu? [ -n "$a" ]
```

Dosya Kontrolleri:

```
-f → Dosya var mı? [ -f dosya.txt ]
-d → Dizin var mı? [ -d /klasor ]
-s → Dosya boş mu? [ -s dosya.txt ]
```

Örnek 3: `ornek3.sh`

```
if [ -f "test.txt" ]
then echo "Dosya mevcut!"
else echo "Dosya bulunamadı!"
fi
```

If-Else yapısının dışında **Case** yapısını da kullanmak mümkündür.

Case yapısı bir değişkenin almış olduğu farklı değerlere göre işlem yapılabilmesini sağlar.

Örnek 4: `ornek4.sh`

```
echo "Hangi mevsimdesiniz? (kış/yaz)"
read mevsim

case $mevsim in
    kış) echo "Hava soğuk!" ;;
    yaz) echo "Hava sıcak!" ;;
    *) echo "Tanımlı olmayan bir mevsim girdiniz." ;;
esac
```

7. DÖNGÜLER

Bash'te **for**, **while** ve **until** döngüleri ile çalışılabilir.

Örnek 5: Aşağıdaki for döngüsü örneği, 1 2 3 4 5 elemanlarından oluşan bir sayı dizisini gezer. **ornek5.sh**

```
for i in 1 2 3 4 5
do echo $i
done
```

Örnek 6: Aşağıdaki for döngüsü örneği, 1'den 10'a kadar (10 dahil değil) 2'şer 2'şer gidilerek oluşturulan sayı dizisini gezer. **ornek6.sh**

```
for i in {1..10..2}
do echo $i
done
```

Örnek 7: Aşağıdaki for döngüsü örneği, bulunulan dizindeki .sh uzantılı dosyaların listesini gösterir. **ornek7.sh**

```
for dosya in *.txt
do echo "Dosya: $dosya"
done
```

Örnek 8: Aşağıdaki while döngüsü örneği, 1'den 5'e kadar sayıları konsola yazdırır. **ornek8.sh**

```
sayi=1
while [ $sayi -le 5 ]
do echo "Sayı: $sayi"; ((sayi++))
done
```

Örnek 9: Until döngüsü, while'ın tersi gibi çalışır. Yani koşul yanlış oldığu sürece çalışır. Aşağıdaki until döngüsü örneği, 1'den 5'e kadar sayıları konsola yazdırır. **ornek9.sh**

```
sayi=1
until [ $sayi -gt 5 ]
do echo "Sayı: $sayi"; ((sayi++))
done
```

8. DİZİLER (ARRAYS)

Bash'te diziler parantez **(…)** kullanılarak tanımlanır ve elemanlar boşluk ile ayrılır. Aşağıda, elemanları *Elma, Armut, Çilek* ve *Muz* olan bir dizi tanımlanmıştır. Her bir dizi elemanına, sıfırdan başlayan bir index numarası ile ulaşılabilir.

Örnek 10: `ornek10.sh`

```
# dizi oluşturmak
meyveler=("Elma" "Armut" "Çilek" "Muz")


# elemanları çağırmak
echo "İlk eleman: ${meyveler[0]}"
echo "Tüm elemanlar: ${meyveler[@]}"
echo "Tüm elemanlar: ${meyveler[*]}"


# diziye eleman eklemek
meyveler+=("Karpuz")
echo "Güncellenmiş dizi: ${meyveler[@]}"


# dizinin toplam eleman sayısını göstermek
echo "Dizideki toplam eleman sayısı: ${#meyveler[@]}"


# diziden eleman silmek
unset meyveler[1]
echo "Silindikten sonra: ${meyveler[@]}"


# döngü ile tüm elemanları gezmek
for meyve in "${meyveler[@]}"
do echo "Meyve: $meyve"
done
```

9. FONKSİYONLAR

Bash'te fonksiyon oluşturmak için fonksiyon adının başına **`function`** yazmak zorunlu değildir ama tercih edilir. Kodun daha iyi anlaşılması için gereklidir.

```
function fonksiyon_adi() {
    #komutlar...
}
```

Örnek 11: Aşağıda parametre gönderilen bir fonksiyon örneği bulunmaktadır. **`ornek11.sh`**

```
function topla() {
    sonuc=$(( $1 + $2 ))
    echo "Toplam: $sonuc"
}
```

Bu fonksiyon **bir dosyada bulunduğu için**, çağırılmadan önce konsola yüklenmesi gerekir. `source ornek11.sh` komutuyla konsola yüklenmesinin ardından, fonksiyon isminden sonra girilmesi istenen parametreler, aralarında boşluk ile `topla 5 3` gibi yazılabilir. Daha sonra bu fonksiyonun, konsoldan yüklemesinin iptal edilebilmesi için `unset topla` komutu kullanılabilir.

Örnek 12: Değer döndüren fonksiyon örneği. **`ornek12.sh`**

```
function kontrol() {
    if [ $1 -gt 10 ]
    then return 1
    else return 0
    fi
}

kontrol 15
echo "Dönen kod: $?"
```

10. SONUÇ

Bash, sistem yönetimi ve betik yazımı için vazgeçilmez bir araçtır. Özellikle otomasyon gerektiren işlemlerde büyük kolaylık sağlar. Kullanıcıların Bash'in temel komutlarını ve betik yazımını öğrenmesi, verimli bir Linux/Unix deneyimi için önemlidir.

Bash dışında çeşitli kabuklar da bulunmaktadır. Her kabuğun kendine özgü avantajları ve kullanım alanları vardır.

Kabuk	Avantajları	Dezavantajları
Bash (Bourne Again Shell)	Yaygın olarak kullanılır ve Linux'ta varsayılan kabuktur.	Varsayılan ayarları sade, bazı özel özellikleri eksiktir.
sh (Bourne Shell)	Hafiftir, hızlıdır ve POSIX uyumludur.	Daha az özelliklidir, modern kabuklara göre sınırlıdır.
zsh (Z Shell)	Daha fazla kişiselleştirme imkânı, güçlü otomatik tamamlama desteği vardır.	Varsayılan olarak gelmez, öğrenme eğrisi yüksektir.
fish (Friendly Interactive Shell)	Kullanımı kolaydır, modern özellikler içerir, otomatik tamamlama gelişmiştir.	Bash betikleriyle tam uyumlu değildir.
csh (C Shell)	C programlama diliyle benzer sözdizimi sunar, geliştiricilere hitap eder.	Standart olmayan sözdizimi vardır, betik yazımı karmaşıktır.
ksh (Korn Shell)	Performansı yüksektir, sh ile uyumludur, script yazımında gelişmiş özelliklere sahiptir.	Bash kadar yaygın değildir, bazı sistemlerde varsayılan olarak gelmez.

Bash, günümüzde birçok alanda kullanılmaktadır:

- **Sistem Yönetimi:** Sunucu yapılandırmaları ve otomatik işlemler.
- **Yazılım Geliştirme:** Derleme ve test süreçlerinin otomasyonu.
- **Veri İşleme:** Büyük veri setleri üzerinde metin işleme ve düzenleme.

Bash'in sağladığı en önemli avantajlar:

- Hızlı ve etkili sistem yönetimi.
- Otomasyon ve betik yazma imkanı.
- Açık kaynaklı ve geniş destek topluluğu.