



# Linux Plus

## Linux Documentation Getting Help

### Man Pages

Dökümatasyon dosyasıdır. man <Command> şeklinde yazılır. Komutu detaylı olarak anlatır. Tüm optionları ile beraber

### Info Pages

info <command> Genelde komutun nasıl geliştiği vs. gibi hususları düz metin olarak anlatan komut. Fazla detaylı

### Whatis Command

whatis <command> komutun kısaca tanıtılmasıdır. Hap bilgi içerir.

### Apropos Command

Komutun tamamını hatırlamıyor ancak belirli kısmını hatırlıyorsak bu komut sayesinde ilgili komutları karşınıza çıkarır. apropos <command>

```
clarusway@DESKTOP-UN6T2ES:~$ apropos pwd
pwd (1)          - print name of current/working directory
pwdx (1)         - report current working directory of a process
unix_chpwd (8)   - Helper binary that verifies the password of the current user
```

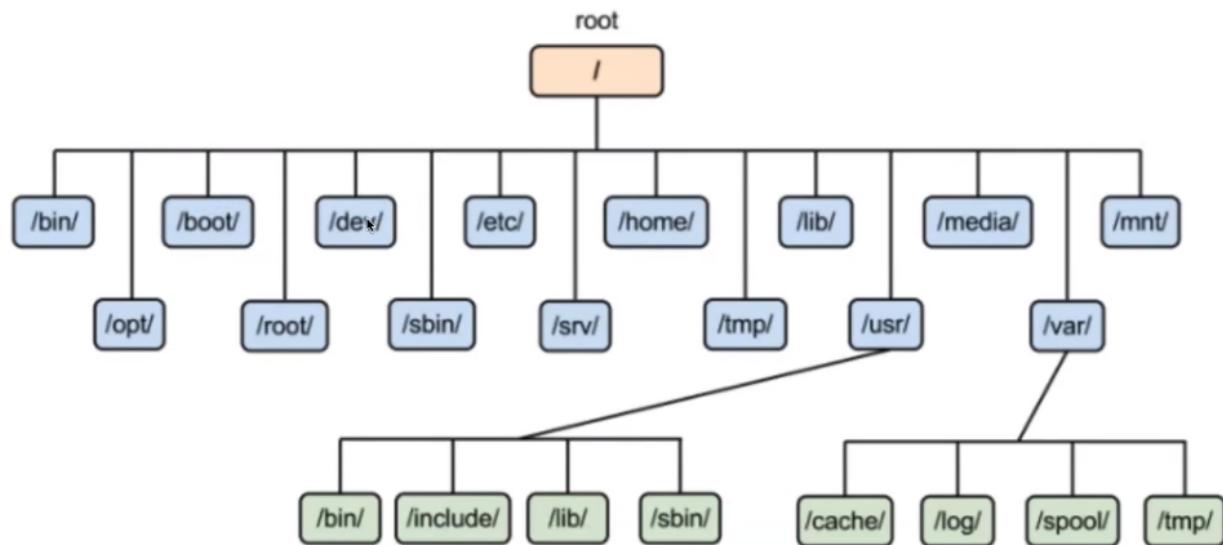
### -- Help Option

Komut hakkında kısa bilgi ve bu komutla bereber kullanabileceğimiz optionları sergiler  
ls --help

## Files:

Linux sistemi, bir dosya ile bir dizin arasında hiçbir fark yaratmaz, çünkü bir dizin yalnızca diğer dosyaların adlarını içeren bir dosyadır. Dizin özel bir dosya türüdür, ancak yine de (büyük/küçük harfe duyarlı!) bir dosyadır.

Linux'ta hemen hemen her şey bir dosyadır. Dosya olmayanlar ise process'tir.



## Dosya Tipleri ve Renkleri:

-rw-----	Regular File
drwxr-xr-x.	Directory File
lrwxrwxrwx.	Link File
crw-rw----	Character Device File
brw-rw----	Block Special File
srw-rw-rw-	Socket File
prw-----.	Named Pipe File

Color	Meaning
Blue	directories
Red	compressed archives
White	text files
Pink	images
Cyan	links
Yellow	Devices
Green	Executables
flashing red	broken links

ls- al

# of HardLinks	owner of file	Size in Bytes	Directory or File Name
File type and Access Permissions	Usergroup	Date & Time	
drwxr-xr-x 22 n100 n100 4096 2012-08-18 18:09 .	n100	2012-08-18 18:09	.
drwxr-xr-x 3 root root 4096 2012-08-18 04:36 ..	root	2012-08-18 04:36	..
-rw----- 1 n100 n100 117 2012-08-18 18:12 .bash_history	n100	2012-08-18 18:12	.bash_history

## Klasörün sahipliğini değiştirme (Change Owner)

```
total 3
drwxr-xr-x 4 root root 33 Apr 20 18:29 aws
drwxr-xr-x 10 root root 181 May  6 08:33 nexus
drwxr-xr-x 2 root root  6 Aug 16 2012 nh
```

Aşağıdaki chown komutu ile sahipliği değiştiriyoruz. -R: recursive yani tüm klasör için değiştiriyor.

```
sudo chown -R ec2-user:ec2-user /opt/nexus
sudo chown -R ec2-user:ec2-user /opt/sonatype-work
```

Aşağıda görüldüğü gibi nexus klasörünün owner'ı ec2-user. Yani bu klasöre sudo yazmadan da girebilirim

```
drwxr-xr-x 4 root      root      33 Apr 20 18:29 aws
drwxr-xr-x 10 ec2-user  ec2-user  181 May  6 08:33 nexus
drwxr-xr-x 2 root      root      6 Aug 16 2012 nh
```

## File Contents Commands

head file.txt	ilk 10 satırı ekrana getirir	cat file.txt -n	dosyanın içeriğini görüntüler. -n ile satır sayısı ekler
head -5	ilk n satırı gösterir	cat file1 file2 file3	üç dosyanın da

			İçeriklerini birleştirip alt alta yazar (concatenate)
<b>tail</b>	Son 10 satırı gösterir	<b>cat &gt; text.txt</b>	komutu ile sana bir satır açar ve dosya içeriğini düzenleyebilirsin. Ekstradan text editör açmaya gerek yok
<b>tail -8</b>	Son n satırı gösterir	<b>cat A.txt &gt; B.txt</b>	A'nın içeriğini B'ye kopyalar
<b>echo emre</b>	Ekrana emre yazdırır	<b>cat A.txt B.txt &gt; C.txt</b>	A ve B'yi alt alta yapıp C'ye kaydeder.
<b>echo emre/ "emre" &gt; file.txt</b>	dosyaya emre yazdırır. Önceki yazınları siler	<b>more file.txt</b>	dosya içeriğini satır satır gösterir.
<b>echo emre/ "emre" &gt;&gt; file.txt</b>	Dosyanın en alt satırına emre'yi ekler. Önceki yazınlar korunur.	<b>less file.txt</b>	dosyanın tamamını indirmeden yavaş yavaş açar
<b>tac file.tx</b>	dosya satırlarını tersten yazar 1-5 yerine 5-1	<b>find /home -name emre</b>	Home klasöründe emre isimli dosyaları arar
<b>grep "Start" file.txt</b>	Dosya içindeki Start kelimesinin geçtiği satırları gösterir	<b>find /home -iname emre</b>	emre isimli klasörleri büyük küçük harfe bakmadan arar. Emre-emre-EMRE vs.
<b>zip -r name.zip /home/emre</b>	emre klasörünü name ismiyle zipler	<b>find . -type f -name "*.txt"</b>	Bulunduğun klasörde txt ile biten tüm dosyalar (-f tipli)



echo ile birden çok satırı tek bir dosyaya yazdırıcasak ya EOF'yi kullanız ya da tırnak ("", ") kullanız.

```
echo '
from flask import Flask
app = Flask(__name__)
...
app.run(host="0.0.0.0", port=80)
' > welcome.py
```

- [ec2-user@docker\_instance image]\$ echo "
line 1
line 2
line 3
" > python.py

```
clarusway@DESKTOP-UN6T2ES:~$ cat file1
this is file1
clarusway@DESKTOP-UN6T2ES:~$ cat file2
this is file2
clarusway@DESKTOP-UN6T2ES:~$ cat file3
this is file3
clarusway@DESKTOP-UN6T2ES:~$ cat file1 file2 file3
this is file1
this is file2
this is file3
clarusway@DESKTOP-UN6T2ES:~$ cat file1 file2 file3 > all
clarusway@DESKTOP-UN6T2ES:~$ cat all
this is file1
this is file2
this is file3
```

grep komutu optionları

grep [options] pattern [files]

Options	Description
-c	This prints only the number of lines that match a pattern
-h	Do not display the filenames headers.
-i	Ignores, case for matching
-l	Displays list of a filenames only.
-n	Display the matched lines and their line numbers.
-v	This prints out all the lines that do not matches the pattern

## ENVIRONMENTS VARIABLES

Sık kullanılan değişkenler

Variable	Description
PATH	This variable contains a <b>colon (:) -separated list of directories</b> in which your system looks for executable files.
USER	The username
HOME	<b>Default path to the user's home directory</b>
EDITOR	Path to the program which edits the content of files
UID	User's unique ID
TERM	Default terminal emulator
SHELL	Shell being used by the user
LANG	The current locales settings.

Bu değişkenlerin başına \$ işaretini konularak kullanılır. Büyük harf ile kullanılması tavsiye edilir. Değişkenden önce de echo komutu çalıştırılır.

```
exit
ubuntu@ip-172-31-95-146:~$ echo $HOME
/home/ubuntu
```

**\$EDITOR** default olarak hangi text editörün kullanılacağı bilgisini içerir

**\$TERM:** terminalde kullanılan emülatör bilgisini içerir.

**UID:** root için 0 kullanıcılar için 1000'den başlar yukarı doğru gider. \$UID ile kullanıcı IDsini sorguluyoruz

```
ubuntu@ip-172-31-91-75:~$ echo $UID
1000
ubuntu@ip-172-31-91-75:~$ sudo su
root@ip-172-31-91-75:/home/ubuntu# echo $UID
0
```

**\$PATH:** Altında belirli dosya yollarının bulunduğu bir değişken. Bu değişkene yeni yollar da ekleyebiliriz. \$PATH değişkenine bir örnek vermek gerekirse, listeleme komutu olan ls /bin klasöründe bulunur. Komutun aslı /bin/ls tir. PATH değişkeninde /bin klasörü tanımlı olduğu için tekrar yazmamıza gerek olmuyor ve sadece ls komutu yazıldığında linux \$PATHin altında ls komutunu bularak çalıştırıyor.

```
clarusway@DESKTOP-UN6T2ES:~$ export PATH=$PATH:/games/awesome
clarusway@DESKTOP-UN6T2ES:~$
```

Yukarıda awesome dosyasını tek başına çağırabiliriz bu sayede



**\$PS1:** Prompt satırının renk vs. özelliklerini tutan değişken

```
[ec2-user@ip-172-31-91-145 ~]$
```

**\$PS2:** cat << EOF komutunu birden fazla satır komut yazarken kullanıyorduk. Sonuna EOF yazmadan komutu kapatmaz. Altta açılan yeni satırların değeri \$PS2'dur.

```
[ec2-user@ip-172-31-91-145 ~]$ cat << EOF
> |
```

**\$PS3:**

COMMON COMMAND

Command	Description
env	The <code>env</code> command is a shell command used to display and manipulate environment variables. It is used to list down environment variables.
printenv	The command prints all or the specified environment variables.
set	The command sets or unsets shell variables. When used without an argument it will print a list of all variables including <b>environment and shell variables, and shell functions</b> .
unset	The command deletes shell and environment variables.
export	The command sets environment variables.

env ve printenv ilgili değişkeni ekrana yazdırabiliriz. unset ile ilgili değişkeni sileriz.

```
[ec2-user@ip-172-31-88-75 aws]$ printenv PATH
/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/home/ec2-user/.local/bin:/home/ec2-user/bin
[ec2-user@ip-172-31-88-75 aws]$ env $PATH
env: /usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/home/ec2-user/.local/bin:/home/ec2-user/bin: No such file or directory
```

**SHELL Variables:** KEY=VALUE şeklinde tanımlıyoruz. Geçicidir. Shell i kapattığımızda kaybolur. Env ve printenv ile değişkenlere erişemeyiz. echo ile değişkenleri görüntüleyebiliriz. Script içinden çağrırlamaz. Unset ile kaldırırız.

**Enviroments Variables:** export KEY=VALUE şeklinde tanımlanır. Kalıcıdır. Shell kapansa bile kalır hatta diğer kullanıcılar da görebilir. 0-9, a-z, A-Z ve (\_) yi kullanabiliyoruz. Diğer özel karakterler yasak ve sayı ile başlanmaz. Env ve printenv ile erişilebilir. Script içinden çağrırlabilir. Unset ile kaldırırız.

İki değişken tanımladık. İkisi de Shell variables. Daha sonra birincisini export ile Env.Variables'a çevirdik.

```
[ec2-user@ip-172-31-88-75 ~]$ var1=one  
[ec2-user@ip-172-31-88-75 ~]$ var2=two  
[ec2-user@ip-172-31-88-75 ~]$ export var1  
[ec2-user@ip-172-31-88-75 ~]$
```

echo ile yazdırınca her ikisi de gözükmür

```
[ec2-user@ip-172-31-88-75 ~]$ echo $var1 $var2  
one two
```

printenv ve env sadece Env.Var.ları gösterir

```
[ec2-user@ip-172-31-88-75 ~]$ printenv var1 var2  
one
```

Aynı şekilde bash komutunu yazdıktan sonra shell var.ları echo ile dahi çağrıramayız.

**Bash komutu** terminali yeniliyor. Yeni terminale geçince önceki shelldeki değişkenler silinir. Ancak Enviroment Variables'lar değişmez

```
[ec2-user@ip-172-31-88-75 ~]$ bash  
[ec2-user@ip-172-31-88-75 ~]$ echo $var1 $var2  
one
```

Env ve shell var.ların temel farkı shell daha yüzeysel ve alt işlemlerde geçerli olmayacağıdır. Ancak env.ler değerini silmediğiniz sürece kalıcıdır.



Aşağıda bir environment bir de shell variable tanımladık. echo ile çağrınlca ikisi de ekran'a geldi. "printenv" ile çağrınlca sadece environment geldi, shell gelmedi.

```
[ec2-user@komputer ~]$ export NAME=Emre  
[ec2-user@komputer ~]$ SURNAMe=Olgun  
[ec2-user@komputer ~]$ echo $NAME $SURNAMe  
Emre Olgun
```

```
[ec2-user@komputer ~]$ printenv NAME SURNAMe  
Emre
```

bash komutu ile terminali resetlediğimiz zaman shell variable silindiği için SURNAMe değişkeni gelmedi.

```
[ec2-user@komputer ~]$ bash  
[ec2-user@komputer ~]$ echo $NAME $SURNAMe  
Emre
```

Daha sonra bir .sh dosyası yazıp içine NAME ve SURNAMe değişkenlerini tanımlarsak \$SURNAMe terminalden silinmiş olsa bile .sh'ın içinde olduğu için sonuç verir.

The screenshot shows a terminal window with the following content:

```
home > ec2-user > my-first-script.sh  
1  #!/bin/bash  
2  NAME="Emre"  
3  SURNAME="OLGUN"  
4  echo "Hello $NAME $SURNAME"  
5  
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL  
[ec2-user@komputer ~]$ ./my-first-script.sh  
Hello Emre OLGUN  
[ec2-user@komputer ~]$
```

\$NAME'i dosyadan silsek bile terminalde environment olarak tanımlandığı için sonuç verir.

```
home > ec2-user > my-first-script.sh
1  #!/bin/bash
2
3  SURNAME="OLGUN"
4  echo "Hello $NAME $SURNAME"
5
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

```
[ec2-user@komputer ~]$ ./my-first-script.sh
Hello Emre OLGUN
```

Ancak scriptten \$SURNAME'i silince, shell variable olması hasebiyle terminalden de silindiği için \$SURNAME gelmez.

```
home > ec2-user > my-first-script.sh
1  #!/bin/bash
2
3
4  echo "Hello $NAME $SURNAME"
5
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

```
[ec2-user@komputer ~]$ ./my-first-script.sh
Hello Emre
[ec2-user@komputer ~]$
```

Terminalden yeni bir SURNAME değişkeni tanımladık. echo ile yazdırınca geldi. Ancak .sh'lı dosyayı çağrırdığımızda dosyanın içindeki SURNAME değeri geldi. Çünkü her zaman öncelik .sh dosyası. Eğer orada yoksa terminalden atanan değer döner.

The screenshot shows a terminal window with the title bar 'my-first-script.sh'. The terminal displays the contents of the script file:

```
#!/bin/bash
NAME="Emre"
SURNAME="OLGUN"
echo "Hello $NAME $SURNAME"
```

Below the script, there are tabs for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', and 'TERMINAL'. The 'TERMINAL' tab is selected, showing the following command-line interaction:

```
[ec2-user@komputer ~]$ export SURNAME=YILMAZ
[ec2-user@komputer ~]$ echo $NAME $SURNAME
Emre YILMAZ
[ec2-user@komputer ~]$ ./my-first-script.sh
Hello Emre OLGUN
```

**alias:** Bu komut sürekli tekrar eden bir şeyi başka (daha kısa) bir karaktere atamaya yarayan bir değişkendir.

```
r:~/lesson$ alias k=kubectl
r:~/lesson$ k get no
NAMES   ROLES
kubernetes-control-plane   41m   v1.26.2
```

**Source:**

`source` komutu ile bir dosya içine yazdığımız key=value değerlerini bulunduğuuz yere alabiliyoruz. Bu şekilde echo \$... komutu ile bu değişkeni terminalden çağırabiliriz.

```
[ec2-user@ip-172-31-67-136 ~]$ echo "The_Best=Emre" > DevOps
[ec2-user@ip-172-31-67-136 ~]$ echo $The_Best

[ec2-user@ip-172-31-67-136 ~]$ source DevOps
[ec2-user@ip-172-31-67-136 ~]$ echo $The_Best
Emre
```

## read [variable-name] Komutu

read komutu ile kullanıcıdan alınan değer read'de belirtilen değişkene atanır. Aşağıda kullanıcının yazdığı değer name değerine atanır.

**variable-name]**

```
#!/bin/bash

echo "Enter your name: "
read name
echo Hello $name
```

```
[ec2-user@ip-172-31-36-108 ~]$ ./run.sh
Enter your name:
Raymond
Hello Raymond
[ec2-user@ip-172-31-36-108 ~]$ 
```

Bu işlemi daha kısa yazabiliriz. Bunun için -p optionu kullanılır.

```
my-first-script.sh
1  #!/bin/bash
2  read -p "Please enter your name: " NAME
3
4  echo "Hello $NAME"
```

Password gibi yazarken ekranın görülmemesi gereken bilgiler istiyorsak -s (secure) optionunu kullanabiliriz. Böylece şifremizi girerken ekranın gösterilmez.

```
read -s -p "Please enter your password: " PASSWORD
echo "$PASSWORD"
```

## Command Line Arguments

Bu argumanları scripti çağırırken yazıyoruz. Örneğin;

```
● ubuntu@ip-172-31-86-222:~$ my-first-script.sh first_argument second_argument
script name is /home/ubuntu/my-first-script.sh
first argument is first_argument
second argument is second_argument
2 arguments have been passed to script
we have those arguments: first_argument second_argument
```

\$0	Bash scriptin ismi
\$1-\$9	Scriptteki ilk 9 argumanı getirir.
\${10}	9'dan sonraki argümanlar {} içinde gösterilir.
\$#	Scriptte kaç tane arguman olduğunu gösterir.
\$@	Scriptteki tüm argumanları yan yana gösterir.
\$?	Bir önceki komutun çalışıp çalışmadığını gösterir. 0 ise çalıştı, 0 dışında bir sayı ise çalışmadı.
\$\$	scriptteki işlem ID'sini gösterir.
\$USER	scripti çalıştırılan kullanıcının adı
\$HOSTNAME	scripti çalıştırılan makinanın adı
\$SECONDS	Script çalışmaya başlayalı kaç saniye geçti onu gösterir.
\$RANDOM	4-5 haneli rastgele bir numara üretir. Şifre vs belirlerken kullanılabilir.
\$LINENO	O kodun script içindeki satır numarasını yazdırır.

## Simple Arithmetic

expr ifadesini sadece rakamlar ile kullanacaksak değerleri ve kullanılacak işlemi yazmamız yeterli. Ancak bir stirng ifade ile beraber kullanılacaksa başında ve sonunda ` işaretü ile beraber kullanılır. Aşağıda görüldüğü gibi işlemin sağ ve solunda birer boşluk olması lazım.

**expr** command **print** the value of expression to **standard output**.

**expr item1 operator item2**

```
home > ec2-user > my-first-script.sh
1  #!/bin/bash
2  first=0
3  second=1
4  expr $first + $second
5  expr $first+$second
6  echo "sum of first and second number is: " `expr $first + $second`
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    **TERMINAL**    PORTS

```
[ec2-user@komputer ~]$ my-first-script.sh
1
0+1
sum of first and second number is: 1
```

let komutunda işlem sonucunu let'in yanındaki değişkene atıyoruz. expr'da ise sadece ekranda gözükmür. Bir yere kaydedilmez. Dikkate edilmesi gereken şey değişken arasında boşluk olmayacağı

```
home > ec2-user > my-first-script.sh
1
2  first=0
3  second=1
4  let TOPLAM=first+second
5  echo $TOPLAM
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    **TERMINAL**

```
[ec2-user@komputer ~]$ my-first-script.sh
1
```

**let** is a builtin function of Bash that helps us to do simple arithmetic. It is similar to **expr** except instead of printing the answer **it saves the result to a variable**.

```
let <arithmetic expression>
```

`$()`'de ise parantez içi boşluksuz yazılmalı

```
home > ec2-user > my-first-script.sh
2   first=0
3   second=1
4   echo "TOTAL = $((first+second))"
5

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

[ec2-user@komputer ~]$ my-first-script.sh
TOTAL = 1
[ec2-user@komputer ~]$ 
```

We can also evaluate arithmetic expression with double parentheses.

**`$((arithmetic expression))`**

**num++, ++num, num--, --num**

`++` ya da `--` 1 topla ya da 1 çıkar anlamına geliyor.

```

#!/bin/bash

number=10
let new_number=number++
echo "Number = $number"
echo "New number = $new_number"

number=10
let new_number==number
echo "Number = $number"
echo "New number = $new_number"

~
[ec2-user@ip-172-31-91-206 ~]$ ./run.sh
Number = 11
New number = 10
Number = 9
New number = 9
[ec2-user@ip-172-31-91-206 ~]$ █

```

## Quotes

### Double Quotes

- The double quote ( "quote" ) protects everything enclosed between two double quote marks except \$, ', " and \.

echo "\$SHELL"  
echo "/etc/.conf"

### Single Quotes

- The single quote ( 'quote' ) protects everything enclosed between two single quote marks.

echo '\$SHELL'  
echo '/etc/.conf'

### Backslash

- Use the backslash to change the special meaning of the characters or to escape special characters within the text such as quotation marks.

echo "Path is \\$PATH"

Çift tırnak özel işaretlerin anlamını korur. Tek tırnak bunları korumaz anlamını yitirir.  
Backslash() kullanırsan sadece başında bulunduğu işaretin normal yapar gücünü alır.

```
● ubuntu@ip-172-31-95-146:~$ echo "$SHELL"  
/bin/bash  
● ubuntu@ip-172-31-95-146:~$ echo '$SHELL'  
$SHELL  
● ubuntu@ip-172-31-95-146:~$ echo "\$SHELL is \$SHELL"  
$SHELL is /bin/bash  
○ ubuntu@ip-172-31-95-146:~$ █
```

## MANAGING USERS AND GROUPS

### SUDO Command (Super User Do)

/etc/sudoers klasöründe kullanıcıların yetkileri saklanır. Hangi komutlar için root yetkisi olması gereklidir.

Herhangi bir komutun önüne sudo yazılırsa o komut admin yetkisi ile çalıştırılır

!! : Son yazılan komutu tekrar dönderir.

```
[ec2-user@ip-172-31-88-75 ~]$ cd /home  
[ec2-user@ip-172-31-88-75 home]$ !!  
cd /home  
[ec2-user@ip-172-31-88-75 home]$ █
```

Bazı SUDO komutları

Commands	Meaning
sudo -l	List available commands.
sudo command	Run command as root.
sudo -u root command	Run command as root.
sudo -u user command	Run command as user.
sudo su	Switch to the superuser account.
sudo su -	Switch to the superuser account with root's environment.
sudo su - username	Switch to the username's account with the username's environment.
sudo -s	Start a shell as root
sudo -u root -s	Same as above.
sudo -u user -s	Start a shell as user.

**sudo su** (Switch User) komutu Root kullanıcıya geçer ancak bu komutla home klasörü değişmez. Yani bu komutu yazdıktan sonra Home klasöre gitmek istersek root kullanıcının home klasörü olan /root'a geçiş yapmaz. Mevcut kullanıcının home klasörüne gider. Kullanıcının env.var.'ları değişmez.

```
[ec2-user@ip-172-31-88-75 ~]$ sudo su  
[root@ip-172-31-88-75 ec2-user]# cd /home  
[root@ip-172-31-88-75 home]#
```

Tamamen root değiliz sadece root kıyafeti giydik. Ancak **sudo su - komutu** hem root yetkisi verir hem de env.var.'ları değiştirir. Tamamen root oluruz. Home klasörü de /root olarak gelir.

```
[root@ip-172-31-88-75 home]# sudo su -  
Last login: Mon Jan 16 14:35:20 UTC 2023 on pts/0  
[root@ip-172-31-88-75 ~]# pwd  
/root
```

Buradan çıkmak için exit ile çıkışın. Benzer şekilde kullanıcı değiştireceğin zaman o kullanıcın tüm özelliklerini kullnmak istersen **sudo su - username** kullan.

## LINUX USER COMMAND

**whoami:** O anda bağlığınız hesabın kullanıcı adını söyler.

**who:** O anda sisteme bağlı olan kullanıcıları gösterir.

**w:** Kimin oturum açtığını, ne kadar bağlılar, hangi işlemleri yaptılar, ne kadar CPU tüketiyor gibi bilgileri gösterir.

**id [Username]:** O kullanıcının id'sini verir.

## USER MANAGEMENT

Linux'ta kullanıcı bilgileri /etc/passwd klasöründe saklanır.

```
clarusway@DESKTOP-UN6T2ES:~$ tail -5 /etc/passwd
clarusway:x:1000:1000:,,,:/home/clarusway:/bin/bash
james:x:1001:1001::/home/james:/bin/sh
john:x:1002:1005:john,room,work,home,other:/home/john:/bin/bash
oliver:x:1003:1005:oliver,room_1,work_1,home_1:/home/oliver:/bin
    /bash
aaron:x:1004:1006:aaron,room_2,work_2,home_2:/home/aaron:/bin/bash
clarusway@DESKTOP-UN6T2ES:~$
```

**useradd [Username]:** Yeni bir kullanıcı oluşturmak için gereken komut

- -m : Bu oluşunca kullanıcıya home directory atar
- -d : Bu directory ye isim atamamıza yarar.
- -c : Bu kullanıcıya belirli bir açıklama atayabiliyoruz.

Aynı şekilde **adduser** komutu da var daha pratik

**userdel [username]:** userdel sadece kullanıcıyı siler. Ancak home klasörü durur. Eğer userdel -r ile silersek komple siler.

**Usermod –[option][value][username]** Kullanıcıların özelliklerini değiştirmek için kullanılır. Psswrd, grup ID, home deirectory, sistemi kilitleme vs. birçok işlem yapılabiliyor.

```
root@DESKTOP-UN6T2ES:~# usermod -a -G linux james
root@DESKTOP-UN6T2ES:~# usermod -a -G linux aaron
```

gruba kullanıcı ekler

**passwd [username]:** kullanıcıya parola atar.

/etc/login.defs klasörü şifre değiştirme işlemleri için kaç gün kaldı ne zaman uyaralım bilgilerini saklıyor.  
/etc/shadow: şifrelerin saklandığı klasör.  
/etc/passwd: Kullanıcıların ve özelliklerinin saklandığı dosya  
/etc/group: grup özelliklerinin saklandığı dosya

**groups [username]:** Var olan grupları listeler

**groupadd [username]:** Yeni grup oluşturur

**usermod -a -G: [group name][user name]** Gruba yeni kullanıcı ekler

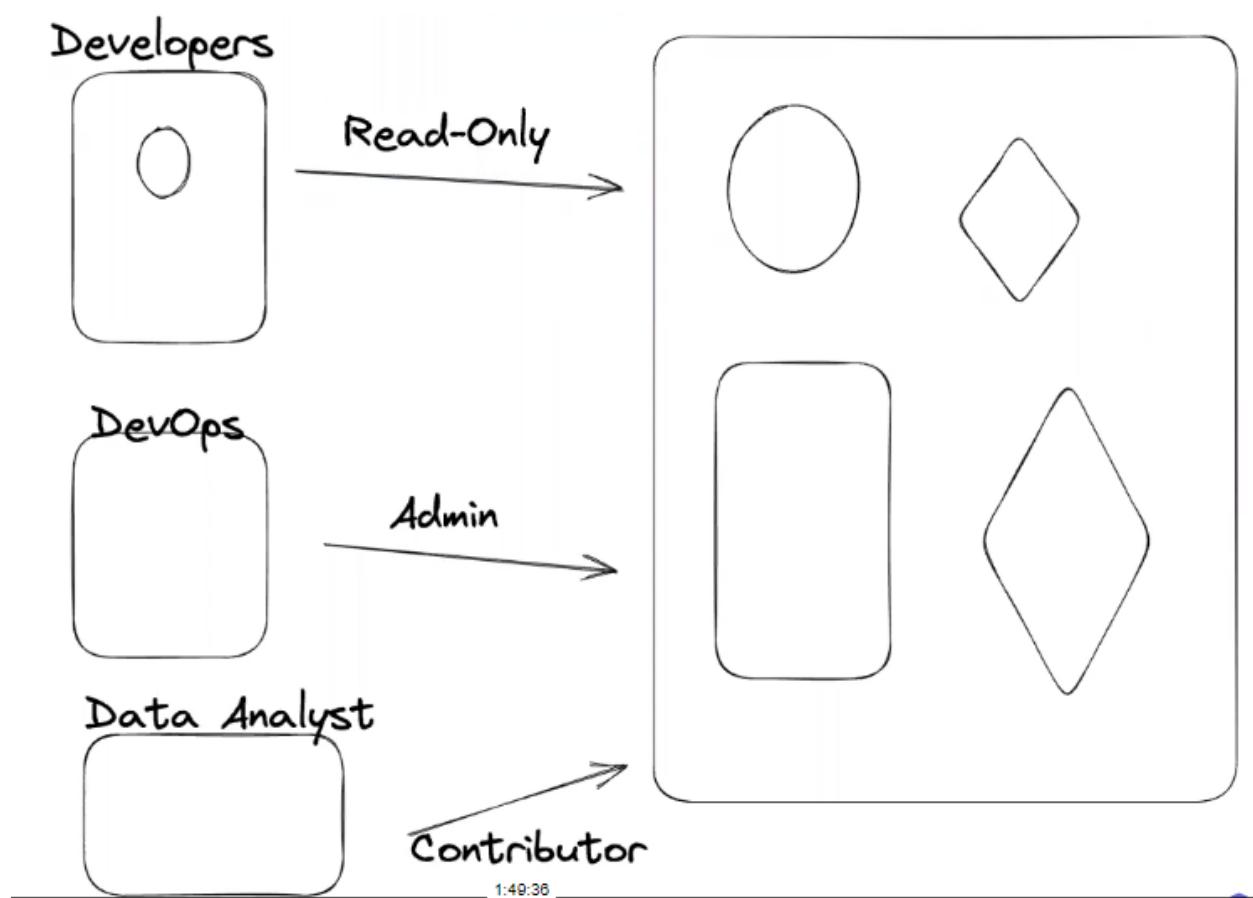
**gpasswd -a/-d [username][groupname]:** -a gruba yeni kullanıcı ekler. -d siler.

**groupmod -n [newname][oldname]:** Var olan grup ismini değiştirir

**groupdel [groupname]:** Var olan grupları siler.

## RBAC (Role Based Access)

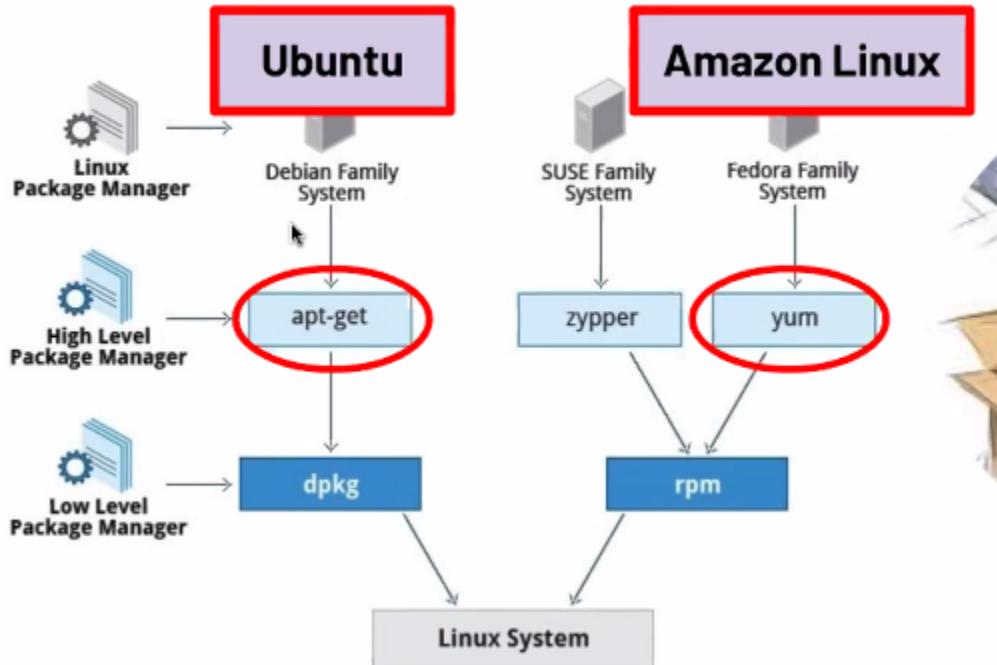
Şirketteki her kullanıcıya ayrı ayrı yetki vermek çok uzun ve meşakkatli bir işlem olur. Özellikle çok sayıda personeli olan şirketlerde. Bunun yerine belirli gruplar belirlenir Developers, Data Analyst gibi. Roller (Read-Only, Admin vs.) bu grplara verilir. Şirkete yeni bir personel geldiğinde ilgili gruba dahil olur ve otomatik olarak gerekli yetkilere sahip olur. Bu sistemin AWS'deki hali IAM Role



**USER Package Management:** Telefonlardaki AppStore ve PlayStore gibi Linux'ta da - yum -apt vs. var

# Package Management

Operating System	Format	Tool(s)
Debian	.deb	apt, apt-cache, apt-get, dpkg
Ubuntu	.deb	apt, apt-cache, apt-get, dpkg
CentOS	.rpm	yum
Fedora	.rpm	dnf
FreeBSD	Ports, .txz	make, pkg



Yarı otomatik toolları yüklerken aşağıdakileri kullanıyoruz. Paket ismini yazarak. Debian package manager

---

```
$ dpkg -i [package-name]      # Installing a package  
$ dpkg -r [package-name]      #Removing a package  
$ dpkg -l                      # Lists installed packages
```

---

APT ve YUM ile paket ismini yazmadan da tüm programları güncelle seçeneği mevcut

APT (Advanced Package Tool)

```
$ apt update                  # Update the installed packages  
$ apt install [package-name]  # Install a package and all its dependencies  
$ apt remove [package-name]   # Remove a package  
$ apt purge [package-name]    # Remove a package and its configuration files
```

apt install tree: Tree komutunu siler. Remove paketi siler, ancak configuration dosyaları saklanır. purge ise komple siler

YUM (Yellowdog Updater Modified)

---

```
$ yum install [package-name]  # Install a package  
$ yum remove [package-name]   # Remove a package  
$ yum update [package-name]   # Update a package
```

---

Bazı toollar apt ile indiriliyor. O zaman da o toolu koyan adamlar github adresini paylaşıyor. wget ile indirip istediğimiz klasörün altına atarak kullanıyoruz.

```

$ yum install [package-name]          # Install a package
$ yum -y install [package-name]        # Skip confirmations during installation
$ yum remove [package-name]           # Remove a package.
$ yum erase [package-name]            # Remove a package (an alias to remove).
$ yum autoremove [package-name]       # Remove a package and unused dependencies.
$ yum update [package-name]           # Update a package
$ yum update                         # Update all installed packages
$ yum info [package-name]             # Get information about a package
$ yum list                            # List all installed and available packages
$ yum list [package-name]              # List available matching package(s)
$ yum list installed                 # List installed packages
$ yum --showduplicates list [package-name] # Lists all available versions
$ yum install [package-name]-[version] # Install a specific version

```

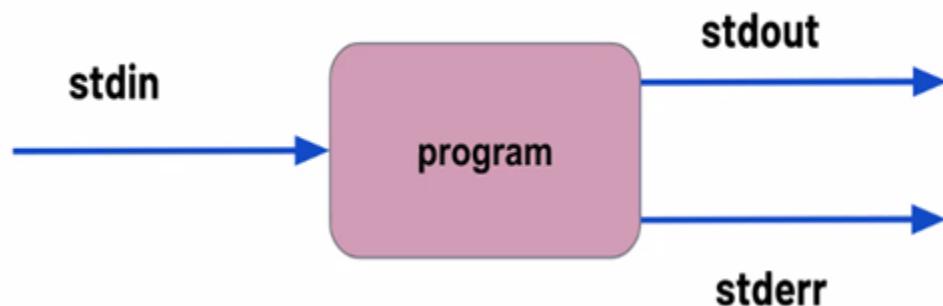
- Herhangi bir uygulamasının versiyonunu kontrol etmek için --version diyoruz  
git --version
- `sudo yum list installed` Yüklü paketleri gösterir.
- `sudo yum --showduplicates list` Yüklenebilecek paketleri gösterir.
- `sudo yum --showduplicates list git` Yüklenebilecek git paketlerini gösterir.

**yum-config-manager:** Paket yöneticisinin yapılandırma dosyalarını yönetmek için kullanılır. Bu komutu kullanarak, yeni bir Yum reposu ekleyip, silebiliriz. Örneğin;

```
sudo yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo
```

⇒ Komutu ile docker reposunu makinaya indiriyoruz.

## FILTERS and CONTROL OPERATORS



Örneğin cd komutunu kullandığımızda ulaşmaya çalıştığımız klasör (cd folder1) stdin'dir. Eğer komutta bir sıkıntı yoksa yani öyle bir klasör varsa komutun çıktısı stdout'dur. Eğer hata varsa bu hata da stderr'den gelir.

Stdin: StandartInput

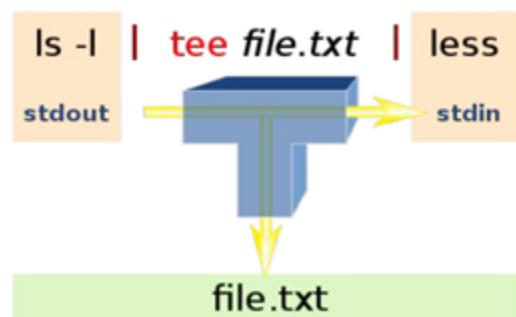
Stdout: StandartOutput

Stderr: StandartError

- cat
- tee
- grep
- cut
- tr
- wc
- sort
- uniq
- comm
- sed
- awk
- diff

**cat:** Dosya içeriğini görüntüler

**tee:** Bu komut kendisine gelen dosyayı/ komutu yanına yazılan dosyaya kaydeder ve yandaki komuta iletir



**grep:** arama komutu

grep ile yukarıda içeriği verilen sample.txt dosyasında arama yaparsak;

```
1 unique7
2 unique6
3 unique
4 unique8
5 unique1
6 unique5
7 unique2
8 unique3
9 unique4
10 unique9
11 duplicate
12 duplicate
13 Average Linux U
14 SUBSCRIBE
15 duplicate
16 YouTube
17 duplicate
18 duplicate
19 duplicate
20 duplicate
```

```
grep Linux sample.txt
>Avarege Linux User
grep unique sample.txt
>unique7
unique6
unique
unique8
unique1
unique5
unique2
unique3
unique4
unique9
grep -w unique sample.txt
>unique
grep -A 3 Linux sample.txt => İçinde Linux geçen ve ondan sonraki 3 satırı yazdırır
>Avarage Linux User
SUBSCRIBE
duplicate
```

```
Youtube
grep -B 3 Linux sample.txt => İçinde Linux geçen ve ondan önceki 3 satırı yazdırır
>unique9
duplicate
duplicate
Avarage Linux User
grep -c unique sample.txt => unique ifadesinin metinde kaç kez geçtiğini sayar
>10
grep -wc unique sample.txt => unique kelimesinin metinde kaç kez geçtiğini sayar
>1
grep -v Linux sample.txt => İçinde Linux yazan satır hariç diğerlerini ekrana yazar
```

**cut:** bir tablonun belirli bir sütünunu almak istiyorsak bu komutu kullanuyoruz

```
cut -d(delimiter) -f(columnNumber) <fileName>
```

**delimiter** sütunların nasıl ayırdığını giriyoruz (boşluk, nokta, virgül vs)

**tr:** translate, Bu komutla yazılan karakterlerin yerine istediğimiz bir karakteri girer ya da karakterleri silebiliriz.

```
cat clarusway.txt
Way To Reinvent Yourself
cat clarusway.txt | tr -d e => e harfini siler
Way To Rinvnt Yourslf
cat clarusway.txt | tr 'aer' 'iou' => a ile i, e ile o, r ile u yu değiştirir
Wiy To Roinvont Youusolf
cat clarusway.txt | tr [a-z] [A-Z] => Büyük/Küçük harf
WAY TO REINVENT YOURSELF
cat clarusway.txt | tr [A-Z] [a-z] => Büyük/Küçük harf
way to reinvent yourself
cat clarusway.txt | tr [:space:] '\t' => space yerine tab atar
WAY      TO      REINVENT      YOURSELF
```

```
ubuntu@clarusway:~$ cat count.txt
one
two
three
four
five
ubuntu@clarusway:~$ cat count.txt | tr '\n' ' '
one two three four five ubuntu@clarusway:~$
```

 satırsonunda görünmez bir \n işaretini var alt satıra geçmek için. tr komutunda \n karakteri yerine boşluk koyarsak satırlar alt alta değil yan yana yazılır.

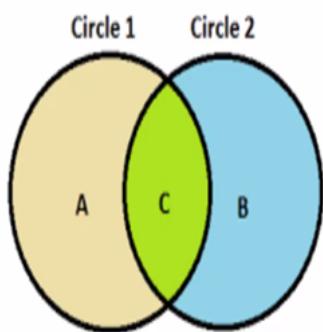
**wc:** Word count, sırasıyla dosyanın kaç satır, kaç kelime ve kaç karakterden oluşupunu gösteriyor. Aşağıdaki optionlarla bunlardan sadece birini görmek istiyorsak kullanıyoruz

- **wc <fileName>** (Counts words, lines and characters)
- **wc -l <fileName>** (Counts only lines)
- **wc -w <fileName>** (Counts only words)
- **wc -c <fileName>** (Counts only characters)

**sort:** alfabetik sırayla yazıyor. -r ile tersten yazar. -f ile büyük küçük duyarlılığı kaldırır. -n sayılarla göre sıralar.

**uniq:** Tekrar edenleri siler. **uniq -d** ise tekrar edenleri gösterir. Bu komutu kullanmadan önce dosya mutlaka sort ile sıralanmalıdır.

**comm:** (Compare) İki ayrı dosyadan ortak olan değerleri bulur. Üç sütun olarak verir sonucu. A birinci sütun, B ikinci sütun C de üçüncü sütun olur



```
ubuntu@clarusway:~$ cat file1
Aaron
James
John
Oliver
Walter
ubuntu@clarusway:~$ cat file2
Guile
James
John
Raymond
ubuntu@clarusway:~$ comm file1 file2
Aaron
          Guile
          James
          John
          Oliver
          Raymond
          Walter
ubuntu@clarusway:~$
```

**sed:** Metin içindeki belirli bir kelime grubunu, cümleyi, paragrafi alıp yerine istediğimiz herhangi bir şey yazabilirim. Stream editor gibi çalışır. On the fly çalışır da diyebiliriz. Yani yapılan işlemi herhangi bir yerde tutmaz sadece memoryde tutar ve sadece konsolda gösterir. Dosya üzerinde herhangi bir değişiklik yapmaz. Yaptığımız işlemlerin nihai halini farklı bir dosyaya yazdırmanız gerekiyor.

```
```sed.txt
Linux is an OS. Linux is life. Linux is a concept.
I like linux. You like linux. Everyone likes linux.
Linux is free. Linux is good. Linux is hope.
```

s ⇒ substitution
sed 's/linux/ubuntu/' sed.txt ⇒ yazılırsa ilk karşılaşışılinux yerine ubuntu yazar.
sed 's/linux/ubuntu/2ig' sed.txt ⇒ g: global yani tüm metinde düzeltir.
          i: B/K harfe алдірма
          2: ikinci eşleşenden itibaren değiştir
sed '2 s/linux/ubuntu/' sed.txt ⇒ ikinci satırda değişiklikleri yapar.
```

**awk:** Tablolarla çalışırken etkili bir komut. awk '{ }' file.txt şeklinde formatı vardır. Süslü parantez içine fonksiyonlar yazılabilir. awk '{print \$11}' ⇒ 11. sütunu getiriyor. Hiçbir delimiter girmezsek awk komutu boşluğu ayraç olarak kabul eder.

```
```awk.txt
This is line 1
This is line 2
This is line 3
This is line 4
This is line 5
```

awk '{print $1}' awk.txt ⇒ 1. sütunu yazdırır. $0 tüm sütunları, $n ilgili satırı yazdırır.
awk '{print $4,$2,$3,$1}' awk.txt ⇒ sütun yerlerini değiştirir.
awk -F: '{print $2}' awk.txt ⇒ ayraç olarak : kullanılır
awk '{ if($4 == "3") print $0}' awk.txt ⇒ 4. sütunda 3 olan satırı bul tamamını yazdır.
awk '{print $2,$4}' awk.txt => 2 ve 4. sütunu yazdırır. Virgülü kaldırırsak bitişik yazar.
```

**diff:** İki dosyayı satır satır karşılaştırır.

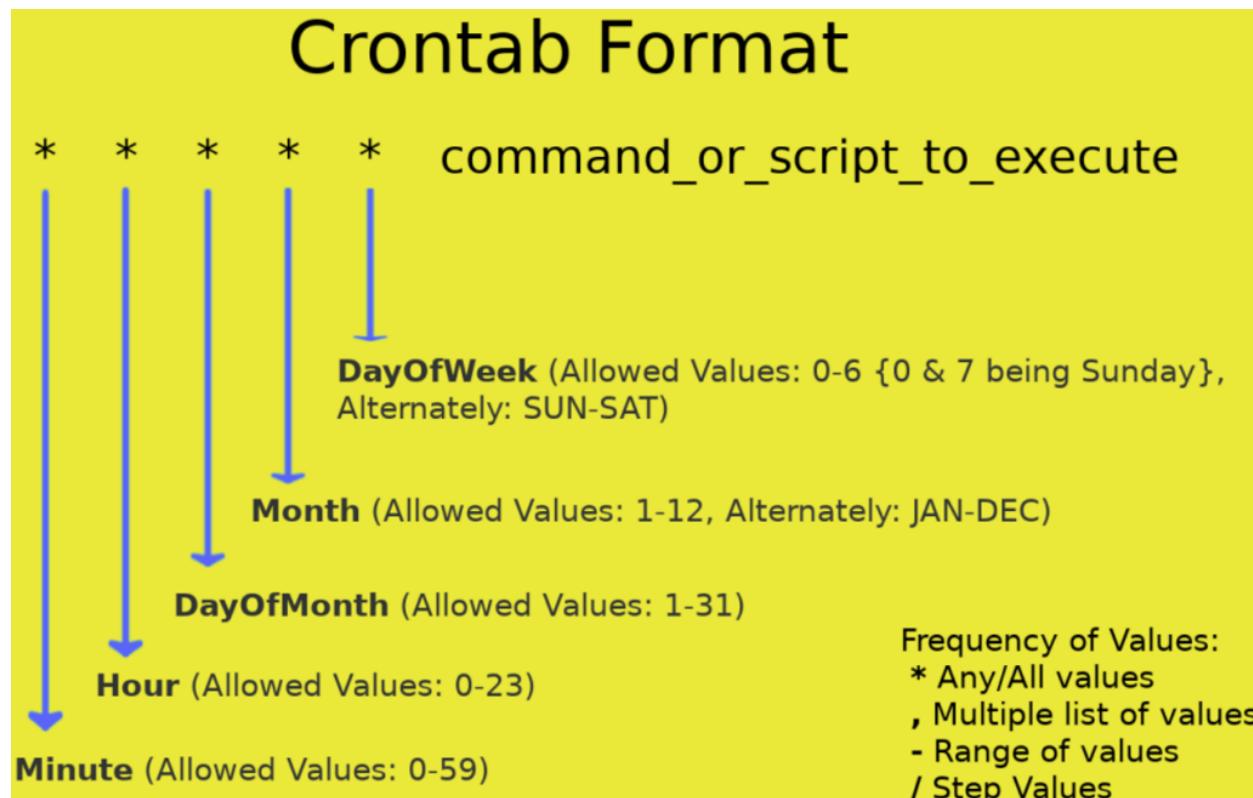
\$ diff <options> file1 file2  
-a dosyaya eklenenleri, -c değişiklikleri, -d silinenleri gösterir.

## Crontab

Linux sisteme üzerinde otomasyonlar kurmamıza yarar.

crontab -e yeni bir crontab düzenlemeye -l var olanları listelemeye yarar.

<https://crontab.guru/> ⇒ \*\*\*\*\* kısmı ile ilgili bilgiler var.



## Control Operators

| Control Operator         | Usage   |
|--------------------------|---|
| ; semicolon              | More than one command can be used in a single line.             |
| & ampersand              | Command ends with & and doesn't wait for the command to finish. |
| \$? dollar question mark | Used to store exit code of the previous command.                |
| && double ampersand      | Used as logical AND.  |
| double vertical bar      | Used as logical OR.   |
| Combining && and         | Used to write if then else structure in the command line.       |
| # pound sign             | Anything written after # will be ignored.                       |

; Birden fazla komutu arka arkaya çalıştırır  
& Çalışan işlemi arka plana atar. Örneğin güncelleme yaptığı zaman tüm dosyaların teker teker yüklenmesini göstermez bize arka planda devam eder işleme Linux konsolunu bize bırakır yazı yazabiliriz

```
user@clarusway:~$ sleep 20 &
[1] 14122
user@clarusway:~$ 
...wait 20 seconds...
user@clarusway:~$ 
[1]+          Done                  sleep 20
```

\$? Yazılığımız bir önceki komutun sonucunu gösterir. rakam şeklinde verir bunu. 0 çalıştı, sıfır dışında bir sayı gelirse çalışmadı.

```
user@clarusway~$ touch file1
user@clarusway~$ echo $?
0
```

&& logic olarak AND anlamı vardır. Yazılan 2. Komutun çalışması için ilk komutun da çalışması gerek

```
@ ubuntu@ip-172-31-92-42:~$ cat kountries.csv && sudo apt update -y
cat: kountries.csv: No such file or directory
```

Kountries diye bir klasör olmadığı için 2.si çalışmadı.

|| Logic olarak OR anlamı var. İlkci çalışmazsa ikincisi çalışır. İlkci çalışırsa 2.si çalışmaz.

```
$ cat kountries.csv || sudo apt update -y
```

Yukarıda ilki çalışmadığı için 2.si çalışır.

; de böyle bir şey yok ne olursa olsun 2 komut da çalışır

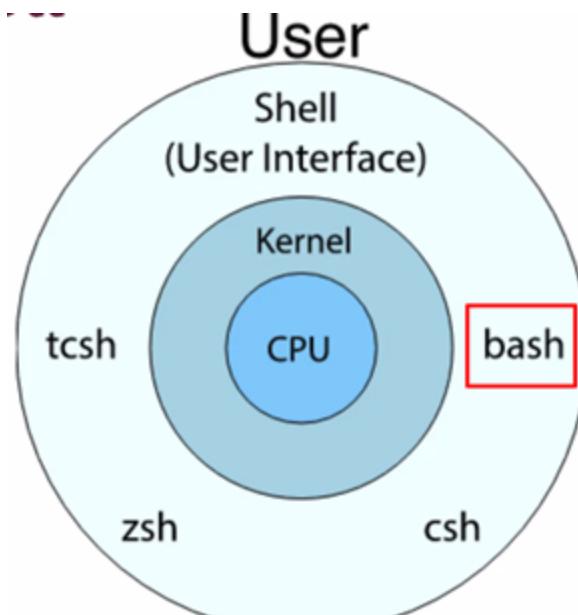
# yorum yazmaya yarar kodu etkilemez

\ Ters slash, özel karakterlerin özel gücünü elinden alır. Ör. \# yorum yazdırmaz sadece # yazdırır. Satır sonunda alt tarafa geçmek için kullanılır. Komut hala devam eder

```
ubuntu@clarusway:~$ echo This command line \
> > is split in three \
> > parts
ubuntu@clarusway:~$ This command line is split in three parts
```

## SHELL SCRIPT

Shell script için bir dosya kullanılır. Bunun içine istediğimiz eylemleri (log kayıtları alma, yeni makine kaldırma vs.) içine kaydedip otomatik olarak çalıştırabiliyoruz.



### Shell Tipleri

**sh:** Bourne Shell

**bash:** Bourne Again shell

**csh:** C shell

**tcsh:** TENEX C shell

**ksh:** Korn shell

Dosya uzantısı .sh tır. Zorunlu değil ama öyle kabul görmüş

Birçok Shell derleyicisi var. Genellikle BASH kullanıyoruz. Kodun başında hangi derleyici ile yazıldığı belirtilir. **#!/bin/bash => BASH kodu**

Dosyayı yazdıktan sonra bu dosyaya chmod +x ile executable yetki vermemiz gereklidir.

Script'i çalıştırmak için **./(script dosyasının adı)** yazılır.

**./** yazmadan da çağrılabılır. Bunun için \$PATH'e (dosya adını) eklemek lazımdır.

## Brace Expansion

- Shell'de birden fazla dosya süslü parantez ile aynı anda açılabilir.

```
[ec2-user@ip-172-31-48-164 ~]$ cat a.txt  
selamün aleyküm  
[ec2-user@ip-172-31-48-164 ~]$ cat b.txt  
aleyküm selam  
[ec2-user@ip-172-31-48-164 ~]$ cat {a,b}.txt  
selamün aleyküm  
aleyküm selam
```

- Ya da ardışık sayıları .. ile yazdırabiliriz.

```
[ec2-user@ip-172-31-48-164 ~]$ echo {1..5}  
1 2 3 4 5
```

- Verilen bir listedeki elemanlara aynı anda ekleme yapabiliriz

```
$ echo {I,want,my,money,back}  
I want my money back  
  
$ echo _{I,want,my,money,back}  
_I _want _my _money _back  
  
$ echo {I,want,my,money,back}_  
I_ want_ my_ money_ back_  
  
$ echo _{I,want,my,money,back}-  
_I- _want- _my- _money- _back-
```

## Command Substitution

Yazdığımız bir komutu bir değişkene atayabiliyoruz. Bunun için **VARIABLE=\$(command)** kullanılır.

Bir diğer yöntem backtick denilen (` `) işaretini kullanılır. **VARIABLE=`command`**

```
content=$(ls)
echo $content
```

content=`ls`
echo $content
```

Aynı şekilde bunları bir string ifadenin içine de yazabiliriz.

```
home > ec2-user > my-first-script.sh
1  #!/bin/bash
2
3  echo "Welcome, your working directory is $(pwd)."
4  echo "Today is `date`"
5  echo "You are `whoami`"

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS
● [ec2-user@komputer ~]$ my-first-script.sh
Welcome, your working directory is /home/ec2-user.
Today is Sat Feb 18 03:38:39 UTC 2023
You are ec2-user
```

Ya da direkt olarak `` içine yazığımız komutu değişkene atamadan da kullanabiliriz.

```
[ec2-user@maven ~]$ echo `pwd`
/home/ec2-user
```

Aynı şekilde \$ ile de

## Liste Belirleme

Liste elemanlarını tek tek verip sonradan çağırabiliriz. Değişkenleri teker teker yazacaksak her biri köşeli parantez ile yazılmalı

```
DISTROS[0]="ubuntu"
DISTROS[1]="fedora"
DISTROS[2]="debian"
DISTROS[3]="centos"
DISTROS[4]="alpine"
```

Ya da tek seferde verebiliriz. Tek bir liste halinde değişkenleri vereceksek parantezlere dikkat etmeliyiz.

```
devops_tools=("docker" "kubernetes" "ansible" "terraform" "jenkins")
```

Elemanları çağırırken de;

```
echo ${DISTROS[0]}
echo ${DISTROS[1]}
...
echo ${devops_tools[@]}
```

### Heredoc (Here Document)

Bu bash'te çoklu satır yazmaya izin veren bir komut. Aşağıdaki gibi bir formatı vardır.

```
[COMMAND] <<[ - ] 'DELIMITER'
          HERE-DOCUMENT
          DELIMITER
```

[COMMAND] kısmına çalıştırılacak istenen komut yazılır. <<'dan sonra gelen karakter/karakterler metnin bittiğini gösteren ayraçtır. Ör:

```
cat << EOF
The current working directory is: $PWD
You are logged in as: $(whoami)
EOF # End of File
```

## IF STATEMENTS

if..then..else

```
if [[ <some test> ]]
then
  <commands>
fi
```

```
if [[ <some test> ]]
then
  <commands>
else
  <other commands>
fi
```

```
if [[ <some test> ]]
then
  <commands>
elif [[ <some test> ]]
then
  <different commands>
else
  <other commands>
fi
```

## COMPARISONS

[ ] ⇒ İçindeki ifadenin sonucuna göre (0) doğru, (1) yanlış sonuç veren bir ifadedir.  
Örneğin; .sh dosyasının içine aşağıdakileri yazıp çalıştırırsak sonuç 0 olur. Çünkü [ ] içindeki mantıksal ifade doğru.

```
[ 100 -eq 100 ]
echo $?
```

Karşılaştırma komutları;

Command	Operation
-lt	<
-gt	>
-le	<=
-ge	>=
-eq	==
-ne	!=

```
#!/bin/bash
read -p "Input a number" number

if [[ $number -gt 50 ]]
then
  echo "The number is big."
fi
```

String Operatos	
Command	Operation
=	equal

<b>!=</b>	not equal
<b>-z</b>	Empty String
<b>-n</b>	Not Empty String

```
#!/bin/bash

if [[ "a" = "a" ]]
then
    echo "They are same"
fi

if [[ "a" != "b" ]]
then
    echo "They are not same"
fi

if [[ -z "" ]]
then
    echo "It is empty"
fi

if [[ -n "text" ]]
then
    echo "It is not empty"
fi
```

Operator	Description
<b>-d file</b>	directory
<b>-e file</b>	exists
<b>-f file</b>	ordinary file
<b>-r file</b>	readable
<b>-s file</b>	size is > 0 bytes
<b>-w file</b>	writable
<b>-x file</b>	executable

```

#!/bin/bash

if [[ -d folder ]]
then
    echo "folder is a directory"
fi

if [[ -f file ]]
then
    echo "file is an ordinary file"
fi

if [[ -w file ]]
then
    echo "file is a writable file"
fi

if [[ -s file ]]
then
    echo "file is > 0 bytes"
fi

```

```

#!/bin/bash
clear
echo "enter a number"
read st1
if (( $st1 == 5 ))
then
    echo "You entered five!"
fi
# -eq -gt -lt -ge -le -a -o
if [ $st1 -eq 5 ]
then
    echo equal to five
elif (( $st1 < 5 ))
then
    echo "less than five"
else
    echo "more than five"
fi

```

```

#!/bin/bash

if [[ "abc" = "abc" ]]

then

echo "They are same"

fi

#!/bin/bash

read -p "Input a number" number

if [[ $number -gt 10 ]] # burada alınacak değerler true ya da false
then

echo "$number is bigger than 10"

fi

```

### Pseudo Code (Sözde Kod)

Yazılacak programın algoritmasını rahat bir şekilde görebilmek için kaba taslak yazılan kod bloğuna denir.

```

1  #!/bin/bash
2
3  read -p "Enter any number: " NUMBER
4
5  if [[ _ ]] # -gt 10
6  then
7      # odd & even
8      if [[ _ ]] # %2 test
9      then
10         # odd
11     else
12         # even
13     fi
14 else
15     # smaller than 10
16 fi

```

# LOOPS

## While loops

while döngüsü içindeki mantıksal ifade doğru olduğu sürece döner.

```
while [[ <some test> ]]
do
  <commands>
done
```

```
#!/bin/bash

number=1

while [[ $number -le 10 ]]
do
  echo $number
  ((number++))
done
echo "Now, number is $number"
```

```
./while-loops.sh
1
2
3
4
5
6
7
8
9
10
Now, number is 11
```

## Until Loops

Until döngüsü içindeki değer yanlış olduğu sürece içindeki mantıksal döngü çalışır. Yani içindeki ifade sağlanana kadar çalışır diyebiliriz.

```
until [[ <some test> ]]
do
  <commands>
done
```

```
#!/bin/bash
1
2
3
4
5
6
7
8
9
Now, number is 10
```

```
#!/bin/bash

number=1

until [[ $number -ge 10 ]]
do
  echo $number
  ((number++))
done
echo "Now, number is $number"
```

## For Loops

Bir işlemi bir listenin tüm elemanlarına uygulamak için kullanılır. [[ ]] kullanmamıza gerek yok.

**for number in {1..10}** şeklinde de yazılabilir.

```
for item in [list]
do
    commands
done
```

```
#!/bin/sh

echo "Numbers:"

for number in 0 1 2 3 4 5 6 7 8 9
do
    echo $number
done
```

```
./for-loop.sh
Numbers:
0
1
2
3
4
5
6
7
8
9
```

## Infinite Loop

### Break Statement

```
effective.xml log.txt pom.xml src target
ec2-user@ip-172-31-88-113# maven-experiment:$ echo `pwd`
/home/ec2-user/maven-experiment
ec2-user@ip-172-31-88-113# maven-experiment:$ echo $(pwd)
/home/ec2-user/maven-experiment
ec2-user@ip-172-31-88-113# maven-experiment:$ █
```