

## MENGGUNAKAN COMMAND LINE

3

### OBJEKTIF :

1. Mahasiswa Mampu Memahami Command Line Interface
  2. Mahasiswa Dapat Mengenal Shell, Commands, Variables, dan Command Type
  3. Mahasiswa Mampu Menggunakan Quoting dan Control Statements
- 

### PENDAHULUAN

Sebagian besar *consumer* sistem operasi dirancang untuk melindungi pengguna dari seluk beluk CLI. Komunitas Linux berbeda dari CLI karena kekuatan, kecepatan, dan kemampuannya untuk menyelesaikan beragam tugas dengan satu instruksi baris perintah.

Ketika pengguna pertama kali bertemu CLI, mereka berpikir itu menantang karena dibutuhkan untuk menghafal sejumlah perintah yang membingungkan dan pilihan mereka. Namun, begitu pengguna mempelajari struktur *command* yang digunakan, di mana *file* dan direktori yang diperlukan berada dan bagaimana menavigasi hierarki sistem *file*, mereka bisa sangat produktif. Kemampuan ini memberikan kontrol yang lebih tepat, kecepatan lebih besar dan kemampuan untuk mengotomatisasi tugas-tugas dengan lebih mudah melalui *script*.

Selain itu, dengan mempelajari CLI, pengguna dapat dengan mudah menjadi produktif hampir seketika setiap distro atau distribusi Linux apapun, mengurangi jumlah waktu yang dibutuhkan untuk membiasakan diri dengan sistem karena variasi dalam GUI.

Copyright © 2019 Network Development Group Inc.

# Why is knowing the command line important?



By understanding the foundation of Linux, you have the ability to work on ANY Linux distribution. This could mean one company with a mixed environment or a new company with a different Linux distribution.

## 3.1 SHELL

Setelah pengguna memasukkan perintah, terminal kemudian menerima apa yang telah diketik pengguna dan meneruskannya ke *shell*. *Shell* adalah interpreter *command line* yang menerjemahkan *command* yang dimasukkan oleh pengguna menjadi tindakan yang harus dilakukan oleh sistem operasi. Jika output dihasilkan oleh *command*, maka teks ditampilkan di terminal. Jika masalah dengan *command* ditemukan, pesan kesalahan akan ditampilkan.

Lingkungan Linux memungkinkan penggunaan banyak *shell* yang berbeda, beberapa di antaranya telah ada selama bertahun-tahun. *Shell* yang paling umum digunakan untuk distribusi Linux disebut **Bash shell**. Bash menyediakan banyak fitur canggih, seperti *command history* dan *inline editing*, yang memungkinkan pengguna dengan mudah menjalankan kembali peintah yang sebelumnya dieksekusi atau variasi melalui pengeditan sederhana.

*Shell* Bash juga memiliki fitur populer lainnya, beberapa di antaranya tercantum di bawah ini:

- **Scripting:** Kemampuan untuk menempatkan *command* dalam *file* dan kemudian menafsirkan (secara efektif menggunakan Bash untuk mengeksekusi isi) *file*, menghasilkan semua *command* yang dieksekusi. Fitur ini juga memiliki beberapa fitur pemrograman, seperti pernyataan kondisional dan kemampuan untuk membuat fungsi.
- **Aliases:** Kemampuan untuk membuat nama panggilan pendek untuk *command* yang panjang.
- **Variables:** Digunakan untuk menyimpan informasi untuk Bash *Shell* dan untuk pengguna. Variabel-variabel ini dapat digunakan untuk memodifikasi cara kerja *command* dan fitur serta memberikan informasi sistem yang vital.

Ketika aplikasi terminal dijalankan, dan sebuah *shell* muncul, menampilkan bagian penting dari antarmuka — *prompt*. Tidak hanya *prompt* di sana untuk menunjukkan bahwa *command* dapat dijalankan, tetapi juga menyampaikan informasi yang berguna kepada pengguna. *Prompt* sepenuhnya dapat dikonfigurasi dan fitur lengkap seperti praktis dan berguna.

Struktur *prompt* dapat bervariasi antara distribusi, tetapi biasanya berisi informasi tentang pengguna dan sistem. Di bawah ini adalah struktur *prompt* yang umum:

```
sysadmin@localhost:~$
```

Prompt yang ditampilkan berisi informasi berikut:

- **User Name:**

```
sysadmin@localhost:~$
```

- **System Name:**

```
sysadmin@localhost:~$
```

- **Current Directory:**

```
sysadmin@localhost:~$
```

Simbol ~ digunakan sebagai singkatan untuk direktori *home* pengguna. Biasanya direktori *home* untuk pengguna berada di bawah direktori /home dan dinamai sesuai nama akun pengguna; misalnya, /home/sysadmin.

## 3.2 COMMANDS

Apa itu *command*? Jawaban paling sederhana adalah bahwa *command* adalah program perangkat lunak yang ketika dieksekusi pada CLI, melakukan tindakan pada komputer.

Untuk menjalankan *command*, langkah pertama adalah mengetik nama *command*. Klik di terminal di sebelah kanan. Ketik `ls` dan tekan **Enter**. Hasilnya di bawah ini:

```
sysadmin@localhost:~$ ls
Desktop Documents Downloads Music Pictures Public Templates Videos
```

**Catatan:** Dengan sendirinya, command `ls` mencantumkan file dan direktori yang ada di direktori kerja saat ini. Pada titik ini, Anda tidak perlu terlalu khawatir tentang output dari command, sebagai gantinya, fokuskan pada pemahaman bagaimana memformat dan menjalankan command.

Banyak *command* dapat digunakan sendiri tanpa *input* lebih lanjut. Beberapa *command* memerlukan *input* tambahan untuk berjalan dengan benar. *Input* tambahan ini datang dalam dua bentuk: *option* dan *argument*.

Format khas untuk suatu *command* adalah sebagai berikut:

```
command [options] [arguments]
```

*Option* digunakan untuk mengubah perilaku inti dari suatu *command* sementara *arguments* digunakan untuk memberikan informasi tambahan (seperti nama file atau nama pengguna). Setiap *option* dan *arguments* biasanya dipisahkan oleh spasi, meskipun *option* seringkali dapat digabungkan.

Perlu diingat bahwa Linux adalah case-sensitive. Command, option, argument, variable, dan file name harus dimasukkan persis seperti yang

### 3.2.1 ARGUMENTS

```
command [options] [arguments]
```

*Argument* dapat digunakan untuk menentukan sesuatu agar *command* dapat bertindak. Jika *command* `ls` diberi nama direktori sebagai *argumen*, akan menampilkan daftar isi direktori itu. Dalam contoh berikut, direktori `/etc/ppp` digunakan sebagai *argumen*; *output* yang dihasilkan adalah daftar file yang terdapat dalam direktori itu:

```
sysadmin@localhost:~$ ls /etc/ppp
ip-down.d  ip-up.d
```

*Command* `ls` juga dapat menerima banyak *arguments*. Untuk menampilkan daftar isi dari direktori `/etc/ppp` dan `/etc/ssh`, berikan keduanya sebagai argumen:

```
sysadmin@localhost:~$ ls /etc/ppp /etc/ssh
/etc/ppp:
ip-down.d  ip-up.d
/etc/ssh:
moduli      ssh_host_dsa_key.pub  ssh_host_rsa_key      sshd_configssh_d
ssh_host_ecdsa_key  ssh_host_rsa_key.pub
ssh_host_dsa_key    ssh_host_ecdsa_key.pub  ssh_import_id
```

### 3.2.2 OPTIONS

```
command [options] [arguments]
```

*Option* dapat digunakan dengan *command* untuk memperluas atau memodifikasi perilaku *command*. Misalnya, menggunakan *option -l* dari *command ls* menghasilkan *long listing*, memberikan informasi tambahan tentang file yang terdaftar, seperti izin, ukuran file dan informasi lainnya:

```
sysadmin@localhost:~$ ls -l
total 0
drwxr-xr-x 1 sysadmin sysadmin 0 Jan 29 20:13 Desktop
drwxr-xr-x 1 sysadmin sysadmin 0 Jan 29 20:13 Documents
drwxr-xr-x 1 sysadmin sysadmin 0 Jan 29 20:13 Downloads
drwxr-xr-x 1 sysadmin sysadmin 0 Jan 29 20:13 Music
drwxr-xr-x 1 sysadmin sysadmin 0 Jan 29 20:13 Pictures
drwxr-xr-x 1 sysadmin sysadmin 0 Jan 29 20:13 Public
drwxr-xr-x 1 sysadmin sysadmin 0 Jan 29 20:13 Templates
drwxr-xr-x 1 sysadmin sysadmin 0 Jan 29 20:13 Videos
```

Seringkali karakter dipilih untuk menjadi mnemonik (rumus atau singkatan) untuk tujuannya, seperti memilih huruf l untuk *long* atau r untuk *reverse*. Secara *default*, command *ls* mencetak hasil dalam urutan abjad, dan dengan menambahkan *option -r*, itu mencetak hasil dalam urutan abjad terbalik.

```
sysadmin@localhost:~$ ls -r
Videos Templates Public Pictures Music Downloads Documents Desktop
```

Dalam kebanyakan kasus, opsi dapat digunakan bersamaan dengan opsi lain. Mereka dapat diberikan sebagai opsi terpisah, seperti pada *-l -r*, atau gabungan, seperti *-lr*. Kombinasi dari dua opsi ini akan menghasilkan *output long listing* dalam urutan abjad terbalik:

```
sysadmin@localhost:~$ ls -lr
total 32
drwxr-xr-x 2 sysadmin sysadmin 4096 Oct 31 20:13 Videos
drwxr-xr-x 2 sysadmin sysadmin 4096 Oct 31 20:13 Templates
drwxr-xr-x 2 sysadmin sysadmin 4096 Oct 31 20:13 Public
drwxr-xr-x 2 sysadmin sysadmin 4096 Oct 31 20:13 Pictures
drwxr-xr-x 2 sysadmin sysadmin 4096 Oct 31 20:13 Music
drwxr-xr-x 2 sysadmin sysadmin 4096 Oct 31 20:13 Downloads
drwxr-xr-x 4 sysadmin sysadmin 4096 Oct 31 20:13 Documents
drwxr-xr-x 2 sysadmin sysadmin 4096 Oct 31 20:13 Desktop
```

Urutan opsi gabungan tidak penting. *Output* dari semua contoh ini akan sama:

```
ls -l -r  
ls -rl  
ls -lr
```

Secara *default option* `-l` dari *command* `ls` menampilkan ukuran file dalam byte:

```
sysadmin@localhost:~$ ls -l /usr/bin/perl  
-rwxr-xr-x 2 root root 10376 Feb 4 2018 /usr/bin/perl
```

Jika ditambahkan *option* `-h` ukuran file akan ditampilkan dalam format *human-readable*:

```
sysadmin@localhost:~$ ls -lh /usr/bin/perl  
-rwxr-xr-x 2 root root 11K Feb 4 2018 /usr/bin/perl
```

Opsi seringkali berupa huruf tunggal; namun, terkadang kata-kata atau frasa juga. Biasanya, perintah yang lebih lama menggunakan huruf tunggal sementara *command* yang lebih baru menggunakan kata lengkap untuk opsi. Opsi huruf tunggal didahului dengan tanda hubung tunggal karakter `-`, seperti *option* `-h`. Opsi kata lengkap diawali dengan dua tanda hubung karakter `--`. *Option* `-h` juga memiliki bentuk kata lengkap yang setara; *option* `--human-readable`.

```
sysadmin@localhost:~$ ls -l --human-readable /usr/bin/perl  
-rwxr-xr-x 2 root root 11K Feb 4 2018 /usr/bin/perl
```

### 3.2.3 HISTORY

Ketika sebuah *command* dieksekusi di terminal, maka akan disimpan dalam daftar *history*. Ini dilakukan untuk membuatnya lebih mudah dalam mengeksekusi *command* yang sama, sehingga tidak perlu untuk mengetik ulang seluruh *command*.

Menekan tombol **Up Arrow** ↑ menampilkan *command* sebelumnya pada baris *prompt*. Seluruh *history command* yang dijalankan di sesi saat ini dapat ditampilkan dengan menekan **Up** berulang kali untuk menelusuri kembali *history command* yang telah dijalankan. Menekan tombol Enter akan menampilkan *command* kembali.

Ketika *command* yang diinginkan ditemukan, tombol **Left Arrow** ← dan **Right Arrow** → dapat memposisikan kursor untuk diedit. Tombol ini berguna juga untuk mengedit termasuk tombol **Home**, **End**, **Backspace** dan **Delete**. Untuk melihat daftar *history* terminal, gunakan *command history*:

```
sysadmin@localhost:~$ date
Wed Dec 12 04:28:12 UTC 2018
sysadmin@localhost:~$ ls
Desktop Documents Downloads Music Pictures Public Templates Videos
sysadmin@localhost:~$ cal 5 2030
      May 2030
Su Mo Tu We Th Fr Sa
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
sysadmin@localhost:~$ history
  1  date
  2  ls
  3  cal 5 2030
  4  history
```

Jika *command* yang diinginkan ada dalam daftar yang menghasilkan *command history*, itu dapat dieksekusi dengan mengetikkan karakter tanda seru "!" dan kemudian nomor di sebelah *command*, misalnya, untuk menjalankan *command cal* lagi:

```
sysadmin@localhost:~$ history
  1  date
  2  ls
  3  cal 5 2030
  4  history
sysadmin@localhost:~$ !3
cal 5 2030
      May 2030
Su Mo Tu We Th Fr Sa
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
```

Jika *command history* ditambahkan angka sebagai argumen, itu mengeluarkan jumlah *command* sebelumnya dari daftar *history*. Misalnya, untuk menampilkan tiga *command* terakhir:

```
sysadmin@localhost:~$ history 3
  6  date
  7  ls /home
  8  history 3
```

Untuk menjalankan *command* nth dari bagian bawah daftar *history*, ketik `!-n` dan tekan Enter. Misalnya, untuk menjalankan command ketiga dari bawah daftar *history* lakukan berikut ini:

```
sysadmin@localhost:~$ !-3
date
Wed Dec 12 04:31:55 UTC 2018
```

Untuk menjalankan tipe *command* terbaru `!!` dan tekan **Enter**:

```
sysadmin@localhost:~$ date
Wed Dec 12 04:32:36 UTC 2018
sysadmin@localhost:~$ !!
date
Wed Dec 12 04:32:38 UTC 2018
```

Untuk menjalankan iterasi terbaru dari *command* tertentu, ketik `!` diikuti oleh nama *command* dan tekan **Enter**. Misalnya, untuk mengeksekusi *command* `ls` terbaru:

```
sysadmin@localhost:~$ !ls
ls /home
sysadmin
```

### 3.3 VARIABLES

Variabel adalah fitur yang memungkinkan pengguna atau *shell* untuk menyimpan data. Data ini dapat digunakan untuk memberikan informasi sistem yang kritis atau untuk mengubah perilaku cara kerja Bash *shell* (atau *command* lain). Variabel diberi nama dan disimpan sementara di memori. Ada dua jenis variabel yang digunakan dalam *shell* Bash: *local* dan *environment*.

#### 3.3.1 LOCAL VARIABLES

Variabel lokal atau variabel *shell*, hanya ada di *shell* saat ini, dan tidak dapat memengaruhi *command* atau aplikasi lain. Ketika pengguna menutup jendela terminal atau *shell*, semua variabel hilang. Mereka sering dikaitkan dengan tugas-tugas berbasis pengguna dan huruf kecil karena konvensi.

Untuk mengatur nilai variabel, gunakan ekspresi penugasan berikut. Jika variabel sudah ada, nilai variabel diubah. Jika nama variabel belum ada, *shell* membuat variabel lokal baru dan menetapkan nilainya:

```
variable=value
```

Contoh berikut membuat variabel lokal bernama variabell dan memberinya nilai Something:

```
sysadmin@localhost:~$ variable1='Something'
```

Perintah `echo` digunakan untuk menampilkan *output* di terminal. Untuk menampilkan nilai variabel, gunakan karakter tanda \$ diikuti oleh nama variabel sebagai *argumen* untuk *command echo*:

```
sysadmin@localhost:~$ echo $variable1  
Something
```

### 3.3.2 ENVIRONMENT VARIABLES

Variabel *environment*, juga disebut variabel global, tersedia di seluruh sistem, di semua *shell* yang digunakan oleh Bash ketika menafsirkan *command* dan melakukan tugas. Sistem secara otomatis membuat ulang variabel *environment* ketika shell baru dibuka. Contohnya termasuk variabel PATH, HOME, dan HISTSIZE. Variabel HISTSIZE menentukan berapa banyak *command* sebelumnya untuk menyimpan dalam daftar histori. *Command* dalam contoh di bawah ini menampilkan nilai variabel HISTSIZE:

```
sysadmin@localhost:~$ echo $HISTSIZE  
1000
```

Untuk memodifikasi nilai variabel yang ada, gunakan ekspresi penetapan (*assignment expression*):

```
sysadmin@localhost:~$ HISTSIZE=500  
sysadmin@localhost:~$ echo $HISTSIZE  
500
```

Ketika dijalankan tanpa *argument*, *command env* menampilkan daftar variabel *environment*. Karena *output* dari *command env* bisa sangat panjang, contoh berikut menggunakan pencarian teks untuk memfilter *output*.

Dalam contoh sebelumnya, variabel1 dibuat sebagai variabel *local*, jadi pencarian berikut dalam variabel *environment* menghasilkan tanpa *output*:

```
sysadmin@localhost:~$ env | grep variable1
```

Karakter pipe ‘|’ meneruskan output dari command *env* ke command *grep*, yang mencari output.

*Command export* digunakan untuk mengubah variabel lokal menjadi variabel *environment* (global).

```
export variable
```

Setelah mengekspor, variabel1 sekarang menjadi variabel *environment*. Sekarang ditemukan dalam pencarian melalui variabel *environment*:

```
sysadmin@localhost:~$ export variable1
sysadmin@localhost:~$ env | grep variable1
variable1=Something
```

*Command export* juga dapat digunakan untuk membuat variabel sebagai variabel *environment* saat pembuatannya dengan menggunakan ekspresi penetapan sebagai *argumen*:

```
sysadmin@localhost:~$ export variable2='Else'
sysadmin@localhost:~$ env | grep variable2
variable2=Else
```

Untuk mengubah nilai variabel *environment*, gunakan ekspresi penetapan:

```
sysadmin@localhost:~$ variable1=$variable1' '$variable2
sysadmin@localhost:~$ echo $variable1
Something Else
```

Variabel yang diekspor dapat dihapus menggunakan *command unset* :

```
sysadmin@localhost:~$ unset variable2
```

### 3.3.3 PATH VARIABLE

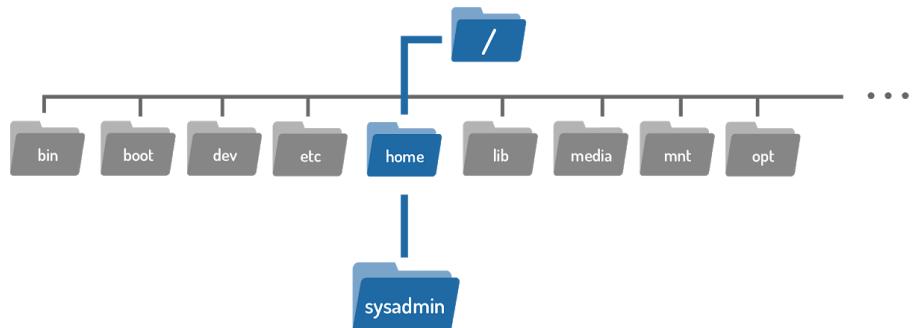
Salah satu variabel Bash *shell* yang paling penting untuk dipahami adalah variabel PATH. Ini berisi daftar yang mendefinisikan direktori mana *shell* mencari *command*. Jika *command* yang valid dimasukkan dan *shell* mengembalikan kesalahan "*command not found*", itu karena *shell* Bash tidak dapat menemukan *command* dengan nama itu di salah satu direktori yang termasuk dalam path. *Command* berikut menampilkan jalur *shell* saat ini:

```
sysadmin@localhost:~$ echo $PATH  
/home/sysadmin/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:  
sysadmin@localhost:~$
```

Setiap direktori dalam daftar dipisahkan oleh tanda titik dua ":". Berdasarkan *output* sebelumnya, path berisi direktori berikut. *Shell* akan memeriksa direktori sesuai urutannya:

```
/home/sysadmin/bin  
/usr/local/sbin  
/usr/local/bin  
/usr/sbin  
/usr/bin  
/sbin  
/bin  
/usr/games
```

Masing-masing direktori diwakili oleh *path*. Path adalah daftar direktori yang dipisahkan oleh karakter “/”. Jika Anda menganggap sistem file sebagai peta, path adalah alamat direktori, yang mencakup arah navigasi langkah demi langkah; mereka dapat digunakan untuk menunjukkan lokasi file apa pun dalam sistem file. Sebagai contoh, /home/sysadmin adalah path ke direktori home:



Jika *command* tidak ditemukan di direktori mana pun yang tercantum dalam variabel PATH, maka shell mengembalikan error:

```

sysadmin@localhost:~$ zed
-bash: zed: command not found
sysadmin@localhost:~$
```

Jika perangkat lunak khusus diinstal pada sistem, Anda mungkin perlu memodifikasi PATH untuk membuatnya lebih mudah untuk menjalankan perintah-perintah ini. Misalnya, yang berikut ini akan menambah dan memverifikasi direktori /usr/bin/custom ke variabel PATH:

```

sysadmin@localhost:~$ PATH=/usr/bin/custom:$PATH
sysadmin@localhost:~$ echo $PATH
/usr/bin/custom:/home/sysadmin/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/
```

Saat memperbarui variabel PATH, selalu sertakan jalur saat ini, agar tidak kehilangan akses ke command yang terletak di direktori tersebut. Ini dapat dilakukan dengan menambahkan \$PATH ke nilai dalam ekspresi penugasan. Ingatlah bahwa nama variabel yang didahului oleh tanda dolar mewakili nilai variabel.

### 3.4 COMMAND TYPES

Untuk mempelajari lebih lanjut tentang *command* adalah dengan melihat dari mana asalnya. *Command type* dapat digunakan untuk menentukan informasi mengenai jenis *command*.

```
type command
```

Ada beberapa sumber *command* yang berbeda di dalam *shell* CLI Anda termasuk *internal commands*, *external commands*, *aliases*, dan *functions*.

#### 3.4.1 INTERNAL COMMANDS

Disebut juga *built-in commands*, *internal commands* dibangun ke dalam *shell* itu sendiri. Contohnya *command cd* (*change directory*) yang merupakan bagian dari Bash *shell*. Saat pengguna mengetikkan *command cd*, Bash *shell* langsung mengeksekusi dan menafsirkannya, sehingga tidak memerlukan program tambahan.

*Command type* mengidentifikasi *command cd* sebagai *internal command*:

```
sysadmin@localhost:~$ type cd
cd is a shell builtin
```

#### 3.4.2 EXTERNAL COMMANDS

*External commands* disimpan dalam *file* yang dicari oleh *shell*. Jika pengguna mengetikkan *command ls*, maka *shell* mencari lewat direktori yang terdaftar dalam variabel PATH untuk menemukan *file ls* yang dapat dieksekusi.

Jika suatu *command* tidak sesuai seperti yang diharapkan atau tidak dapat diakses, akan bermanfaat untuk mengetahui di mana *command shell* atau

versi yang digunakan. Akan menjenuhkan jika harus mencarinya secara *manual* di setiap direktori yang tercantum dalam variabel PATH. Sebagai gantinya, gunakan *command which* untuk menampilkan *full path* ke *command* yang dimaksud:

```
which command
```

*Command which* mencari lokasi *command* dengan mencari variabel PATH.

```
sysadmin@localhost:~$ which ls  
/bin/ls  
sysadmin@localhost:~$ which cal  
/usr/bin/cal
```

*External commands* juga dapat dieksekusi dengan mengetikkan *full path* ke *command*. Misalnya, untuk menjalankan *command ls*:

```
sysadmin@localhost:~$ /bin/ls  
Desktop Documents Downloads Music Pictures Public Templates Videos
```

Untuk *external commands*, *command type* menampilkan lokasi *command*:

```
sysadmin@localhost:~$ type cal  
cal is /usr/bin/cal
```

Dalam beberapa kasus *output* dari *command type* berbeda secara signifikan dari *output command which*:

```
sysadmin@localhost:~$ type echo  
echo is a shell builtin  
sysadmin@localhost:~$ which echo  
/bin/echo
```

Menggunakan *option -a* dari *command type* menampilkan semua lokasi yang berisi *command* bernama:

```
sysadmin@localhost:~$ type -a echo  
echo is a shell builtin  
echo is /bin/echo
```

### 3.4.3 ALIASES

Alias dapat digunakan untuk memetakan *command* yang lebih panjang ke urutan *key* yang lebih pendek. Ketika *shell* menjumpai alias dieksekusi, itu menggantikan urutan yang lebih panjang sebelum melanjutkan untuk menafsirkan *command*.

Misalnya, *command ls -l* biasanya alias menjadi *l* atau *ll*. Karena *command* yang lebih sedikit ini lebih mudah untuk diketik, sehingga lebih cepat untuk menjalankan *ls -l command line*.

Untuk menentukan alias apa yang ditetapkan pada *shell* saat ini, gunakan *command alias*:

```
sysadmin@localhost:~$ alias
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l='ls -CF'
alias la='ls -A'
alias ll='ls -alF'
alias ls='ls --color=auto'
```

Alias dari contoh sebelumnya dibuat oleh file inisialisasi. *File-file* ini dirancang untuk membuat proses pembuatan alias otomatis.

Alias baru dapat dibuat menggunakan format berikut, di mana *name* adalah nama yang akan diberikan alias dan *command* adalah *command* yang akan dieksekusi ketika alias dijalankan.

```
alias name=command
```

Contohnya, *command cal 2019* menampilkan kalender untuk tahun 2019. Misal Anda sering menjalankan *command* ini. Alih-alih mengeksekusi *command* seluruhnya setiap saat, Anda dapat membuat alias yang disebut *mycal* dan jalankan alias, seperti yang ditunjukkan pada gambar berikut:

```
sysadmin@localhost:~$ alias mycal="cal 2019"
sysadmin@localhost:~$ mycal
               2019
  January          February          March
Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa   Su Mo Tu We Th Fr Sa
      1  2  3  4  5           1  2           1  2
  6  7  8  9 10 11 12  3  4  5  6  7  8  9  3  4  5  6  7  8  9
13 14 15 16 17 18 19 10 11 12 13 14 15 16 10 11 12 13 14 15 16
20 21 22 23 24 25 26 17 18 19 20 21 22 23 17 18 19 20 21 22 23
27 28 29 30 31       24 25 26 27 28       24 25 26 27 28 29 30
                           31
```

Alias yang dibuat dengan cara ini hanya bertahan selama *shell* terbuka. Setelah *shell* ditutup, alias baru hilang. Selain itu, setiap *shell* memiliki alias sendiri, jadi alias yang dibuat dalam satu *shell* tidak akan tersedia di shell baru yang dibuka.

*Command type* dapat mengidentifikasi alias ke *command* lain:

```
sysadmin@localhost:~$ type ll
ll is aliased to `ls -alF'
sysadmin@localhost:~$ type -a ls
ls is aliased to `ls --color=auto'
ls is /bin/ls
```

*Output* dari *command* ini menunjukkan `ll` adalah alias untuk `ls -alF`, dan bahkan `ls` adalah alias untuk `ls --color=auto`.

#### 3.4.4 FUNCTIONS

*Functions* juga dapat dibangun menggunakan *command* yang ada untuk membuat *command* baru, atau untuk menimpa *command* bawaan ke *shell* atau *command* yang disimpan dalam *file*. *Aliases* dan *functions* biasanya dimuat dari *file* inisialisasi ketika *shell* pertama kali dimulai.

### 3.5 QUOTING

Tanda kutip atau *quotation* digunakan di seluruh administrasi Linux dan sebagian besar bahasa pemrograman komputer untuk membiarkan sistem tahu bahwa informasi yang terkandung dalam tanda kutip harus diabaikan atau diperlakukan dengan cara yang sangat berbeda dari biasanya akan diperlakukan. Ada tiga jenis tanda kutip yang memiliki arti khusus bagi Bash *shell*: tanda kutip

ganda ", tanda kutip tunggal ', dan tanda kutip belakang`. Setiap rangkaian tanda kutip memperingatkan *shell* untuk tidak memperlakukan teks dalam tanda kutip dengan cara normal.

### 3.5.1 DOUBLE QUOTES

Kutipan ganda menghentikan *shell* dari menafsirkan beberapa *metacharacters* (karakter khusus), termasuk karakter *glob*.

Karakter glob, juga disebut *wild cards*, adalah simbol yang memiliki arti khusus untuk shell; mereka ditafsirkan oleh shell itu sendiri sebelum mencoba menjalankan command apa pun. Karakter global termasuk karakter tanda bintang \*, karakter tanda tanya ?, dan tanda kurung [ ].

Dalam tanda kutip ganda tanda bintang hanyalah tanda bintang, tanda tanya hanyalah tanda tanya, dan seterusnya, yang berguna ketika Anda ingin menampilkan sesuatu di layar yang biasanya merupakan karakter khusus untuk *shell*. Dalam *command echo* di bawah ini, Bash *shell* tidak mengubah pola *glob* menjadi nama *file* yang cocok dengan pola:

```
sysadmin@localhost:~$ echo "The glob characters are *, ? and [ ]"
The glob characters are *, ? and [ ]
```

Kutipan ganda masih memungkinkan substitusi *command*, *substitusi variabel*, dan mengizinkan beberapa karakter meta *shell* lain yang belum dibahas. Demonstrasi berikut menunjukkan bahwa nilai variabel PATH masih ditampilkan:

```
sysadmin@localhost:~$ echo "The path is $PATH"
The path is /usr/bin/custom:/home/sysadmin/bin:/usr/local/sbin:/usr/local/bin:/us
```

### 3.5.2 SINGLE QUOTES

Kutipan tunggal mencegah *shell* melakukan interpretasi karakter khusus, termasuk *glob*, variabel, substitusi *command*, dan karakter metakarakter lain yang belum dibahas.

Sebagai contoh, untuk membuat karakter \$ hanya berarti \$, daripada bertindak sebagai indikator untuk *shell* untuk mencari nilai variabel, jalankan *command* kedua yang ditampilkan di bawah ini:

```
sysadmin@localhost:~$ echo The car costs $100
The car costs 00
sysadmin@localhost:~$ echo 'The car costs $100'
The car costs $100
```

### 3.5.3 BACKLASH CHARACTER

Ada juga teknik alternatif untuk kutip tunggal pada dasarnya karakter tunggal. Pertimbangkan pesan berikut:

```
The service costs $1 and the path is $PATH
```

Jika kalimat di atas ditempatkan dalam tanda kutip ganda, \$1 dan \$PATH dianggap variabel.

```
sysadmin@localhost:~$ echo "The service costs $1 and the path is $PATH"
The service costs  and the path is /usr/bin/custom:/home/sysadmin/bin:/usr/local/
```

Jika ditempatkan dalam tanda kutip tunggal, \$1 dan \$PATH tidak dianggap sebagai variabel.

```
sysadmin@localhost:~$ echo 'The service costs $1 and the path is $PATH'
The service costs $100 and the path is $PATH
```

Tetapi bagaimana jika Anda ingin \$PATH diperlakukan sebagai variabel dan \$1 tidak?

Dalam kasus ini, gunakan karakter *backslash* \ di depan karakter tanda \$ untuk mencegah *shell* menafsirkannya. *Command* di bawah ini menunjukkan menggunakan karakter \:

```
sysadmin@localhost:~$ echo The service costs \$1 and the path is $PATH
The service costs $1 and the path is /usr/bin/custom:/home/sysadmin/bin:/usr/loc
```

### 3.5.4 BACKQUOTES

*Backquotes*, atau *backticks*, digunakan untuk menentukan *command* dalam suatu *command*, suatu proses yang disebut substitusi *command*. Ini memungkinkan penggunaan *command* yang kuat dan canggih.

Walaupun mungkin terdengar membingungkan, sebuah contoh seharusnya membuat segalanya lebih jelas. Untuk memulai, catat *output* dari *command date*:

```
sysadmin@localhost:~$ date  
Mon Nov  4 03:35:50 UTC 2018
```

Sekarang, perhatikan *output* dari *command echo*:

```
sysadmin@localhost:~$ echo Today is date  
Today is date
```

Dalam *command* sebelumnya, kata *date* diperlakukan sebagai teks biasa, dan *shell* meneruskan *date* ke *command echo*. Untuk mengeksekusi *command date* dan mengirimkan *output command* itu ke *command echo*, masukkan *command date* di antara dua karakter *backquote*:

```
sysadmin@localhost:~$ echo Today is `date`  
Today is Mon Nov 4 03:40:04 UTC 2018
```

## 3.6 STATEMENT KONTROL

Statement kontrol memungkinkan Anda untuk menggunakan beberapa *command* sekaligus atau menjalankan *command tambahan*, tergantung pada keberhasilan *command* sebelumnya. Biasanya pernyataan kontrol ini digunakan dalam *script*, tetapi mereka juga dapat digunakan di baris *command* juga.

### 3.6.1 SEMICOLON

```
command1; command2; command3
```

Karakter titik koma “;” dapat digunakan untuk menjalankan banyak *command*, satu demi satu. Setiap *command* berjalan secara independen dan berurutan; terlepas dari hasil *command* pertama, *command* kedua berjalan setelah yang pertama selesai, lalu yang ketiga dan seterusnya.

Misalnya, untuk mencetak bulan Januari, Februari dan Maret 2030, jalankan *command* berikut:

```
sysadmin@localhost:~$ cal 1 2030; cal 2 2030; cal 3 2030
      January 2030
Su Mo Tu We Th Fr Sa
          1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31

      February 2030
Su Mo Tu We Th Fr Sa
          1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28

      March 2030
Su Mo Tu We Th Fr Sa
          1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31
```

### 3.6.2 DOUBLE AMPERSAND

```
command1 && command2
```

*Double ampersand “&&” bertindak sebagai logis “dan”; jika *command* pertama berhasil, maka *command* kedua juga akan berjalan. Jika *command* pertama gagal, maka *command* kedua tidak akan berjalan.*

Untuk lebih memahami bagaimana ini bekerja, pertimbangkan dulu konsep kegagalan dan keberhasilan untuk *command*. *Command* berhasil ketika mereka bekerja dengan baik dan gagal ketika terjadi kesalahan. Sebagai contoh, perhatikan *command ls*. *Command* berhasil jika direktori yang diberikan dapat diakses dan gagal jika tidak.

Dalam contoh berikut, *command* pertama berhasil karena direktori /etc/ppp ada dan dapat diakses sementara *command* kedua gagal karena tidak ada direktori /etc/junk:

```
sysadmin@localhost:~$ ls /etc/ppp
ip-down.d ip-up.d
sysadmin@localhost:~$ ls /etc/junk
ls: cannot access /etc/junk: No such file or directory
```

Untuk menggunakan keberhasilan atau kegagalan *command* `ls` bersama dengan `&&` jalankan *command* seperti berikut. Pada contoh pertama, *command echo* dieksekusi karena *command* `ls` berhasil:

```
sysadmin@localhost:~$ ls /etc/ppp && echo success
ip-down.d ip-up.d
success
```

Dalam contoh kedua, *command echo* tidak dieksekusi karena *command* `ls` gagal:

```
sysadmin@localhost:~$ ls /etc/junk && echo success
ls: cannot access /etc/junk: No such file or directory
```

### 3.6.3 DOUBLE PIPE

```
command1 || command2
```

Pipa ganda “`||`” adalah logis “atau”. Bergantung pada hasil dari *command* pertama, *command* kedua akan berjalan atau dilewati. Dengan pipa ganda, jika *command* pertama berjalan dengan sukses, *command* kedua dilewati; jika *command* pertama gagal, maka *command* kedua dijalankan. Dengan kata lain, Anda pada dasarnya memberi tahu *shell*, “Jalankan *command* pertama ini atau yang kedua”.

Dalam contoh berikut, *command echo* hanya dijalankan jika *command* `ls` gagal:

```
sysadmin@localhost:~$ ls /etc/ppp || echo failed
ip-down.d ip-up.d
sysadmin@localhost:~$ ls /etc/junk || echo failed
ls: cannot access /etc/junk: No such file or directory
failed
```