YILDIZ TEKNİK ÜNİVERSİTESİ

BİLGİSAYAR MÜHENDİSLİĞİ

İŞLETİM SİSTEMLERİ

(SHELL PROGRAMLAMA DOKÜMANI v1)

ARŞ. GRV. UĞUR ÇEKMEZ

Shell Programlama

Bu dokümanda Shell programlama konusuna giriş maksadıyla temel komutlar ve bir Shell programcığının nasıl çalıştırılabileceği ele alınacaktır.

Hello, World

```
# hello_world dosyasi
# Bunlar yorum satirlaridir
#

# bu komut, script calistirinda terminal ekranini temizler
clear

# echo komutu ekrana deger yazdirmamizi saglar. Bu degerler yazi, sayi veya degiskenlerden olusabilir
echo "Hello, World"
```

Yazmış olduğumuz bu shell scriptini çalıştırabilmek için öncelikle terminal ile dosyanın bulunduğu dizine gidip gerekli çalıştırma izinlerini aşağıdaki şekilde veriyoruz

```
$ chmod +x hello_world
```

İzinlerini vermiş olduğumuz dosyamızı çalıştırmak için aşağıdaki komut yeterlidir

```
$ sh hello_world
```

Komut çıktısı aşağıdaki şekilde olacaktır

```
$ Hello, World
```

Ekrana bilgiler yazdırma

Aşağıdaki komut dizisi ile ekrana bilgisayar hakkında bazı veriler yazdıralım

```
#
# info dosyasi
#
clear

# Oturum acmis olup o an terminali kullanan kullaniciyi ekrana yazdirir
echo "Merhaba $USER"

# date degiskeni anlik tarih bilgisi icindir. echo ve date komutlari arasindaki noktali virgul ( ; ) birden fazla komutu ard arda calistirabilmemize olanak tanir. Calistirilan komutlar birbirinden bagimsizdir.
echo "Bugun \c" ; date

# who degiskeni oturum acmis olan tum kullanicilari listeler pipe ( | ) ile birden fazla komutu ard arda isleme koyabiliriz ve soldan saga islenen komutlar, bir onceki komutun ciktisini parametre olarak alirlar
echo "Oturum acmis kullanici sayisi : \c" ; who | wc -l

# cal degiskeni ay bazinda takvimi gosterir
echo "Takvim"
cal
```

echo komutu ile ekrana bigiler yazdırırken aşağıdaki eklentilerden faydalanabiliriz

```
#
# echo_secenekleri dosyasi
#

clear

# ekrana basildiginda bir uyari sesi cikarir
echo "deneme yazi \a"

# eger yazi arasinda ise kendinden onceki bir karakteri siler
echo "deneme yazi \b"

# ekran ciktisinin sonunda yer alan yeni satiri siler
echo "deneme yazi \c"

# ekran ciktisinin sonuna bir yeni satir ekler
echo "deneme yazi \n"

# satirbasi acar
echo "deneme yazi \r"

# bir tab tusu kadar bosluk birakir
echo "deneme yazi \t"

# \ karakterinin yazilabilmesi
echo "deneme yazi \\"
```

Shell değişkenleri

Shell programlarında iki tip değişken türü yer alır. Birincisi tamamı büyük harflerden oluşan sistem değişkenleridir. Bunlar hali hazırda tanımlanmıştır ve sistemin temel bileşenlerini göstermektedirler. İkinci tip değişkenler ise tamamı küçük harflerden oluşması beklenen kullanıcı değişkenleridir.

SHELL	/usr/local/bin/fish	Shell adı
COLUMNS	193	Terminal ekranının sütun sayısı
LINES	24	Terminal ekranının satır sayısı
HOME	/Users/ugur	Home klasörünün konumu
USER	ugur	Kullanıcı adımız
OSTYPE	darwin14	İşletim sistemi tipi
PATH	/usr/local/bin:/usr/bin:/bin:/usr/sbin	Path klasörünün konumu
PWD	/Users/ugur/Desktop	Bulunduğumuz klasörün konumu

Kullanıcı değişkenleri aşağıdaki şekilde oluşturulabilir

```
#
# degisken_tanimla dosyasi
#
clear

# Degisken isimleri alt cizgi (_) veya harf ile baslar, esittir isaretinden once ve sonra bosluk konulmaz
# Degisken isimleri buyuk ve kucuk harflere duyarlidir
# Yazim aninda degeri belli olmayan degiskenler icin NULL degeri, degisken isminden sonra esittir konularak verilebilir
degisken0=
degisken1=10
degisken2="deneme"
degisken3=Deneme

echo $degisken1
echo $degisken2
echo $degisken3

# yukaridaki degiskenlerin degerleri kullanici tarafindan verildi. Degiskenlere ayni zamanda sistem degiskenleri de eklenebilir

degisken4=pwd

echo $degisken4
```

Aritmetik İşlemler

Aritmetik işlemler aşağıdaki yapıya uygun olmak durumundadır:

expr operation

expr, aritmetik işlem için kullanılan etikettir. **operation** ise aşağıdaki yapıya uygun olmak durumundadır:

bir sayı veya **operation operator operation**

Aritmetik işlemler aşağıdaki şekilde örneklendirilebilir:

```
#
# aritmetik_islemler dosyasi
#

clear

expr 1 + 2 # toplama
expr 3 \* 4 # carpma
expr 2 - 1 # cikarma
expr 10 % 3 # kalan alma (bazi terminallerde \% seklinde yazmak gerekebilir)

echo `expr 3 + 4` # Burada back quote kullanilir. Bu sembol ~ tusunun altinda yer alir
```

Kullanıcı Veri Girişi

Kullanıcının terminal ekranından veri girişi yapabilmesi için read komutu kullanılır. Komut işlendiği anda kullanıcı veri girişi yapana kadar sistem beklemede kalır. Girilen veri, bir değişkende tutulur.

```
#
# kullanici_veri_girisi dosyasi
#
clear

echo "Lutfen adinizi giriniz"
read kullanici_adi
echo "Merhaba $kullanici_adi !"
```

Eşleştirme Sembolleri (Wild Cards)

***** Tüm isimlerle eşleşir

? Yalnızca bir karakter ile eşleşir

[...] İçine yazılan herhangi bir karakter ile eşleşir

```
#
# eslestirme dosyasi
#
clear

ls * # tum dosyalar listelenir
ls a* # a ile baslayan tum dosyalar listelenir
ls *.py # uzantisi .py olan tum dosyalar listelenir
ls deneme_*.py # deneme_ ile baslayip uzantisi .py olan tum dosyalar listelenir

ls ? # tek karakterli adi olan tum dosyalar listelenir
ls deneme? # deneme ile baslayip ardindan yalnizca bir karakter gelen tum dosyalar listelenir
ls [de]* # d veya e ile baslayan tum dosyalar listelenir
```

Diğer İşlem Komutları

Aşağıdaki komutları tek tek deneyip ne tür çıktılar verdiğini inceleyebiliriz

```
#
# cesitli_dosya_islemleri dosyasi
#

mkdir klasor_ismi
cd klasor_ismi
touch deneme
ls deneme
mv deneme deneme.js
ls deneme # hata mesaji verecek
ls deneme.js
echo “dosyaya yaz beni” > deneme.js
cat deneme.js

touch deneme2
echo “1” > deneme2
echo “3” > deneme2 # > isareti dosyaya bastan yazar
echo “1” >> deneme2 # >> isareti, dosyaya yeni bir satir ile yazmaya devam eder
echo “7” >> deneme2
echo “5” >> deneme2

sort deneme2 # satirlari siralar (A-Z , 0-9, vs.) ancak dosyayi guncellemez

sort deneme2 > deneme3 # siralanmis dosyayi yeni bir dosyaya yazar

rm deneme2 # dosyayi siler
```

Pipe (|)

Birden fazla komutu birbirine bağlamaya yarar. Pipe kullanarak bir komutun çıktısını diğer bir komuta parametre olarak verebiliriz. Bu sayede ilk çıktıyı geçici bir alana yazıp oradan tekrar kullanmak gibi üçüncü bir işlem yapmamış oluruz.

```
#
# pipe dosyasi
#

ps aux # ps aux ile anlik olarak kullanicinin calisan islemlerini satir satir gorebiliyoruz
egrep 1 deneme2 # egrep ile istedigimiz bir klasorde veya dosyada, dosyalar icinde bulunan belli metinleri filtreleyebiliyoruz. Buradaki islem, deneme2 'nin icinde 1'i filtrelemektir.

ps aux | egrep root # Iki komutu birlestirdigimizde su sonuc cikar : icerisinde root kelimesi gecen satirlari filtrele. Ilgili aramayi da ps aux komutu ciktisinden edin.
```

Process İşlemleri

ps aux komutu ile ekrana yazılan satırların yapısı aşağıdaki şekildedir

```
USER               PID  %CPU %MEM      VSZ    RSS   TT  STAT STARTED      TIME COMMAND
```

USER : Bu işlemi çalıştıran kullanıcı
PID : Process ID
%CPU : İşlemin anlık olarak kullandığı CPU yüzdesi
%MEM : İşlemin anlık olarak kullandığı bellek yüzdesi
VSZ : Kullanılan sanal bellek miktarı (Kilobayt olarak)
RSS : Resident Set Size, İşlemin fiziksel olarak boyutu?
TTY : Bu işlemi kontrol eden terminal penceresi
STAT : Çoklu karakter işlem durumu?
STARTED : Ne zaman başlatıldı
TIME : Ne kadar süredir çalışıyor
COMMAND : İşlem dosyası nerede

Dosya Karakter Dönüştürme ve Düzenleme

Karakterler ve kelimeler üzerinde işlem yapabilmek için kullanılan komutlardan biri olan **tr** için kullanım örnekleri aşağıdaki şekildedir

```
#
# tr_komutu dosyasi
#

# asagidaki komuttan sonra tr, girilen kucuk harfli karakterleri buyuk harfe cevirecektir
echo "col1;col2;col3;col4" | tr a-z A-Z

# Birebir karakter bulma ve degistirme icin asagidaki komut kullanilabilir
echo "col1;col2;col3;col4" > birdosya.txt
cat birdosya.txt | tr ';' ',' # tr, icerigini aldigi dosyadaki noktali virgulleri virgul ile degistirecek

echo "col1;col2;col3;col4" > birdosya.txt
cat birdosya.txt | tr -d ';' # tr, icerigini aldigi dosyadaki tum noktali virgulleri silecek
```

Sütun Değer Kıyaslamaları

Bir dosyadaki satırların belli sütunlarının örneğin sayısal değerlerini kıyaslamak için **awk** komutunun kullanım örneği aşağıdaki şekildedir

```
#
# awk_komutu dosyasi
#

# asagidaki komuttan sonra tr, girilen kucuk harfli karakterleri buyuk harfe cevirecektir
echo "ahmet;4;1" > birdosya.txt
echo "ahmet;3;2" >> birdosya.txt
echo "ahmet;2;3" >> birdosya.txt
echo "ahmet;1;4" >> birdosya.txt
echo "ahmet;0;5" >> birdosya.txt

cat birdosya.txt  | awk -F ';' 'int($2)>3' # ikinci sutunu 3ten buyuk olan satirlar
cat birdosya.txt  | awk -F ';' 'int($3)<=4' # ucuncu sutunu 4ten kucuk veya esit olan satirlar
```

Karşılaştırma İşlemleri

Shell üzerinde karşılaştırma işlemleri (>, <, ==, +, -, /, %) yapabilmek için hesap makinesini açabiliriz. Bunun için bc komutunu kullanabiliriz. Karşılaştırma işlemlerinde true deyimine karşılık 1, false deyimine karşılık 0 cevabı döndürülür.

```
$ bc
```

```
#
# hesap_makinesi dosyasi
#
bc

3>1 # 1
3<1 # 0
1==1 # 1
1+1 # 2
3-1 # 2
5%5 # (Kalan) 0
```

Sorgular

Sorgu işlemleri if then else fi komutları ile yapılmaktadır. Sorgu dogru (true döndürüyor) ise veya çıkış durumu (exit status) 0 ise if sorgusu koşulu sağlanmış olur.

```
#
# sorgular dosyasi
#

# $0 = shell script dosyamizin adi ,
# $1 = scripti calistirirken komut satirina girdigimiz birinci parametre ,
# $2 = scripti calistirirken komut satirina girdigimiz ikinci parametre ..

touch birdosya.txt # Dosya olusturduk
echo "Ornek bir metni dosyaya koyduk" > birdosya.txt # dosyaya veri ekledik

if cat $1 # eger cat $1 komutu sonuc dondurur ise (exit status 0)
  then     # bu durumda
    echo "$1 isimli dosya mevcut" # bu komutu calistir
  else # eger degil ise
    echo "$1 isimli dosya mevcut DEGIL" # bu komutu calistir
fi # sorguyu tamamla

# [ expr ] komutu iki degeri kiyaslamak icin kullanilir. Sonuc dogru ise 0,
# degil ise sifirdan farkli bir deger dondurulur
if [ $# -gt 3 ] # eger girilen toplam parametre sayisi 3ten buyuk ise
  then             # bu durumda
    echo "$# adet parametre girdiniz" # bu komutu calistir
  else # eger degil ise
    echo "$# adet parametre yeterli degildir" # bu komutu calistir
fi # sorguyu tamamla

# test komutu iki degeri kiyaslamak icin kullanilir. Sonuc dogru ise 0,
# degil ise sifirdan farkli bir deger dondurulur
if test $3 = $4 # eger $3 parametresi ile  s4 ayni degerde string ise
  then          # bu durumda
    echo "$3 ile $4 ayni" # bu komutu calistir
  else # eger degil ise
    echo "$3 ile $4 farkli" # bu komutu calistir
fi # sorguyu tamamla

# birden fazla alt alta sorgu yapmak icin elif ( else if ) omutu kullanilir
if [ -w $1 ]# eger $1 bir dosya ve okunabilir ise
  then          # bu durumda
    echo "$1 yazilabilir bir dosyadir" # bu komutu calistir
  elif [ -r $1 ] # ilk kosul saglanmadiysa buna bakalim
   then          # bu durumda
    echo "$1 yazilamaz ama okunabilir bir dosyadir" # bu komutu calistir
  else # eger degil ise
    echo "$1 ne yazilabilir ne de okunabilir bir dosyadir" # bu komutu calistir
fi # sorguyu tamamla
```

Karşılaştırma	Anlamı	test ile kullanımı	expr olarak kullanımı
-eq	eşittir	if test 2 -eq 2	if [2 -eq 2]
-ne	eşit degildir	if test 2 -ne 1	if [2 -ne 1]
-lt, -gt	küçüktür, büyüktür	if test 2 -lt 3	if [2 -gt 3]
-le, -ge	küçük ya da eşittir, büyük ya da eşittir	if test 2 -le 2	if [2 -ge 2]

Karşılaştırma	Anlamı
string1 = string2	stringler esit midir?
string != string2	stringler esit degil midir?
string1	string1'in degeri varsa true
-n string1	string1 tanimli ise true (NULL olabilir)
-z string1	string1 tanimli ise ama degeri NULL ise true

Karşılaştırma	Anlamı
-s dosya	bos olmayan bir dosya ise true
-f dosya	yalnizca dosya ise true (klasor false)
-d klasor	yalnizca klasor ise true (dosya false)
-w dosya	yazilabilir bir dosya ise true
-r dosya	okulabilir bir dosya ise true
-x dosya	calistirilabilir bir dosya ise true

Karşılaştırma	Anlamı
!expr1	DEGIL (NOT)
expr1 -a expr2	VE (AND)
expr1 -o expr2	VEYA (OR)

Döngüler

```
#
# donguler dosyasi
#

# for icin temel kullanim. $i degiskeni, for icindeki i sayacini ifade eder
for i in 1 2 3 4 5
  do
    echo “sayac $i”
done

# yukaridaki ile ayni sonucu cikarir
for i in {1..5}
  do
    echo “sayac $i”
done

# yukaridaki ile ayni sonucu cikarir
for ((i=1; i<=5; i++))
  do
    echo “sayac $i”
done

# 1’den baslar 10’a kadar 2’ser 2’ser sayar
for i in {1..10..2}
  do
    echo “sayac $i”
done

#  $(..) kalibi ile bir shell komutunun ciktisini for icin kullanabiliriz
for i in $(ls)
do
    echo “dosya $i”
done

#  liste cikaracak her komutu for icin kullanabiliriz. Asagidaki ~/ (home) dizini
# altindaki dosyalari ve klasorleri listeler. Dolayisi ile her bir dongude i, dosya ismi olacaktir
for i in ~/*
do
    if [ -f $i ]
    then
      echo “dosya : $i”
    elif [ -d $i ]
    then
      echo “klasor : $i”
    else
      echo “bilemedim bu ne : $i”
    fi
done
```

```
#
# donguler2 dosyasi
#

i=10 # sayacimiza ilk degeri verdik
while [1 $i -gt 0 ]
  do
    echo “sayac $i”
    i=`expr $i - 1` # sayacimizi bir azalttik
done
```

Case ifadesi

Case ile verilen degerler arasindan secimler yapilip buna gore ciktilar elde edilebilir

```
#
# case dosyasi
#

read ne_ariyorum

case $ne_ariyorum in
   “arac”) echo “arac icin www.arac.com”;; # degiskenimiz arac ise
   “ev”) echo “ev icin www.ev.com”;; # degiskenimiz ev ise
   *) echo “biz sadece arac ve ev icin yonlendirebiliyoruz”;; # diger durumlar
esac
```