Panduan Praktikum Sistem Operasi

2012

Panduan ini berisi langkah-langkah mudah Pemrograman Shell Bash pada sistem Operasi Linux.

Pemrograman Shell Bash di Linux

LABORATORIUM SISTEM INFORMASI
JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNIK
UNIVERSITAS TRUNOJOYO

PENDAHULUAN

Panduan ini berisi langkah-langkah mudah pemrograman Shell Bash pada sistem Operasi Linux. Setiap praktikan diharuskan mengerjakan contoh yang disediakan dan menyelesaikan tugas-tugas yang dibebankan oleh Asisten Praktikum. Praktikum terdiri dari 5 modul yang masing-masing terdiri dari Tugas Pendahuluan yang harus diselesaikan sebelum Praktikum, Tugas Praktik di Laboratorium di dampingi Asisten dan Tugas Penutup yang harus diselesaikan setelah Praktikum. Panduan ini diharapkan memudahkan Praktikan menyerap poin-poin penting dalam pemrograman Shell

Tujuan

Panduan ini ditulis untuk membantu peserta praktikum (praktikan) memahami dasardasar pemrograman skrip shell pada sistem operasi Linux, sekaligus memperkenalkan beberapa program sederhana namun *powerful* yang tersedia di bawah shell Bash (Bourne Again Shell). Panduan ini dapat digunakan secara mandiri atau latihan bersama kelompok serta dapat diberlakukan pada hampir semua distribusi Linux.

Syarat Pengguna

Panduan ini menganggap pengguna telah mampu:

- Menggunakan shell Linux Bash interaktif, termasuk beberapa perintah dasar terutama yang terkait dengan pengelolaan file dan direktori. Pada Distro Ubuntu Desktop GNome, Shell dapat diakses melalui program bernama Terminal.
- Menggunakan bahasa pemrograman tertentu seperti Pascal atau C, setidaknya mencakup struktur program, variabel, seleksi kondisi, perulangan dan fungsi merupakan pengetahuan yang sangat berguna.

Persiapan

Secara umum Linux terdiri dari 2 level pengguna.Pengguna pertama adalah root atau super user. Pengguna ini mempunyai akses tak terbatas terhadap sistem. Pada shell Bash, pengguna ini mendapatkan prompt yang diakhiri tanda "#". Pengguna kedua adalah pengguna umum yang aksesnya terhadap sistem ditentukan oleh root. Pengguna ini ditandai ditandai dengan prompt '$'.

Saat menggunakan Linux, sebaiknya ada login sebagai pengguna biasa, prompt $. Jika prompt anda bukan $, tulis perintah berikut sebagai command line dan tekan Enter.

```
PS1="$ " ; export PS1
```

Ada baiknya anda mencoba contoh di bawah ini. Pada prompt yang disediakan, tuliskan 4 baris pertama secara interaktif.

```
$ echo '#!/bin/sh' > skripku.sh
$ echo 'echo Halo, saya belajar Shell Linux' >> skripku.sh $
chmod +x skripku.sh
$ ./skripku.sh
Halo, saya belajar Shell Linux
```

Akan lebih baik jika anda berkeinginan menuliskan 4 baris kode berikut memanfaatkan suatu Text Editor, misalnya vi atau gEdit, dan menyimpannya sebagai halo.sh.

```
#!/bin/sh
# Nama file skrip: halo.sh
# Ini adalah komentar!
echo Halo, saya belajar Shell Linux # Ini juga komentar!
```

Agar skrip atau program shell halo.sh dan file-file skrip shell lainnya dapat diekskusi, anda harus mengubah status dari file tersebut sehingga statusnya menjadi executable. Caranya adalah sebagai berikut:

```
$ chmod +x halo.sh
```

Kemudian jalankan skrip tersebut dengan:

```
$ ./halo.sh
```

Referensi

Tabel di bawah ini memberikan daftar perintah yang sering digunakan dalam pemrograman shell Linux.

Perintah	Deskripsi	Contoh
&	Menjalankan perintah sebelumnya sebagai latar belakang	ls &
&&	Perbandingan AND	if ["$foo" -ge "0"] && ["$foo" -le "9"]
||	Perbandingan OR	if ["$foo" -lt "0"] \|\| ["$foo" -gt "9"]
^	Awal dari baris	grep "^foo"
$	Akhir dari baris	grep "foo$"
=	Kesamaan string	if ["$foo" = "bar"]
!	Perbandingan NOT	if ["$foo" != "bar"]
$$	PID dari shell aktif	echo "my PID = $$"
$!	PID dari perintah background terakhir	ls & echo "PID of ls = $!"
$?	Status exit dari perintah terakhir	ls ; echo "ls returned code $?"
$0	Nama dari perintah aktif (saat dipanggil)	echo "I am $0"
$1	Nama dari parameter pertama dari program	echo "My first argument is $1"
$9	Nama dari parameter ke-9 dari program	echo "My ninth argument is $9"
$@	Semua parameter program (termasuk spasi dan tanda petik)	echo "My arguments are $@"
$*	Semua parameter program (spasi dan tanda petik dihilangkan)	echo "My arguments are $*"
-eq	Kesamaan numerik	if ["$foo" -eq "9"]
-ne	Ketidaksamaan numerik	if ["$foo" -ne "9"]
-lt	Kurang dari	if ["$foo" -lt "9"]
-le	Kurang dari atau sama	if ["$foo" -le "9"]
-gt	Lebih dari	if ["$foo" -gt "9"]
-ge	Lebih dari atau sama	if ["$foo" -ge "9"]
-z	String panjangnya nol	if [-z "$foo"]
-n	String panjangnya tidak nol	if [-n "$foo"]

-nt	Lebih baru daripada	if ["$file1" -nt "$file2"]
-d	Apakah direktori?	if [-d /bin]
-f	Apakah file?	if [-f /bin/ls]
-r	File bersifat readable?	if [-r /bin/ls]
-w	File bersifat writable?	if [-w /bin/ls]
-x	File bersifat executable?	if [-x /bin/ls]
()	Definisi fungsi	function myfunc() { echo hello }

MODUL I

SHELL INTERAKTIF dan SKRIP

I.1 Tugas Pendahuluan

1. Apa yang dimaksud shell pada sistem operasi Linux? Apakah Windows juga mempunyai shell?
2. Sebutkan 20 perintah shell yang banyak digunakan untuk mengelola sistem operasi Linux!
3. Apa fungsi dari perintah ‘sudo’?

I.2 Tugas Praktik (di Laboratorium)

Pastikan anda berada di direktori home anda sendiri.

1. Gunakan perintah-perintah yang berkaitan dengan pengelolaan file seperti cp, mv, touch dan cat
2. Baca manual dari grep dan tr. Coba berbagai variasidari contoh yang diberikan pada panduan pemrograman shell.
3. Coba buka file /etc/passwd dengan perintah cat atau more. Gunakan grep dan ambil informasi mengenai pengguna tertentu!
4. Biasakan diri anda dengan Text Editor berbasis console seperti vi dan pico!

I.3 Tugas Penutup

1. Buat sebuah program skrip sederhana untuk menampilkan daftar file yang ada di dalam direktori aktif!
2. -
3. -

Shell dapat dikatakan sebagai tempat interaksi antara pengguna dan sistem Linux yang sedang digunakan. Shell juga bertugas menerjemahkan atau mengeksekusi program. Shell menyediakan suatu antarmuka teks (command line). Shell dapat diguakan untuk mengelola (administering) sistem Linux seperti menambah pengguna, mengatur file dan direktori dan memantau kerja dari sistem operasi. Shell Linux ditandai oleh $ (user reguler) atau # (root).

Shell Bash dapat digunakan dalam dua modus, yaitu modus interaktif dan modus skrip. Pada modus interaktif, anda menuliskan satu baris perintah disamping prompt dan mengakhirinya dengan ENTER. Hasil eksekusi akan ditampilkan langsung pada layar dan anda kembali disediakan prompt untuk memasukkan perintah-perintah berikutnya. Pada modus skrip, anda menuliskan skrip atau kode program, berisi perintah-perintah Linux, menyimpannya ke dalam sebuah file dengan ekstensi .sh. Selanjutnya file ini dijalankan pada modus Interaktif.

I.1 Shell Interaktif

Bagian ini memperlihatkan bagaimana memanfaatkan shell interaktif untuk mengadministrasi sistem Linux yang digunakan.

Sebagai latihan awal, coba tulis (akhiri dengan enter) setiap perintah di bawah ini dan perhatikan luaran yang dihasilkan (‘$’ tidak diketik, itu hanya menandakan prompt):

```
$ date; whoami; pwd
$ ps
$ top
$ ls -la

$  echo  ”Kereeeen”
$ a=10; echo $a
$ b=109; echo "B bernilai :" $b
```

Secara garis besar anda mengetahui apa fungsi dari perintah di atas, dengan melihat luarannya. Perintah date untuk mendapatkan tanggal dan jam, whoami (who am i) untuk mengetahui nama login dari pengguna, pwd (print working directory) untuk mendapatkan nama direktori aktif (dimana anda berada). Perintah top (table of processes) untuk memperoleh daftar proses yang sedang berjalan. Perintah ls (list) untuk mendapatkan daftar file dan direktori dimana pengguna aktif berada.

Bagaimana dengan perintah-perintah lainnya? Anda belum tahu kegunaannya? Luaran dari eksekusi di atas belum jelas? Anda dapat mengetahui fungsi dan cara menggunakan perintah di Linux memanfaatkan perintah man, singkatan dari manual.

```
$ man ls
$ man ps
$ man file
```

Perintah ‘man ls’ digunakan untuk mendapatkan manual dari perintah ls. Manual berisi fungsi dari perintah, cara pemanggilan perintah bersama dengan parameter-parameter yang mungkin disertakan dan penjelasan lebih lanjut mengenai setiap cara penggunaan dan sering disertai beberapa contoh.

Bagaimana mencari file dengan ekstensi tertentu? Misalnya *.jpg? Silakan gunakan perintah ‘man find, baca dokumentasinya secara teliti dan selesaikan masalah ini.

Sekarang jalankan perintah-perintah di bawah ini, satu demi satu dan jelaskan maksudnya!

```
$ find . -name '*.jpg' $
file nama_file
$ echo “belajar shell linux” > nama_file $
echo “keren abis” >> nama_file
$ echo “OK...sepakat....te o pe deh” >> nama_file
$ cat nama_file
$ more nama_file
$ grep keren nama_file $
man grep
```

Apa kegunaan dari perintah grep? Coba jalankan dua baris perintah yang mengandung perintah grep. Samakah hasilnya? Apa perbedaannya?

```
$ cat nama_file | grep "OK"
$ grep "OK" nama_file
```

I.2 Pipeline dan Redirection

Pipeline (garis pipa, |) dapat digunakan untuk mengirimkan luaran dari suatu perintah ke perintah lain. Menggunakan *pipe*, luaran dari perintah di sebelah kiri *pipe* akan dijadikan masukan bagi perintah di sebelah kanan *pipe*.

Jalankan baris demi baris di bawah ini dan cermati hasilnya!

```
$ echo ”aris joko faza fenny ubaid”
$ echo ”aris joko faza fenny ubaid” | tr ” ” ”\n”
$ echo ”aris joko faza fenny ubaid” | tr ” ” ”\n” | sort
$ echo ”aris joko faza fenny ubaid” | tr ” ” ”\n” | sort -r
```

Anda tentu telah paham manfaat dari perintah echo, yaitu hanya mencetak teks yang dijadikan parameter saat pemanggilannya. Perintah tr digunakan untuk mentranslasi daftar string. Parameter “ “ “\n” pada pemanggilan tr mengatakan bahwa ganti spasi (“ “) dengan karakter baris baru (*newline*, \n). Perintah sort dapat digunakan untuk mengurutkan baris-baris string (teks). Apa yang dihasilkan oleh baris ke-4 di atas? Apa perbedaannya dengan baris ke-3?

Redirection atau pengalihan digunakan untuk mengalihkan luaran eksekusi dari suatu perintah. Secara default, luaran dari suatu perintah adalah layar (screen). Kita dapat mengalihkan luaran tersebut ke suatu file dengan menggunakan operator > atau >>. Operator > digunakan untuk membuat file baru dan memasukkan teks baru ke dalamnya. Operator >> digunakan untuk menambahkan entri selanjutnya ke dalam file yang dibuat oleh operator > sebelumnya.

Sekarang, jalankan perintah di bawah ini secara benar dan perhatikan hasil yang diperoleh!

```
$ date > sekarang.txt
$ cat sekarang.txt
$ date >> sekarang.txt
$ cat sekarang.txt
```

I.3 Skrip Shell Pertama

Pada bagian ini anda akan belajar menuliskan sebuah skrip yang sekedar menampilkan pesan "Halo dunia". Gunakan Text Editor dan buat file teks bernama pertama.sh seperti di bawah ini:

```
#!/bin/bash
# Nama file skrip: pertama.sh
# Skrip ini menampilkan teks menggunakan perintah echo
echo Halo dunia          # mencetak teks Halo dunia
```

Baris pertama memberitahukan sistem Linux bahwa file akan dieksekusi oleh program bernama /bin/bash. Ini merupakan lokasi standard dari shell Bash pada banyak distribusi GNU/Linux. Pada beberapa distribusi Linux, ada perintah /bin/sh yang merupakan link simbolik ke bash. Baris pertama dari setiap skrip harus dan hanya mencantumkan "`#!/bin/bash`" atau "`#!/bin/sh`".

Baris kedua dimulai dengan simbol khusus, `#`. Ini menandakan baris tersebut sebagai suatu komentar. Shell akan mengabaikannya, tidak memroses komentar tersebut.

Satu-satunya pengecualian adalah simbol `#!` yang diletakkan di baris pertama dari file skrip - seperti pada contoh di atas. Ini adalah direktif khusus.

Jika anda terbiasa dengan Perl, tentu telah menjumpai adanya `#!/usr/bin/perl` pada baris pertama setiap skrip Perl untuk memberitahukan shell interaktif bahwa program tersebut akan dieksekusi oleh Perl. Pada pemrograman shell Bash adalah `#!/bin/sh`.

Baris ketiga menjalankan suatu perintah `echo` dengan dua parameter atau argumen - pertama adalah "`Halo`" dan kedua adalah "`dunia`". Perintah `echo` akan secara otomatis meletakkan spasi tunggal di antara parameter-parameternya.

Simbol # menandakan komentar. Karakter # dan apapun yang mengikutinya, pada baris yang sama, diabaikan oleh shell.

Sekarang jalankan `chmod 755 pertama.sh` atau (`chmod +x pertama.sh`) untuk membuat file teks *executable* dan jalankan skrip dengan `./pertama.sh`.

Pada jendela shell (terminal) terlihat sebagai berikut:

```
$ chmod 755 pertama.sh $
./pertama.sh
Halo dunia
```

I.4 Perintah echo

Apakah hanya untuk menampilkan dua kata tersebut kita harus menulis skrip? TIDAK. Anda dapat menggunakan shell interaktif untuk memperoleh hasil yang sama. Tuliskan langsung perintah `echo Halo dunia` di shell dan perhatikan luarannya:

```
$ echo Halo dunia
Halo dunia
```

Sekarang buat sedikit perubahan. Pertama, ingat bahwa `echo` meletakkan SATU spasi antara parameter-parameternya. Tempatkan beberapa spasi antara teks "Halo" dan "dunia". Hasil seperti apa yang diharapkan? Bagaimana jika anda meletakkan karakter TAB di antaran dua kata tersebut?

Bagaimana luaran dari perubahan tersebut? Output skrip sama persis dengan

sebelumnya. Kita memanggil program `echo` dengan dua argumen; `echo` tidak menghiraukan berapa pun jumlah spasi di antaranya. Sekarang, coba ubah lagi skrip tersebut menjadi:

```
#!/bin/sh
# Ini baris komentar!
echo "Halo           dunia"       # Ini juga komentar
```

Kali ini spasi yang ditambahkan tampil sebagai output. Mengapa? Karena `echo` dipanggil dengan hanya SATU argumen yang diapit oleh petik ganda "`Halo dunia`". Tampilan di layar tepat sama dengan yang dituliskan dalam skrip. Perlu dipahami bahwa shell mem-*parse* argumen SEBELUM melewatkannya ke program yang dipanggil. Shell menghilangkan tanda petik dan melewatkan string tersebut sebagai satu argumen.

Contoh kedua, masih berkaitan dengan perintah `echo`. Tulis skrip berikut. Perkirakan luarannya sebelum menjalankannya:

```
#!/bin/sh
# Nama file skrip: kedua.sh
# Apa yang dihasilkan skrip ini?
echo "Halo      dunia  "       # mencetak teks Halo dunia
echo "Halo dunia "
echo "Halo * dunia "
echo Halo * dunia
echo Halo      dunia
echo "Halo" dunia
echo Halo "      "   dunia
echo "Halo \"*\" dunia "
echo `halo` dunia
echo 'hello' dunia
```

Apakah luarannya sesuai yang diharapkan? Jika tidak, jangan khawatir! Ini hanya beberapa hal yang akan dibahas pada bagian-bagian lanjut dalam panduan ini. Dan tentu anda akan belajar perintah-perintah lain yang lebih powerful daripada `echo`.

MODUL II

VARIABEL & KARAKTER *ESCAPE*

II.1 Tugas Pendahuluan

1. Bagaimana mengubah semua file terakhir .html menjadi.php? (coba gunakan berbagai bentuk perintah mv)!
2. Jelaskan dengan singkat shell lain yang tersedia di Linux selain Bash!

II.2 Tugas Praktik (di Laboratorium)

1. Tulis dan jalankan skrip berikut:

```
#!/bin/bash
# Nama file skrip: var10.sh
a=5.66
b=8.67
c=`echo $a + $b | bc`
echo "$a + $b = $c"
```

2. Tulis dan jalankan skrip berikut:

```
#!/bin/bash

echo "Halo, $LOGNAME"
echo "Tanggal sekarang `date`"
echo "Pengguna: `who i am`"
echo "Direktori aktif `pwd`"
```

3. Modifikasi program no.2 di atas sehingga juga dapat menampilkan daftar file dan direktori yang terdapat di dalam direktorti aktif!

II.3 Tugas Penutup

1. Tulis dan jalankan skrip berikut:

```
#!/bin/bash

echo ”Daftar file Anda: ”;
ls -l
echo ”Thank U very Much”;
```

2. -

3. -

Saat ini, setiap bahasa pemrograman mempunyai konsep tentang variabel - suatu nama simbolik mewakili suatu alamat memory yang dapat diberikan suatu nilai, dibaca dan dimanipulasi. Demikian pula pada shell Bash. Bagian ini menjelaskan tentang variabel yang dapat dibuat oleh pemrogram. Bagian II akan menguraikan tentang variabel yang disediakan oleh sistem operasi.

Kembali perhatikan contoh pertama, Halo dunia. Ini dapat dikerjakan menggunakan variabel.

II.1 Pembuatan Variabel

Perlu dicatat bahwa harus tidak ada spasi (ruang) sebelum dan sesudah tanda sama dengan ("="): `VAR=value` benar sedangkan `VAR = value` tidak bekerja. Pada kasus pertama, shell melihat simbol "=" dan memperlakukan perintah sebagai pemberian atau pelewatan variabel. Pada kasus kedua, shell menganggap bahwa VAR harus berupa nama perintah dan mencoba mengeksekusinya.

Coba tulis kode berikut ke dalam file skrip var1.sh:

```
#!/bin/sh
# Nama file skrip: var1.sh
MY_MESSAGE="Halo dunia" echo
$MY_MESSAGE
```

Ini melewatkan string "Halo dunia" ke variabel `MY_MESSAGE` kemudian meng-`echo`-kan nilai dari variabel tersebut.

Kita harus mengapit string `Halo dunia` dengan tanda petik. Tidak ada masalah dengan perintah `echo Halo dunia`. Perintah echo menerima semua parameter yang diberikan. Variabel hanya dapat memegang satu nilai, sehingga suatu string yang mengandung spasi harus diapit tanda petik agar shell mengetahui dan menganggap semuanya sebagai satu. Jika tidak, shell akan mencoba mengeksekusi perintah `dunia` setelah `MY_MESSAGE=Halo`.

Shell tidak mengenal tipe variabel; variabel boleh menyimpan string, integer dan bilangan ril - apa pun yang diinginkan. Programmer yang telah terbiasa dengan PHP dan Perl menyukai ini. Jika anda menggunakan C, Pascal atau Ada, ini terasa aneh. Semua nilai disimpan sebagai string, tetapi rutin yang mengharapkan numerik dapat memperlakukannya sebagai nilai numerik.

Jika anda melewatkan string ke suatu variabel kemudian mencoba untuk menambahkan 1 ke variabel tersebut, anda dilarang melakukannya:

```
$ x="hello"
$ y=`expr $x + 1`
expr: non-numeric argument
```

Karena program eksternal bernama `expr` hanya mengharapkan numerik. Berikut ini adalah contoh pembuatan variabel yang benar:

```
MY_MESSAGE="Halo dunia"
MY_SHORT_MESSAGE=hi
MY_NUMBER=1;            MY_PI=3.142
MY_OTHER_PI="3.142";    MY_MIXED=123abc
```

Karakter khusus harus dengan tepat di-*escape* untuk menghindari salah interpretasi oleh shell.

Kita dapat secara interaktif meminta pengguna memasukkan nilai untuk suatu variabel

menggunakan perintah `read`. Kode berikut menanyakan pengguna nilai untuk variabel MY_NAME, menggabungkan nilai variabel tersebut dengan string lain dan mencetaknya.

```
#!/bin/sh
# Nama file skrip: var2.sh
echo Who are U?
read MY_NAME
echo "Hello $MY_NAME - hope you're well."
```

Apa yang terjadi jika baris terakhir tidak dilingkupi tanda pentik ganda? Karakter petik tunggal di dalam "you're" tidak cocok sehingga dapat memunculkan error. Hati-hati!

Contoh di atas menggunakan perintah bawaan (*built-in*) shell `read` yang membaca baris dari input standard (biasanya keyboard) ke dalam variabel. Jika anda memasukkan nama lengkap (lebih dari satu kata) dan tidak menggunakan petik ganda di awal dan akhir perintah `echo`, maka luaran masih tetap benar. Bagaimana ini terjadi? Variabel `MY_NAME` telah lingkupi petik ganda. Perintah `read` secara otomatis menempatkan tanda petik sekeliling inputnya, sehingga spasi tersebut diperlakukan dengan tepat.

II.2 Lingkup Variabel

Variabel dalam shell Bash tidak harus dideklarasikan, sebagaimana dilakukan dalam bahasa lain seperti C. Tetapi jika anda membaca suatu variabel yang tidak dideklarasi, hasilnya adalah string kosong. Anda tidak mendapatkan pesan error. Ini dapat menyebabkan beberapa bug kecil - jika anda mempunyai variabel `MY_OBFUSCATED_VARIABLE=Hello` dan kemudian memanggil `echo $MY_OSFUCATED_VARIABLE`. Anda tidak akan mendapatkan apapun (karena `OBFUSCATED` kedua salah ejaan).

Ada perintah `export` yang mempunyai suatu efek penting terhadap lingkup variabel. Anda harus memahami bagaimana perintah ini digunakan. Buat sebuah skrip shell kecil bernama `myvar2.sh`:

```
#!/bin/sh
# Nama file skrip: myvar2.sh
echo "MYVAR is: $MYVAR"
MYVAR="hako sen there"
echo "MYVAR is: $MYVAR"
```

Sekarang jalankan skrip tersebut:

```
$ ./myvar2.sh
MYVAR is:
MYVAR is: hi there
```

MYVAR belum diset ke suatu nilai, sehingga blank (kosong). Bagimana jika kita berikan suatu nilai untuk variabel tersebut melalui shell interaktif? Apa hasilnya? Lakukan apa yang diilustrasikan di bawah ini:

```
$ MYVAR=hello
$   ./myvar2.sh
MYVAR is:
MYVAR is: hi there
```

Ternyata variabel MYVAR masih belum menyimpan suatu nilai. Mengapa? Saat anda memanggil `myvar2.sh` dari shell interaktif, suatu shell baru dilahirkan untuk menjalankan skrip. Ini terjadi karena adanya baris `#!/bin/sh` pada awal skrip. Artinya, variabel MYVAR pada shell interaktif berbeda dengan MYVAR di dalam skrip yang dijalankan pada shell lain.

Bagaimana agar nilai variabel pada shell interaktif dapat masuk ke dalam skrip? Perintah `export` jawabannya. Sekarang tulis baris-baris berikut:

```
$ export MYVAR
$ ./myvar2.sh
MYVAR is: hello
MYVAR is: hi there
```

Perhatikan baris ke-3 dari skrip yang bertugas untuk mengubah nilai dari `MYVAR`. Tetapi tidak ada cara yang dapat digunakan untuk mengirim balik nilai variabel ke shell interaktif. Coba baca nilai dari `MYVAR` setelah eksekusi skrip shell:

```
$ echo $MYVAR
hello
```

Begitu skrip shell keluar, variabel lingkungannya dihancurkan. Tetapi `MYVAR` menjaga nilainya `hello` di dalam shell interaktif. Adakah caranya agar nilai variabel dari dalam skrip shell juga dapat diekspor ke shell interaktif? Gunakan operator titik "." saat memanggil skrip shell. Jika sebelumnya anda memanggil skrip dengan "`./nama_file.sh`" maka sekarang anda harus menulisnya "`. ./nama_file.sh`".

Perhatikan contoh berikut:

```
$ MYVAR=hello
$  echo  $MYVAR
hello
$ . ./myvar2.sh
MYVAR is: hello
MYVAR  is:  hi  there
$ echo $MYVAR
hi there
```

Inilah bagaimana file `.profile` atau `.bash_profile` bekerja, sebagai contoh. Dalam kasus ini, kita tidak perlu meng-`export MYVAR`. Pastikan bukan `echo MYVAR` tetapi `echo $MYVAR`.

Satu hal lain yang sering memunculkan kesalahan dalam pemanfaatan variabel adalah seperti pada contoh berikut:

```
#!/bin/sh
# Nama file skrip: user.sh
echo " Siapa nama anda?"
read USER_NAME
echo " Halooo $USER_NAME"

echo  "  Buat  file  bernama  $USER_NAME_file"
touch $USER_NAME_file

echo "Test  1  2  3"  >>  "${USER_NAME}_file"
cat "${USER_NAME}_file"
```

Apa yang dihasilkan skrip user.sh di atas? Misalnya anda memasukkan "steve" sebagai `USER_NAME`, apakah skrip akan membuatkan file bernama `steve_file`? TIDAK. Ini akan memunculkan error kecuali sebelumnya telah ada ada sebuah variabel bernama `USER_NAME_file`. Shell tidak mengetahui dimana ujung dari variabel. Bagaimana menyelesaikan ini? Jawabannya adalah kita meletakkan variabel tersebut di dalam kurung kurawal.

Sekarang ubah skrip di user.sh menjadi sebagai berikut:

```
#!/bin/sh
echo "Siapa nama anda?"
read USER_NAME
echo "Halooo $USER_NAME"

echo "Buat file bernama ${USER_NAME}_file"
touch ${USER_NAME}_file

echo "Test 1 2 3" >> "${USER_NAME}_file"
cat "${USER_NAME}_file"
```

Shell telah mengetahui bahwa kita mengacu ke variabel `USER_NAME` dan ingin menambahkan akhiran `_file` kepada nilai dari variabel. Berhati-hatilah. Banyak pemrogram shell pemula melupakan hal sederhana ini.

Perlu juga dicatat bahwa petik ganda yang melingkupi `"${USER_NAME}_file"` - jika pengguna memasukkan "Steve Parker" (perhatikan spasi) maka tanpa tanda petik, argumen yang dilewatkan ke perintah `touch` menjadi `Steve` dan `Parker_file` - yaitu, dapat dikatakan berupa `touch Steve Parker_file` yang akan men-touch dua file, bukan satu. Tanda petik menghindari masalah ini.

II.3 *Wildcard*

Wildcard atau karakter asterisk (*) bukanlah hal baru jika anda telah biasa menggunakan *console* Linux. Bagian ini memaparkan bagaimana asterisk digunakan dalam skrip shell.

Bagaimana anda menyalin semua file dari direktori `/tmp/a` ke dalam direktori `/tmp/b`. Bagaimana jika yang disalin semua file berekstensi .txt? Semua file terakhiran .html? Anda dapat menjawabnya dengan:

```
$ cp /tmp/a/* /tmp/b/
$ cp /tmp/a/*.txt /tmp/b/
$ cp /tmp/a/*.html /tmp/b/
```

Sekarang bagaimana anda mendapatkan daftar file di dalam `/tmp/a/` tanpa menggunakan perintah `ls /tmp/a/`? Bagaimana dengan `echo /tmp/a/*`? Apa perbedaan utama luaran dari echo dan `ls`? Bergunakah ini? Bagaimana anda mengganti nama (*rename*) semua file terekstensi .txt menjadi .bak?

Perintah:

```
$ mv *.txt *.bak
```

Tidak akan memberikan hasil yang diharapkan. Coba gunakan `echo` bukan `mv`.

II.4 Karakter *Escape*

Ada sejumlah karakter tertentu memiliki arti khusus bagi shell; Misalnya karakter petik ganda (") yang menyebabkan shell memperhitungkan spasi dan TAB dalam pemrosesan teks, sebagai contoh:

```
$ echo Halo        Dunia
Halo Dunia
$ echo "Halo        Dunia"
Halo     Dunia
```

Bagaimana menampilkan `Halo      "Dunia"` ?

```
$ echo "Halo    \"Dunia\""
```

Karakter " pertama dan terakhir membungkus semuanya ke dalam satu parameter yang dilewatkan ke perintah `echo` sehingga spasi yang ada antara dua kata tetap dijaga. Tetapi kode:

```
$ echo "Halo    " Dunia ""
```

Akan diinterpretasikan sebagai tiga parameter:

- "Halo "
- Dunia
- ""

Sehingga luarannya berupa:

```
Hello    World
```

Perhatikan, tanpa petik ganda yang mengapit kata `Dunia` dinyatakan hilang. Ini karena tanda petik pertama dan kedua mematikan teks `Hello` dan diikuti oleh spasi, sebagai argumen pertama; argumen kedua adalah teks `Dunia` tanpa tanda petik dan argumen ketiga merupakan string kosong "".

Anda harus berhati-hati, perhatikan kode berikut:

```
$ echo "Hello    "World""
```

Sebenarnya memanggil perintah echo dengan hanya satu parameter (tak ada spasi antara paramater yang diapit tanda petik) dan anda dapat menguji ini dengan mengganti perintah `echo` dengan misalnya perintah `ls`.

Sebagian besar karakter (`*`, `'`, dll) tidak diterjemahkan jika diletakkan di dalam petik ganda ("). Semua karakter diterima begitu saja dan dilewatkan ke perintah yang dipanggil. Contoh menggunakan asterisk (*) adalah sebagai berikut:

```
$ echo *
case.shtml escape.shtml first.shtml
functions.shtml  hints.shtml  index.shtml
ip-primer.txt raid1+0.txt

$ echo *txt
ip-primer.txt raid1+0.txt

$ echo "*"
*
```

```
$  echo  "*txt"
*txt
```

Pada contoh pertama, * berarti semua file di dalam direktori aktif. Pada contoh kedua, *txt berarti semua file yang berakhiran `txt`. Pada contoh ketiga, ditempatkan * di dalam petik ganda dan ini diinterpretasikan secara literal. Pada contoh ke-4, sama dengan contoh ke-3, hanya * disambung dengan string txt.

Namun, `"`, `$`, `` ` ``, dan `\` masih diterjemahkan oleh shell, bahkan saat ada di apit oleh petik ganda. Karakter backslash (\) dapat digunakan untuk menandakan karakter-karakter khusus ini sehingga tidak diterjemahkan oleh shell, tetapi dilewatkan langsung kepada perintah yang akan dijalankan (misalnya `echo`). Sehingga untuk menampilkan string (misal nilai `$X` adalah 5):

```
A quote is ", backslash is \, backtick is `. A
few spaces are and dollar is $. $X is 5.
```

Kita harus menulis:

```
$ echo "A quote is \", backslash is \\, backtick is \`."
A quote is ", backslash is \, backtick is `.
$ echo "A few spaces are     ; dollar is \$. \$X is ${X}."
A few spaces are     ; dollar is $. $X is 5.
```

Kita telah melihat kekhususan dari " untuk memastikan spasi dalam teks. Tanda dollar juga khusus karena menandai suatu variabel, sehingga `$X` digantikan oleh shell dengan isi dari variabel `X`. Backslash bersifat khusus karena menandakan karakter khusus lain. Perhatikan pemanfaatan karakter escape di bawah ini:

```
$ echo "This is \\ a backslash"
This is \ a backslash
$ echo "This is \" a quote and this is \\ a backslash"
This is " a quote and this is \ a backslash
```

Jadi *backslash* sendiri harus di-*escape*-kan untuk dapat ditampilkan.

MODUL III

PERULANGAN dan SELEKSI

III.1 Tugas Pendahuluan

1. Apa manfaat dari karakter escape? Bagaimana menampilkan karakter ^, % dan ~?
2. Anda sudah mempelajari perulangan dan seleksi kondisi pada bahasa. Apa kegunaan keduanya? Buat sebuah program dalam bahasa C yang melibatkan kedua fitur ini, misalnya program tebak angka!
3. Apa yang dikeluarkan oleh perintah berikut:

```
$ ls -ld {,usr,usr/local}/{bin,sbin,lib}
```

III.2 Tugas Praktik (di Laboratorium)

1. Buat sebuah skrip untuk memeriksa keshahihan username dari sistem Linux! Penguna memberikan inputan dari keyboard (gunakan perintah `read`).
2. Buat sebuah program untuk menampilkan daftar semua pengguna yang ada di dalam sistem Linux mencakup Username, Nama lengkap dan Home directory-nya!
3. Selesaikan program tebak angka pada tugas pendahuluan dengan menggunakan perintah `case...esac`!

III.3 Tugas Penutup

1. -
2. -
3. -

III.1 Perulangan

Sebagian besar bahasa pemrograman mempunyai konsep perulangan atau loop. Jika kita perlu mengulangi suatu tugas sebanyak 20 kali, kita tidak harus menuliskan kode yang sama sebanyak 20 kali. Shell Bash menyediakan perulangan `for` dan `while`. Shell Linux memang menyediakan lebih sedikit fitur dibandingkan bahasa dunia C/C++.

Perulangan For

Perulangan `for` digunakan untuk melakukan pekerjaan berulang sebanyak daftar yang disediakan:

```
#!/bin/sh
# Nama file skrip: for1.sh
for i in 1 2 3 4 5
do
  echo "Looping ... number $i"
done
```

Coba eksekusi dan perhatikan hasil yang diperoleh. Program di atas meminta shell mengeksekusi perintah-perintah di dalam `blok do...done` sebanyak jumlah elemen di dalam daftar setelah in, yaitu 5 (1, 2, 3, 4, 5).

Penulisan elemen dari daftar juga dapat berbentuk seperti pada contoh di bawah ini:

```
#!/bin/sh
# Nama file skrip: for2.sh
for i in hello 1 * 2 goodbye
do
  echo "Looping ... i is set to $i"
done
```

Pastikan bahwa anda memahami apa yang terjadi di sini. Coba hilangkan tanda `*` dan ulangi lagi dengan kembali menyertakan *. Anda tentu tahu apa yang dihasilkan saat eksekusi. Coba jalankan skrip di atas dari dalam direktori lain. Coba apit * dengan tanda petik ganda, menjadi "*". Bagaimana jika * didahului backslash (`\*`) ?

Skrip pertama (for1.sh) memberikan hasil eksekusi berikut:

```
Looping..... number 1
Looping..... number 2
Looping..... number 3
Looping..... number 4
Looping..... number 5
```

Sedangkan skrip kedua (for2.sh) menunjukka hasil ekskusi berikut:

```
Looping ... i is set to hello
Looping ... i is set to 1
Looping ... i is set to (nama dari file pertama dalam direktori aktif)
    ... dan seterusnya ...
Looping ... i is set to (nama dari file terakhir dalam direktori aktif)
Looping ... i is set to 2
Looping ... i is set to goodbye
```

Seperti terlihat, `for` melakukan perulangan sebanyak input yang diberikan kepadanya, dari input pertama sampai terakhir.

Perulangan While

Perulangan `while` dapat digunakan untuk melakukan pekerjaan berulang yang jumlah perulangannya tidak pasti tetapi bergantung pada suatu kondisi yang harus dipenuhi. Perhatikan skrip berikut:

```
#!/bin/sh
# Nama file skrip: while.sh
INPUT_STRING=hello
while [ "$INPUT_STRING" != "bye" ]
do
  echo "Ketikkan sesuatu (bye untuk keluar)"
  read INPUT_STRING
  echo "Anda mengetikkan: $INPUT_STRING"
done
```

Apa yang terjadi di sini? Pernyataan `echo` dan `read` akan dijalankan secara terus menerus sampai anda mengetikan "bye". Mengapa variabel `INPUT_STRING` diberi nilai awal “hello”?

Simbol titik-dua (`:`) selalu bernilai true; ini dapat digunakan pada waktu tertentu. Skrip di bawah ini menunjukkan cara keluar program yang lebih elegan dari sebelunnya.

```
#!/bin/sh
# Nama file skrip: while2.sh
while :
do
  echo "Ketikkan sesuatu (^C untuk keluar)"
  read INPUT_STRING
  echo "Anda mengetikkan: $INPUT_STRING"
done
```

Shell menyediakan while berbentuk `while read f`. Contoh di bawah ini menggunakan pernyataan `case` yang akan dihabas pada bagian lain panduan ini. Skrip ini membaca dari file `myfile`, dan untuk setiap baris, memberitahukan pengguna bahasa apa yang digunakan. Setiap baris harus diakhiri dengan LF (baris baru) - jika `cat myfile` tidak berakhir dengan suatu baris blank (kosong) maka baris terakhir tersebut tidak akan diproses.

```
#!/bin/sh
# Nama file skrip: while3a.sh
while read f
do
  case $f in
        hello)          echo English    ;;
        howdy)          echo American   ;;
        gday)           echo Australian ;;
        bonjour)        echo French     ;;
        "guten tag")    echo German     ;;
        "apa kabar")    echo Indonesian ;;
        *)              echo Unknown Language: $f
                ;;
   esac
done < myfile
```

Pada banyak sistem Linux, ini dapat pula dilakukan dengan:

```
#!/bin/sh
# Nama file skrip: while3b.sh
while f=`line`
do
  .. proses f ..
done < myfile
```

Karena `while read f` bekerja hanya dengan Linux dan tidak bergantung pada program eksternal `line` maka bentuk pada while3a.sh lebih disukai. Mengapa metode ini menggunakan backtick (`)?

Banyak programmer lebih menyukai $i (bukan $f) sebagai default untuk perulangan. Berikut ini adalah contoh eksekusi dari skrip while3a.sh:

```
$ i=THIS_IS_A_BUG
$ export i
$ ./while3a.sh something
Unknown Language: THIS_IS_A_BUG
```

Hindari kesalahan ketik. Lebih baik menggunakan `${x}`, bukan hanya `$x`. Jika `x="A"` dan anda ingin mengatakan "A1" maka anda perlu `echo ${x}1`, sedangkan `echo $x1` akan mencoba mengakses variabel `x1` yang sebenarnya tidak ada.

Ada hal yang sangat menarik pada shell Bash. Pembuatan banyak direktori dengan nama hampir sama dapat dilakukan dengan mudah. Misalnya membuat 8 direktori berawalan "rc" kemudian dilanjutkan suatu angka atau huruf dan diakhir dengan ".d", dapat ditulis:

```
mkdir rc{0,1,2,3,4,5,6,S}.d
```

lebih sederhana dari pada anda menulis:

```
for runlevel in 0 1 2 3 4 5 6 S
do
  mkdir   rc${runlevel}.d
done
```

Perhatikan cara rekursif penggunaan perintah ls berikut. Kita dapat menampilkan daftar file dari banyak direktori sekaligus.

```
$ cd /
$ ls -ld {,usr,usr/local}/{bin,sbin,lib}
drwxr-xr-x     2 root     root          4096 Oct 26 01:00 /bin
drwxr-xr-x     6 root     root          4096 Jan 16 17:09 /lib
drwxr-xr-x     2 root     root          4096 Oct 27 00:02 /sbin
drwxr-xr-x     2 root     root         40960 Jan 16 19:35 usr/bin
drwxr-xr-x    83 root     root         49152 Jan 16 17:23 usr/lib
drwxr-xr-x     2 root     root          4096 Jan 16 22:22 usr/local/bin
drwxr-xr-x     3 root     root          4096 Jan 16 19:17 usr/local/lib
drwxr-xr-x     2 root     root          4096 Dec 28 00:44 usr/local/sbin
drwxr-xr-x     2 root     root          8192 Dec 27 02:10 usr/sbin
```

Perulangan while sering digunakan bersama dengan seleksi kondisi memanfaatkan perintah `test` dan `case`.

III.2 Seleksi Kondisi

Test

Test digunakan secara virtual oleh setiap skrip shell yang ditulis. Test memang tidak sering dipanggil secara langsung. Test sering dipanggil sebagai `[`. `[` adalah link simbolik ke perintah test, membuat program shell lebih nyaman dibaca.

Coba tulis perintah di bawah ini dan perhatikan luarannya:

```
$ type [
[ is a shell builtin

$ which [
/usr/bin/[

$ ls -l /usr/bin/[
lrwxrwxrwx 1 root root 4 Mar 27 2000 /usr/bin/[ -> test
```

Ini berarti bahwa '[' sebenarnya adalah sebuah program yang sudah built-in seperti `ls`. Karena termasuk perintah maka saat penggunaan '[' harus diberi jarak (spasi) saat digunakan bersama string atau perintah lain.

Seleksi kondisi seperti ini:

```
if [$foo == "bar" ]
```

tidak akan bekerja. Baris tersebut diterjemahkan sebagai `if test$foo == "bar" ]`, dimana terdapat ']' tanpa '[' permulaan. Letakkan spasi di awal dan akhir semua operator. Jika anda menjumpai kata 'SPACE' di dalam panduan ini, berarti anda diminta menggatinya dengan spasi. Jika tidak ada spasi maka skrip shell tidak bekerja:

```
if SPACE [ SPACE "$foo" SPACE == SPACE "bar" SPACE ]
```

Test merupakan tool perbandingan yang sederhana tetapi powerful. Gunakan perintah `man test` untuk mengetahui manual lengkap dari perintah test.

Test sering dipanggil secara tak langsung melalui pernyataan `if` dan `while`. Sintaks untuk `if...then...else...` adalah

```
if [ ... ]
then
  #   if-code
else
  #   else-code
fi
```

Pernyataan `fi` merupakan kebalikan dari `if`. Bentuk seperti ini juga digunakan pada `case` dan `esac`. Anda juga perlu waspadai sintaks - perintah "`if [ ... ]`" dan "`then`" harus pada baris-baris berbeda. Sebagai alternatif, titik-koma "`;`" dapat digunakan untuk memisahkannya:

```
if [ ... ]; then
  #  lakukan  sesuatu
fi
```

Anda juga dapat menggunakan `elif`, seperti ini:

```
if  [ something ]; then
 echo "Something"
 elif  [  something_else  ];  then
   echo "Something else"
 else
   echo "None of the above"
fi
```

Skrip di atas akan meng-echo "Something" jika test terhadap `[ something ]` berhasil, jika tidak maka dilakukan test terhadap `[ something_else ]` dan memroses echo "Something else" jika berhasil. Jika semuanya gagal, maka akan menjalankan echo "None of the above".

Coba praktekkan potongan kode berikut. Sebelum menjalankannya set variabel X untuk beberapa nilai (coba -1, 0, 1, hello, bye, dan lain-lain). Anda dapat melakukan ini sebagai berikut:

```
$ X=5
$  export  X
$ ./test.sh
  ... output of test.sh ... $
X=hello
$ ./test.sh
  ... output of test.sh ... $
X=test.sh
$ ./test.sh
  ... output of test.sh ...
```

Kemudian coba lagi, dengan `$X` sebagai nama dari file yang ada, misalnya `/etc/hosts`.

```
#!/bin/sh
#  Nama  file  skrip:  test.sh
if [ "$X" -lt "0" ]
then
  echo "X is less than zero"
fi
if [ "$X" -gt "0" ]; then
  echo  "X  is  more  than  zero"
fi
[ "$X" -le "0" ] && \
      echo "X is less than or equal to  zero" [
"$X" -ge "0" ] && \
      echo "X is more than or equal to zero" [
"$X" = "0" ] && \
      echo "X is the string or number \"0\"" [
"$X" = "hello" ] && \
      echo "X matches the string \"hello\"" [
"$X" != "hello" ] && \
      echo "X is not the string \"hello\"" [
-n "$X" ] && \
      echo "X is of nonzero length"
[ -f "$X" ] && \
      echo  "X  is  the  path  of  a  real  file"  ||  \
      echo "No such file: $X"
[ -x "$X" ] && \
      echo "X is the path of an executable file" [
"$X" -nt "/etc/passwd" ] && \
      echo "X is a file which is newer than /etc/passwd"
```

Perlu diketahui bahwa kita dapat menggunakan titik-koma (;) untuk menggabungkan dua baris (menjadi satu baris). Ini sering dilakukan untuk menghemat ruang dalam pernyataan if yang sederhana. Karakter *backslash* memberitahukan shell bahwa ini bukanlah akhir baris, tetapi dua atau lebih baris agar diperlakukankan sebagai satu baris. Ini berguna untuk memudahkan pembacaan kode. Biasanya baris berikutnya diinden (agak masuk ke dalam).

Perintah `test` dapat mengerjakan banyak pengujian terhadap bilangan, string dan nama file.

Perlu diketahui bahwa parameter `-a`, `-e` (berarti "file exists"), `-S` (file is a Socket), `-nt` (file is newer than), `-ot` (file is older than), `-ef` (paths refer to the same file) dan `-O` (file is owned my user) tidak tersedia pada shell Bash lama (seperti /bin/sh pada Solaris, AIX, HPUX, dan lain-lain).

Ada cara yang lebih mudah dalam menuliskan pernyataan `if`: Perintah `&&` dan `||` menyebabkan kode tertentu dijalankan jika `test` bernilai True.

```
#!/bin/bash
[ $X -ne 0 ] && echo "X isn't zero" || echo "X is zero"
[ -f $X ] && echo "X is a file" || echo "X is not a file" [
-n $X ] && echo "X is of non-zero length" || \
     echo "X is of zero length"
```

Perlu dicatat bahwa ketika anda mengubah nilai X ke suatu nilai non-numerik, beberapa perbandingan awal memberikan pesan berikut:

```
test.sh: [: integer expression expected before -lt
test.sh: [: integer expression expected before -gt
test.sh: [: integer expression expected before -le
test.sh: [: integer expression expected before -ge
```

Ini karena -lt, -gt, -le, -ge adalah perbandingan dirancang hanya untuk integer dan tidak berjalan terhadap string. Perbandingan string seperti `!=` akan memperlakukan "5" sebagai suatu string, tetapi tidak dapat memperlakukan "Hello" sebagai suatu integer, sehingga perbandingan integer bermasalah. Jika anda ingin skrip shell berjalan lebih baik, maka periksa terlebih dahulu isi variabel sebelum dilakukan test - mungkin menjadi seperti ini:

```
echo -en "Please guess the magic number: "
read X
echo $X | grep "[^0-9]" > /dev/null 2>&1
if [ "$?" -eq "0" ]; then
  # If the grep found something other than 0-9 #
  then it's not an integer.
  echo "Sorry, wanted a number"
else
  # The grep found only 0-9, so it's an integer. #
  We can safely do a test on it.
  if [ "$X" == "7" ]; then
    echo "You entered the magic number!"
  fi
fi
```

Pada cara ini anda dapat meng-`echo` pesan yang lebih bermakna kepada pengguna dan keluar (exit). Variabel `$?` akan dijelaskan pada bagian lain panduan ini dan perintah `grep` perlu dipelajari lebih lanjut. Perintah `grep [0-9]` mencari baris teks yang mengandung digit (0-9). Tanda caret (^) dalam `grep [^0-9]` hanya akan mencari baris yang hanya

tidak mengandung bilangan.

Apa yang dihasilkan perintah `grep -v [0-9]`?

Perintah test dapat digunakan dalam perulangan while. Contohnya adalah:

```
#!/bin/sh
# Nama file skrip: test2.sh
X=0
while [ -n "$X" ]
do
  echo "Masukkan teks (ENTER untuk keluar)"
  read X
  echo "Anda menulis: $X"
done
```

Kode ini akan terus meminta input sampai anda menekan ENTER (X panjangnya nol). Bagaimana jika `while [ -n "$X" ]` diganti dengan `while [ -n $X ]`, dengan menghilangkan dua tanda petik gandanya?

Perhatikan eksekusi di bawah ini:

```
$ ./test2.sh
Masukkan teks (ENTER untuk keluar)
fred
Anda menulis: fred
Masukkan teks (ENTER untuk keluar)
wilma
Anda menulis: wilma
Masukkan teks (ENTER untuk keluar)
Anda menulis:
$
```

Kode di atas dapat dibuat lebih rapi dengan menyertakan test di dalam loop:

```
#!/bin/sh
# Nama file skrip: test3.sh
X=0
while [ -n "$X" ]
do
  echo "Enter Masukkan teks (ENTER untuk keluar)"
  read X
  if [ -n "$X" ]; then
    echo " Anda menulis: $X"
  fi
done
```

Anda telah menggunakan dua sintaks berbeda untuk pernyataan `if` di atas, yaitu:

```
if [ "$X" -lt "0" ]
then
  echo "X lebih kecil daripada nol"
fi
.......... dan..........
if [ ! -n "$X" ]; then
  echo "You said: $X" fi
```

Case

Pernyataan `case` menghemat pernyataan `if .. then .. else` banyak baris. Sintaksnya sangat sederhana:

```
#!/bin/sh
# Nama file skrip: talk.sh
echo "Please talk to me ..."
while :
do
  read INPUT_STRING
  case  $INPUT_STRING  in
        hello)
                echo  "Hello  yourself!"
                ;;
        bye)
                echo  "See  you  again!"
                break
                ;;
        *)
                echo  "Sorry,  I  don't  understand"
                ;;
  esac
done
echo
echo "That's all folks!"
```

Coba jalankan kode di atas dan perhatikan bagaimana kerjanya...

```
$ ./talk.sh
Please talk to me ...
hello
Hello yourself!
What do you think of politics?
Sorry, I don't understand
bye
See you again!

That's all folks!
```

Baris `case` sendiri selalu berformat sama. Dimulai dengan memeriksa nilai suatu variabel kondisi, pada contoh di atas adalah variabel `INPUT_STRING`.

Pilihan-pilihan kemudian didaftarkan dan diikuti oleh suatu kurung tutup seperti `hello)` dan `bye)`. Ini berarti bahwa jika `INPUT_STRING` cocok dengan `hello` maka bagian kode itu dieksekusi, sampai dengan dua titik-koma. Jika `INPUT_STRING` cocok dengan `bye` maka pesan "goodbye" dicetak dan keluar dari perulangan. Jika anda ingin keluar dari skrip maka dapat menggunakan perintah `exit`, bukan `break`.

Pilihan ketiga, di sini `*)`, merupakan kondisi *catch-all* default; tidak harus tetapi sering berguna untuk tujuan debugging bahkan jika kita mengetahui nilai apa yang akan dimiliki variabel dalam test.

Pernyataan `case` lengkap diakhiri dengan `esac` dan loop `while` diakhiri dengan `done`.