Pengantar Pemrograman Bash Shell di Linux

From OnnoWiki

Sumber: http://pemula.linux.or.id/programming/bash-shell.html

Contents

Pendahuluan

Apa itu shell ? shell adalah program (penterjemah perintah) yang menjembatani user dengan sistem operasi dalam hal ini kernel (inti sistem operasi), umumnya shell menyediakan prompt sebagai user interface, tempat dimana user mengetikkan perintah-perintah yang diinginkan baik berupa perintah internal shell (internal command), ataupun perintah eksekusi suatu file progam (eksternal command), selain itu shell memungkinkan user menyusun sekumpulan perintah pada sebuah atau beberapa file untuk dieksekusi sebagai program.

Macam - macam shell?

Tidak seperti sistem operasi lain yang hanya menyediakan satu atau 2 shell, sistem operasi dari keluarga unix misalnya linux sampai saat ini dilengkapi oleh banyak shell dengan kumpulan perintah yang sangat banyak, sehingga memungkinkan pemakai memilih shell mana yang paling baik untuk membantu menyelesaikan pekerjaannya, atau dapat pula berpindah-pindah dari shell yang satu ke shell yang lain dengan mudah, beberapa shell yang ada di linux antara lain:

- Bourne shell(sh),
- C shell(csh),
- Korn shell(ksh),
- Bourne again shell(bash),
- dsb.

Masing - masing shell mempunyai kelebihan dan kekurangan yang mungkin lebih didasarkan pada kebutuhan pemakai yang makin hari makin meningkat, untuk dokumentasi ini shell yang digunakan adalah bash shell dari GNU, yang merupakan pengembangan dari Bourne shell dan mengambil beberapa feature (keistimewaan) dari C shell serta Korn shell, Bash shell merupakan shell yang cukup banyak digunakan pemakai linux karena kemudahan serta banyaknya fasilitas perintah yang disediakan.versi bash shell yang saya gunakan adalah 2.04

```
[fajar@linux$]echo $BASH_VERSION
bash 2.04.12(1)-release
```

Mungkin saat anda membaca dokumentasi ini versi terbaru dari bash sudah dirilis dengan penambahan feature yang lain.

Pemrograman Shell ?

Yaitu menyusun atau mengelompokkan beberapa perintah shell (internal atupun eksternal command) menjadi kumpulan perintah yang melakukan tugas tertentu sesuai tujuan penyusunnya. Kelebihan shell di linux dibanding sistem operasi lain adalah bahwa shell di linux memungkinkan kita untuk menyusun serangkaian perintah seperti halnya bahasa pemrograman (interpreter language), melakukan proses I/O, menyeleksi kondisi, looping, membuat fungsi, dsb. adalah proses - proses yang umumnya dilakukan oleh suatu bahasa pemrograman, jadi dengan shell di linux kita dapat membuat program seperti halnya bahasa pemrograman, untuk pemrograman shell pemakai unix atau linux menyebutnya sebagai script shell.

Kebutuhan Dasar

Sebelum mempelajari pemrograman Bash shell di linux sebaiknya anda telah mengetahui dan menggunakan perintah - perintah dasar shell baik itu internal command yang telah disediakan shell maupun eksternal command atau utility, seperti

- cd, pwd, times, alias, umask, exit, logout, fg, bg, ls, mkdir, rmdir, mv, cp, rm, clear, ...
- utilitas seperti cat, cut, paste, chmod, lpr,...
- redirection (cara mengirim output ke file atau menerima input dari file), menggunakan operator redirect >, >>, <, <<, contohnya:

```
ls > data
hasil ls dikirim ke file data, jika file belum ada akan dibuat tetapi jika sudah ada isinya akan ditimpa.
```

```
ls >> data
hampir sama, bedanya jika file sudah ada maka isinya akan ditambah di akhir file.
```

```
cat < data
file data dijadikan input oleh perintah cat
```

- pipa (output suatu perintah menjadi input perintah lain), operatornya : | , contoh:

```
ls -l | sort -s
ouput perintah ls -l (long) menjadi input perintah sort -s (urutkan secara descending), mending pake ls -
```

```
ls -l | sort -s | more
```

```
cat  databaru
```

- Wildcard dengan karakter *, ?, [ ], contohnya:

```
ls i*
tampilkan semua file yang dimulai dengan i
```

```
ls i?i
tampilkan file yang dimulai dengan i, kemudian sembarang karakter tunggal, dan diakhiri dengan i
```

```
ls [ab]*
tampilkan file yang dimulai dengan salah satu karakter a atau b
```

Simple Bash Script

Langkah awal sebaiknya periksa dulu shell aktif anda, gunakan perintah ps (report process status)

```
[fajar@linux$]ps
 PID TTY          TIME CMD
  219 tty1     00:00:00 bash
  301 tty1     00:00:00 ps
```

bash adalah shell aktif di system saya, jika disystem anda berbeda misalnya csh atau ksh ubahlah dengan perintah change shell

```
[fajar@linux$]chsh
Password:
New shell [/bin/csh]:/bin/bash
Shell changed
```

atau dengan mengetikkan bash

```
[fajar@linux$]bash
```

sekarang coba anda ketikkan perintah dibawah ini pada prompt shell

```
echo "Script shell pertamaku di linux"
```

```
[fajar@linux$]echo "Script shell pertamaku di linux"
Script shell pertamaku di linux
```

string yang diapit tanda kutip ganda (double quoted) akan ditampilkan pada layar anda, echo adalah statement (perintah) built-in bash yang berfungsi menampilkan informasi ke standard output yang defaultnya adalah layar. jika diinginkan mengulangi proses tersebut, anda akan mengetikkan kembali perintah tadi, tapi dengan fasilitas history cukup menggunakan tombol panah kita sudah dapat mengulangi perintah tersebut, bagaimana jika berupa kumpulan perintah yang cukup banyak, tentunya dengan fasilitas hirtory kita akan kerepotan juga mengulangi perintah yang diinginkan apalagi jika selang beberapa waktu mungkin perintah-perintah tadi sudah tertimpa oleh perintah lain karena history mempunyai kapasitas penyimpanan yang ditentukan. untuk itulah sebaiknya perintah-perintah tsb disimpan ke sebuah file yang dapat kita panggil kapanpun diinginkan.

coba ikuti langkah - langkah berikut:

- Masuk ke editor anda, apakah memakai vi,pico,emacs,dsb...
- ketikkan perintah berikut

```
#!/bin/bash
echo "Hello, apa khabar"
```

- simpan dengan nama file tes
- ubah permission file tes menggunakan chmod

```
[fajar@linux$]chmod 755 tes
```

- jalankan

```
[fajar@linux$]./tes
```

kapan saja anda mau mengeksekusinya tinggal memanggil file tes tersebut, jika diinginkan mengeset direktory kerja anda sehingga terdaftar pada search path ketikkan perintah berikut

```
PATH=$PATH:.
```

setelah itu script diatas dapat dijalankan dengan cara

```
[fajar@linux$]tes
Hello, apa khabar
```

tanda #! pada /bin/bash dalam script tes adalah perintah yang diterjemahkan ke kernel linux untuk mengeksekusi path yang disertakan dalam hal ini program bash pada direktory /bin, sebenarnya tanpa mengikutkan baris tersebut anda tetap dapat mengeksekusi script bash, dengan catatan bash adalah shell aktif. atau dengan mengetikkan bash pada prompt shell.

```
[fajar@linux$]bash tes
```

tentunya cara ini kurang efisien, menyertakan path program bash diawal script kemudian merubah permission file sehingga dapat anda execusi merupakan cara yang paling efisien.

Sekarang coba kita membuat script shell yang menampilkan informasi berikut:

1. Waktu system
2. Info tentang anda
3. jumlah pemakai yang sedang login di system

contoh scriptnya:

```
#!/bin/bash
#myinfo

#membersihkan tampilan layar
clear

#menampilkan informasi
echo -n "Waktu system   :"; date
echo -n "Anda           :"; whoami
echo -n "Banyak pemakai :"; who | wc -l
```

sebelum dijalankan jangan lupa untuk merubah permission file myinfo sehingga dapat dieksekusi oleh anda

```
[fajar@linux$]chmod 755 myinfo
[fajar@linux$]./myinfo
Waktu system   : Sat Nov 25  22:57:15 BORT 2001
Anda           : fajar
Banyak pemakai : 2
```

tentunya layout diatas akan disesuaikan dengan system yang anda gunakan statement echo dengan opsi -n akan membuat posisi kursor untuk tidak berpindah ke baris baru karena secara default statement echo akan mengakhiri proses pencetakan ke standar output dengan karakter baris baru (newline), anda boleh mencoba tanpa menggunakan opsi -n, dan lihat perbedaannya. opsi lain yang dapat digunakan adalah -e (enable), memungkinkan penggunaan backslash karakter atau karakter sekuen seperti pada bahasa C atau perl, misalkan :

```
echo –e "\abunyikan bell"
```

jika dijalankan akan mengeluarkan bunyi bell, informasi opsi pada statement echo dan backslash karakter selengkapnya dapat dilihat via man di prompt shell.

```
[fajar@linux$]man echo
```

Pemakaian Variabel

Secara sederhana variabel adalah pengenal (identifier) berupa satuan dasar penyimpanan yang isi atau nilainya sewaktu-waktu dapat berubah baik oleh eksekusi program (runtime program) ataupun proses lain yang dilakukan sistem operasi. dalam dokumentasi ini saya membagi variabel menjadi 3 kategori:

1. Environment Variable
2. Positional Parameter
3. User Defined Variable

Environment Variable

atau variabel lingkungan yang digunakan khusus oleh shell atau system linux kita untuk proses kerja system seperti variabel PS1, PS2, HOME, PATH, USER, SHELL,dsb...jika digunakan akan berdampak pada system, misalkan variabel PS1 yang digunakan untuk mengeset prompt shell pertama yaitu prompt tempat anda mengetikkan perintah - perintah shell (defaultnya "\s-\v\$"), PS2 untuk prompt pelengkap perintah, prompt ini akan ditampilkan jika perintah yang dimasukkan dianggap belum lengkap oleh shell (defaultnya ">"). anda dapat mengeset PS1 dan PS2 seperti berikut.

simpan dahulu isi PS1 asli system anda, sehingga nanti dapat dengan mudah dikembalikan

```
[fajar@linux$]PS1LAMA=$PS1
```

sekarang masukkan string yang diinginkan pada variabel PS1

```
[fajar@linux$]PS1="Hi ini Promptku!"
Hi ini Promptku!PS2="Lengkapi dong ? "
```

maka prompt pertama dan kedua akan berubah, untuk mengembalikan PS1 anda ke prompt semula ketikkan perintah

```
[fajar@linux$]PS1=$PS1LAMA
```

Jika anda ingin mengkonfigurasi prompt shell, bash telah menyediakan beberapa backslash karakter diantaranya adalah:

```
\a      ASCII bell character (07)
\d      date dengan format "Weekday Month Date" (misalnya "Tue May 26")
\e      ASCII escape character (033)
\H      hostname (namahost)
\n      newline (karakter baru)
\w      Direktory aktif
\t      time dalam 24 jam dengan format HH:MM:SS
dll     man bash:-)
```

contoh pemakaiannya:

```
[fajar@linux$]PS1="[\t][\u@\h:\w]\$"
```

agar prompt shell hasil konfigurasi anda dapat tetap berlaku (permanen) sisipkan pada file .bashrc atau .profile

Positional Parameter

atau parameter posisi yaitu variabel yang digunakan shell untuk menampung argumen yang diberikan terhadap shell baik berupa argumen waktu sebuah file dijalankan atau argumen yang dikirim ke subrutin. variabel yang dimaksud adalah 1,2,3,dst..lebih jelasnya lihat contoh script berikut :

```
#!/bin/bash
#argumen1

echo $1 adalah salah satu $2 populer di $3
```

Hasilnya

```
[fajar@linux$]./argumen1 bash shell linux
bash adalah salah satu shell populer di linux
```

ada 3 argumen yang disertakan pada script argumen1 yaitu bash, shell, linux, masing2 argumen akan disimpan pada variabel 1,2,3 sesuai posisinya. variabel spesial lain yang dapat digunakan diperlihatkan pada script berikut:

```
#!/bin/bash
#argumen2

clear
echo "Nama script anda : $0";
echo "Banyak argumen   : $#";
echo "Argumennya adalah: $*";
```

Hasilnya:

```
[fajar@linux$]./argumen 1 2 3 empat
Nama script anda  : ./argumen
Banyak argumen    : 4
Argumennya adalah : 1 2 3 empat
```

User Defined Variable

atau variabel yang didefinisikan sendiri oleh pembuat script sesuai dengan kebutuhannya, beberapa hal yang perlu diperhatikan dalam mendefenisikan variabel adalah:

- dimulai dengan huruf atau underscore
- hindari pemakaian spesial karakter seperti *,$,#,dll...
- bash bersifat case sensitive, maksudnya membedakan huruf besar dan kecil, a berbeda dengan A, nama berbeda dengan Nama,NaMa,dsb..

untuk mengeset nilai variabel gunakan operator assignment (pemberi nilai)"=", contohnya :

```
myos="linux"        #double-quoted
nama='pinguin'      #single-quoted
hasil=`ls -l`;      #back-quoted
angka=12
```

kalau anda perhatikan ada 3 tanda kutip yang kita gunakan untuk memberikan nilai string ke suatu variabel, adapun perbedaannya adalah:

- dengan kutip ganda (double-quoted), bash mengizinkan kita untuk menyisipkan variabel di dalamnya. contohnya:

```
#!/bin/bash

nama="pinguin"
kata="Hi $nama, apa khabarmu"    #menyisipkan variabel nama
echo $kata;
```

Hasilnya:

```
Hi pinguin, apa khabarmu
```

- dengan kutip tunggal (single-quoted), akan ditampilkan apa adanya. contohnya:

```
#!/bin/bash

nama="pinguin"
kata='Hi $nama, apa khabarmu'    #menyisipkan variabel nama
echo $kata;
```

Hasilnya:

```
Hi $nama, apa khabarmu
```

- dengan kutip terbalik (double-quoted), bash menerjemahkan sebagai perintah yang akan dieksekusi, contohnya:

```
#!/bin/bash

hapus=`clear`;
isi=`ls -l`;        #hasil dari perintah ls -l disimpan di variabel isi

#hapus layar
echo $hapus

#ls -l
echo $isi;
```

Hasilnya: silahkan dicoba sendiri

Untuk lebih jelasnya lihat contoh berikut:

```
#!/bin/bash
#varuse

nama="fajar"
OS='linux'
distro="macam-macam, bisa slackware,redhat,mandrake,debian,suse,dll"
pc=1
hasil=`ls -l $0`

clear
echo -e "Hi $nama,\npake $OS\nDistribusi, $distro\nkomputernya, $pc buah"
echo "Hasil ls -l $0 adalah =$hasil"
```

Hasilnya:

```
[fajar@linux$]./varuse
Hi fajar,
pake linux Distribusi, macam-macam, bisa slackware,redhat,mandrake,debian,suse,dll
komputernya, 1 buah
Hasil ls -l ./varuse adalah -rwxr-xr-x 1 fajar users 299 Nov 21 06:24 ./varuse
```

untuk operasi matematika ada 3 cara yang dapat anda gunakan, dengan statement builtin let atau expr atau perintah subtitusi seperti contoh berikut:

```
#mat1

a=10
b=5
#memakai let
jumlah=$(($a+$b))
kurang=$(($a-$b))
kali=$(($a*$b))

#memakai expr
bagi=`expr $a / $b`

#memakai perintah subtitusi $((ekspresi))
```

```
modul=$(($a%$b))  #sisa pembagian

echo "$a+$b=$jumlah"
echo "$a-$b=$kurang"
echo "$a*$b=$kali"
echo "$a/$b=$bagi"
echo "$a%$b=$modul"
```

Hasilnya:

```
[fajar@linux$]./mat1
10+5=15
10-5=5
10*5=50
10/5=2
10%5=0
```

fungsi expr begitu berdaya guna baik untuk operasi matematika ataupun string contohnya:

```
[fajar@linux$]mystr="linux"
[fajar@linux$]expr length $mystr
5
```

Mungkin anda bertanya - tanya, apakah bisa variabel yang akan digunakan dideklarasikan secara eksplisit dengan tipe data tertentu?, mungkin seperti C atau pascal, untuk hal ini oleh Bash disediakan statement declare dengan opsi -i hanya untuk data integer (bilangan bulat). Contohnya:

```
#!/bin/bash

declare -i angka
angka=100;
echo $angka;
```

apabila variabel yang dideklarasikan menggunakan declare -i ternyata anda beri nilai string (karakter), maka Bash akan mengubahnya ke nilai 0, tetapi jika anda tidak menggunakannya maka dianggap sebagai string.

Simple I/O

I/O merupakan hal yang mendasar dari kerja komputer karena kapasitas inilah yang membuat komputer begitu berdayaguna. I/O yang dimaksud adalah device yang menangani masukan dan keluaran, baik itu berupa keyboard, floppy, layar monitor,dsb. sebenarnya kita telah menggunakan proses I/O ini pada contoh - contoh diatas seperti statement echo yang menampilkan teks atau informasi ke layar, atau operasi redirect ke ke file. selain echo, bash menyediakan perintah builtin printf untuk mengalihkan keluaran ke output standard, baik ke layar ataupun ke file dengan format tertentu, mirip statement printf kepunyaan bahasa C atau perl. berikut contohnya:

Output dengan printf

```
#!/bin/bash
#pr1
```

```
url="pemula.linux.or.id";
angka=32;

printf "Hi, Pake printf ala C\n\t\a di bash\n";
printf "My url %s\n %d decimal = %o octal\n" $url $angka $angka;
printf "%d decimal dalam float = %.2f\n" $angka $angka
```

Hasilnya:

```
[fajar@linux$]./pr1
Hi, Pake printf ala C
    di bash
My url  pemula.linux.or.id
32 decimal = 40 octal
32 decimal dalam float = 32.00
```

untuk menggunakan format kontrol sertakan simbol %, bash akan mensubtitusikan format tsb dengan isi variabel yang berada di posisi kanan sesuai dengan urutannya jika lebih dari satu variabel, \n \t \a adalah karakter sekuen lepas newline,tab, dan bell,

Format control keterangan

```
%d      untuk format data integer
%o      octal
%f      float atau decimal
%x      Hexadecimal
```

pada script diatas %.2f akan mencetak 2 angka dibelakang koma, defaultnya 6 angka, informasi lebih lanjut dapat dilihat via man printf

Input dengan read

Setelah echo dan printf untuk proses output telah anda ketahui, sekarang kita menggunakan statement read yang cukup ampuh untuk membaca atau menerima masukan dari input standar

syntax :

```
read -opsi [nama_variabel...]
```

berikut contoh scriptnya:

```
#!/bin/bash
#rd1

echo -n "Nama anda :"
read nama;

echo    "Hi $nama,  apa khabarmu";
echo    "Pesan dan kesan :";
read
echo    "kata $nama, $REPLY";
```

Hasilnya:

```
[fajar@linux$]./rd1
Nama anda : pinguin
Hi pinguin, apa khabarmu
Pesan & kesan :
 pake linux pasti asyk – asyk aja
kata pinguin, pake linux pasti asyk – asyk aja
```

jika nama_variabel tidak disertakan, maka data yang diinput akan disimpan di variabel REPLY contoh lain read menggunakan opsi

```
–t(TIMEOUT), –p (PROMPT), –s(SILENT), –n (NCHAR) dan –d(DELIM)
```

```
#!/bin/bash

read –p "User Name : " user
echo –e "Password 10 karakter,\njika dalam 6 second tidak dimasukkan pengisian password diakhiri"
read –s –n 10 –t 6 pass
echo    "kesan anda selama pake linux, _underscore=>selesai"
read –d _ kesan

echo    "User = $user"
echo    "Password = $pass"
echo    "Kesan selama pake linux = $kesan"
```

Hasilnya: silahkan dicoba sendiri :-)

Opsi Keterangan

```
–p      memungkinkan kita membuat prompt sebagai informasi pengisian
–s      membuat input yang dimasukkan tidak di echo ke layar (seperti layaknya password di linux)
–n      menentukan banyak karakter yang diinput
–d      menentukan karakter pembatas masukan
```

informasi secara lengkap lihat man bash

Output dengan konstanta ANSI

Pengaturan Warna

Untuk pewarnaan tampilan dilayar anda dapat menggunakan konstanta ANSI (salah satu badan nasional amerika yang mengurus standarisasi).

syntaxnya:

```
\033[warnam
```

Dimana:

```
m menandakan setting color
```

contohnya:

```
[fajar@linux$]echo –e "\033[31m HELLO\033[0m"
HELLO
```

konstanta 31m adalah warna merah dan 0m untuk mengembalikan ke warna normal (none), tentunya konstanta warna ansi ini dapat dimasukkan ke variabel PS1 untuk mengatur tampilan prompt shell anda, contohnya:

```
[fajar@linux$]PS1="\033[34m"
[fajar@linux$]
```

berikut daftar warna yang dapat anda gunakan:

```
foreground
      None    0m
      Black       0;30     Dark Gray     1;30
      Red         0;31     Light Red     1;31
      Green       0;32     Light Green   1;32
      Brown       0;33     Yellow        1;33
      Blue        0;34     Light Blue    1;34
      Purple      0;35     Light Purple  1;35
      Cyan        0;36     Light Cyan    1;36
      Light Gray  0;37     White         1;37
```

background

```
      dimulai dengan 40 untuk BLACK,41 RED,dst
```

lain-lain

```
      4 underscore,5 blink, 7 inverse
```

tentunya untuk mendapatkan tampilan yang menarik anda dapat menggabungkannya antara foreground dan background

```
[fajar@linux$]echo –e "\033[31;1;33m Bash and ansi color\033[0m"
```

Bash and ansi color

Pengaturan posisi kursor

sedangkan untuk penempatan posisi kursor, dapat digunakan salah satu cara dibawah.

- Menentukan posisi baris dan kolom kursor:

```
\033[baris;kolomH
```

- Pindahkan kursor keatas N baris:

```
\033[NA
```

- Pindahkan kursor kebawah N baris:

```
\033[NB
```

- Pindahkan kursor kedepan N kolom:

```
\033[NC
```

- Pindahkan kursor kebelakang N kolom:

```
\033[ND
```

Contohnya:

```
#!/bin/bash

SETMYCOLOR="\033[42;1;37m"
GOTOYX="\033[6;35H"
clear
echo –e "\033[3;20H INI DIBARIS 3, KOLOM 20"
echo –e "\033[44;1;33;5m\033[5;35H HELLO\033[0m";
echo –e "$SETMYCOLOR$GOTOYX ANDA LIHAT INI\033[0m"
```

Hasilnya: Silahkan dicoba sendiri Menggunakan utulity tput untuk penempatan posisi kursor

kita dapat pula mengatur penempatan posisi kursor di layar dengan memanfaatkan utility tput,

syntaxnya:

```
tput cup baris kolom
```

contohnya:

```
#!/bin/bash

clear
tput cup 5 10
echo  "HELLO"
tput cup 6 10
echo  "PAKE TPUT"
```

jika dijalankan anda akan mendapatkan string HELLO di koordinat baris 5 kolom 10, dan string PAKE TPUT dibaris 6 kolom 10. informasi selengkapnya tentang tput gunakan man tput, atau info tput

Seleksi dan Perulangan

Bagian ini merupakan ciri yang paling khas dari suatu bahasa pemrograman dimana kita dapat mengeksekusi suatu pernyataan dengan kondisi terntentu dan mengulang beberapa pernyataan dengan kode script yang cukup singkat.

test dan operator

test adalah utility sh shell yang berguna untuk memeriksa informasi tentang suatu file dan berguna untuk melakukan perbandingan suatu nilai baik string ataupun numerik

```
syntaxnya: test ekspresi
```

proses kerja test yaitu dengan mengembalikan sebuah informasi status yang dapat bernilai 0 (benar) atau 1 (salah) dimana nilai status ini dapat dibaca pada variabel spesial $?.

```
[fajar@linux$]test 5 -gt 3
[fajar@linux$]echo $?
0
```

pernyataan 5 -gt (lebih besar dari) 3 yang dievaluasi test menghasilkan 0 pada variabel status $? itu artinya pernyataan tersebut benar tetapi coba anda evaluasi dengan expresi berikut

```
[fajar@linux$]test 3 -lt 1
[fajar@linux$]echo $?
1
```

status bernilai 1, berarti pernyataan salah.

anda lihat simbol -gt dan -lt, itulah yang disebut sebagai operator, secara sederhana operator adalah karakter khusus (spesial) yang melakukan operasi terhadap sejumlah operand, misalkan 2+3, "+" adalah operator sedangkan 2 dan 3 adalah operandnya, pada contoh test tadi yang bertindak sebagai oparatornya adalah -lt dan -gt, sedangkan bilangan disebelah kiri dan kanannya adalah operand. cukup banyak operator yang disediakan bash antara lain:

Operator untuk integer

```
Operator        Keterangan
bil1 -eq bil2   Mengembalikan Benar jika bil1 sama dengan bil2
bil1 -ne bil2   -||- Benar jika bil1 tidak sama dengan bil2
bil1 -lt bil2   -||- Benar jika bil1 lebih kecil dari bil2
bil1 -le bil2   -||- Benar jika bil1 lebih kecil atau sama dengan bil2
bil1 -gt bil2   -||- Benar jika bil1 lebih besar dari bil2
bil1 -ge bil2   -||- Benar jika bil1 lebih besar atau sama dengan bil2
```

Operasi string

```
Operator        Keterangan
-z STRING       Mengembalikan Benar jika panjang STRING adalah zero
STRING1 == STRING2      -||- Benar jika STRING1 sama dengan STRING2
```

Operator file

```
Operator        Keterangan
-f FILE         Mengembalikan Benar jika FILE ada dan merupakan file biasa
-d FILE         -||- Benar jika FILE ada dan meruapakan direktory
```

Operator logika

```
ekspr1 -o ekspr2        Benar jika jika salah satu ekspresi benar (or,||)
ekspr1 -a ekspr2        Benar jika ekspresi1 dan ekspresi2 benar (and,&&)
! ekspresi      Mengembalikan Benar jika ekspresi tidak benar (not!)
```

untuk informasi lebih lengkap man bash atau info bash di prompt shell anda.

Seleksi

if

Statement builtin if berfungsi untuk melakukan seleksi berdasarkan suatu kondisi tertentu

syntax:

```
if test-command1;
   then
      perintah1;
elif test-command2;
   then
      perintah2;
else
      alternatif_perintah;
fi
```

contoh script if1:

```
#!/bin/bash
#if1

clear;
if [ $# -lt 1 ];
   then
     echo "Usage : $0 [arg1 arg2 ...]"
     exit 1;
fi
```

```
echo "Nama script anda : $0";
echo "Banyak argumen   : $#";
echo "Argumennya adalah: $*";
```

Hasilnya:

```
[fajar@linux$]./if1

Usage : ./if1 [arg1 arg2 ...]
```

statement dalam blok if...fi akan dieksekusi apabila kondisi if terpenuhi, dalam hal ini jika script if1 dijalankan tanpa argumen. kita tinggal membaca apakah variabel $# lebih kecil (less than) dari 1, jika ya maka eksekusi perintah di dalam blok if ..fi tsb. perintah exit 1 akan mengakhiri jalannya script, angka 1 pada exit adalah status yang menandakan terdapat kesalahan, status 0 berarti sukses, anda dapat melihat isi variabel $? yang menyimpan nilai status exit, tetapi jika anda memasukkan satu atau lebih argumen maka blok if...fi tidak akan dieksekusi, statement diluar blok if..filah yang akan dieksekusi.

contoh script if2:

```
#!/bin/bash

kunci="bash";
read -s -p "Password anda : " pass
if [ $pass != $kunci ]
then echo "Maaf, anda gagal";
else echo "Sukses, anda layak dapat linux";
fi
```

Hasilnya

```
[fajar@linux$]./if2
Password anda : bash
Sukses, anda layak dapat linux
[fajar@linux$]./if2
Password anda : Bash
Maaf, anda gagal
```

klausa else akan dieksekusi jika if tidak terpenuhi, sebaliknya jika if terpenuhi maka else tidak akan dieksekusi

contoh script if3: penyeleksian dengan kondisi majemuk

```
#!/bin/bash

clear
echo "MENU HARI INI";
echo "--------------";
echo "1. Bakso     ";
echo "2. Gado-Gado ";
echo "3. Exit      ";
read -p "Pilihan anda [1-3] :" pil;

if [ $pil -eq 1 ];
```

```
then
   echo "Banyak mangkuk =";
   read jum
  let bayar=jum*1500;
elif [ $pil -eq 2 ];
then
   echo "Banyak porsi =";
   read jum
   let bayar=jum*2000;
elif [ $pil -eq 3 ];
then
   exit 0
else
   echo "Sorry, tidak tersedia"
   exit 1
fi

echo "Harga bayar = Rp. $bayar"
echo "THX"
```

Hasilnya:

```
[fajar@linux$]./if3
MENU HARI INI
-------------
1. Bakso
2. Gado-Gado
3. Exit
Pilihan anda :2

Banyak porsi = 2

Harga bayar = Rp. 4000
THX
```

statement builtin case

seperti halnya if statement case digunakan untuk menyeleksi kondisi majemuk, dibanding if, pemakaian case terasa lebih efisien

syntax:

```
case WORD in [ [(] PATTERN [| PATTERN]...) COMMAND-LIST ;;]...
esac
```

contoh script cs1

```
#!/bin/bash

clear
echo -n "Masukkan nama binatang :";
read binatang;

case $binatang in
    pinguin | ayam | burung ) echo "$binatang berkaki 2"
                  exit
                  ;;
    onta | kuda | anjing ) echo "$binatang berkaki 4"
                  exit
```

```
                ;;
    *) echo "$binatang blom didaftarkan"
                exit
                ;;
esac
```

Hasilnya:

```
[fajar@linux$]./cs1
Masukkan nama binatang : pinguin
pinguin berkaki 2
```

Perulangan

statement for

syntax:

```
for NAME [in WORDS ...]; do perintah; done
```

contoh script for1

```
#!/bin/bash

for angka in 1 2 3 4 5;
do
   echo "angka=$angka";
done
```

Hasilnya:

```
[fajar@linux$]./for1
angka=1
angka=2
angka=3
angka=4
angka=5
```

contoh script for2 berikut akan membaca argumen yang disertakan waktu script dijalankan

```
#!/bin/bash

for var
do
   echo $var
done
```

Hasilnya:

```
[fajar@linux$]./for2 satu 2 tiga
satu
```

```
2
tiga
```

atau variasi seperti berikut

```
#!/bin/bash

for var in `cat /etc/passwd`
do
   echo $var
done
```

Hasilnya: hasil dari perintah cat terhadap file /etc/passwd disimpan ke var dan ditampilkan menggunakan echo $var ke layar, mendingan gunakan cat /etc/passwd saja biar efisien. :-)

statement while

selama kondisi bernilai benar atau zero perintah dalam blok while akan diulang terus

syntax:

```
while KONDISI; do perintah; done;
```

contoh script wh1 mencetak bilangan ganjil antara 1-10

```
#!/bin/bash

i=1;
while [ $i -le 10 ];
do
  echo "$i,";
  let i=$i+2;
done
```

Hasilnya:

```
[fajar@linux$]./wh1
1,3,5,7,9,
```

kondisi tidak terpenuhi pada saat nilai i=11 (9+2), sehingga perintah dalam blokwhile tidak dieksekusi lagi contoh script wh2 akan menghitung banyak bilangan genap dan ganjil yang ada.

```
#!/bin/bash

i=0;
bil_genap=0;
bil_ganjil=0;

echo -n "Batas loop :";
read batas

if [ -z $batas ] || [ $batas -lt 0 ]; then
  echo "Ops, tidak boleh kosong atau Batas loop  harus >= 0";
```

```
  exit 0;
fi

while [ $i -le $batas ];
do
    echo -n "$i,";
    if [ `expr  $i % 2` -eq 0 ]; then
       let bil_genap=$bil_genap+1;
    else
       let bil_ganjil=$bil_ganjil+1;
    fi
    let i=$i+1;    #counter untuk mencapai batas
done

echo
echo "banyak bilangan genap  = $bil_genap";
echo "banyak bilangan ganjil = $bil_ganjil";
```

Hasilnya:

```
[fajar@linux$]./wh2
Batas loop : 10
0,1,2,3,4,5,6,7,8,9,10,
banyak bilangan genap  = 6
banyak bilangan ganjil = 5
```

untuk mengetahui apakah nilai i berupa bilangan genap kita cukup menggunakan operasi matematika % (mod), jika nilai i dibagi 2 menghasilkan sisa 0 berarti i adalah bilangan genap (semua bilangan genap yang dibagi dengan 2 mempunyai sisa 0) maka pencacah (bil_genap) dinaikkan 1, selain itu i bilangan ganjil yang dicatat oleh pencacah bil_ganjilproses ini dilakukan terus selama nilai i lebih kecil atau samadengan nilai batas yang dimasukkan. script juga akan memeriksa dahulu nilai batas yang dimasukkan apabila kosong atau lebih kecil dari 0 maka proses segera berakhir.tentunya dengan statement while kita sudah dapat membuat perulangan pada script kedai diatas agar dapat digunakan terus-menerus selama operator masih ingin melakukan proses perhitungan. lihat contoh berikut:

```
#!/bin/bash
#kedai

lagi='y'
while  [ $lagi == 'y' ] || [ $lagi == 'Y' ];
do
   clear
   echo "MENU HARI INI";
   echo "--------------";
   echo "1. Bakso      ";
   echo "2. Gado-Gado  ";
   echo "3. Exit       ";
   read -p "Pilihan anda [1-3] :" pil;

if [ $pil -eq 1 ];
then
   echo -n "Banyak mangkuk =";
   read jum
   let bayar=jum*1500;
elif [ $pil -eq 2 ];
then
   echo -n "Banyak porsi =";
   read jum
   let bayar=jum*2000;
elif [ $pil -eq 3 ];
then
```

```
   exit 0
else
   echo "Sorry, tidak tersedia"
   exit 1
fi

echo "Harga bayar = Rp. $bayar"
echo "THX"
echo
echo -n "Hitung lagi (y/t) :";
read lagi;

    #untuk validasi input
    while  [ $lagi != 'y' ] && [ $lagi != 'Y' ] && [ $lagi != 't' ] && [ $lagi != 'T' ];
    do
       echo "Ops, isi lagi dengan (y/Y/t/Y)";
       echo -n "Hitung lagi (y/t) :";
       read lagi;
    done

done
```

proses pemilihan menu dan perhitungan biaya akan diulang terus selama anda memasukkan y/Y dan t/T untuk berhenti. dalam script terdapat validasi input menggunakan while, sehingga hanya y/Y/t/T saja yang dapat diterima soalnya saya belum mendapatkan fungsi yang lebih efisien :-)

statement until

jika while akan mengulang selama kondisi benar, lain halnya dengan statement until yang akan mengulang selama kondisi salah. berikut contoh script ut menggunakan until

```
#!/bin/bash

i=1;
until [ $i -gt 10 ];
do
  echo $i;
  let i=$i+1
done
```

Hasilnya:

```
[fajar@linux$]./ut
1,2,3,4,5,6,7,8,9,10,
```

perhatikan kodisi until yang salah [$i -gt 10], dimana nilai awal i=1 dan akan berhenti apabila nilai i = 11 (bernilai benar) 11 -gt 10.

statement select

select berguna untuk pembuatan layout berbentuk menu pilihan, anda lihat contoh script pembuatan menu diatas kita hanya melakukannya dengan echo secara satu persatu, dengan select akan terlihat lebih efisien.

syntax:

```
select varname in (&ltitem list>); do perintah; done
```

sewaktu dijalankan bash akan menampilkan daftar menu yang diambil dari item list, serta akan menampilkan prompt yang menunggu masukan dari keyboard, masukan tersebut oleh bash disimpan di variabel builtin REPLY, apabila daftar item list tidak dituliskan maka bash akan mengambil item list dari parameter posisi sewaktu script dijalankan. lebih jelasnya lihat contoh berikut:

```
#!/bin/bash
#menu1

clear
select menu
do
  echo "Anda memilih $REPLY yaitu $menu"
done
```

Hasilnya:

```
layout:
[fajar@linux$]./menu1 Slackware Redhat Mandrake
1) Slackware
2) Redhat
3) Mandrake
 #? 1
Anda memilih 1 yaitu Slackware
```

karena item list tidak disertakan dalam script, maka sewaktu script dijalankan kita menyertakan item list sebagai parameter posisi, coba gunakan statement select pada program kedai diatas.

```
#!/bin/bash
#kedai

lagi='y'
while  [ $lagi == 'y' ] || [ $lagi == 'Y' ];
do
   clear
   select menu in "Bakso" "Gado-Gado" "Exit";
   case $REPLY in
        1) echo -n "Banyak mangkuk =";
           read jum
           let bayar=jum*1500;
           ;;
        2) echo -n "Banyak porsi =";
           read jum
           let bayar=jum*2000;
           ;;
        3) exit 0
           ;;
        *) echo "Sorry, tidak tersedia"
           ;;
  esac
do
```

```
echo "Harga bayar = Rp. $bayar"
echo "THX"
echo
echo -n "Hitung lagi (y/t) :";
read lagi;
```

```
    #untuk validasi input
    while  [ $lagi != 'y' ] && [ $lagi != 'Y' ] && [ $lagi != 't' ] && [ $lagi != 'T' ];
    do
       echo "Ops, isi lagi dengan (y/Y/t/Y)";
       echo -n "Hitung lagi (y/t) :";
       read lagi;
    done
done
```

Array

adalah kumpulan variabel dengan tipe sejenis, dimana array ini merupakan feature Bash yang cukup indah :-) dan salah satu hal yang cukup penting dalam bahasa pemrograman, anda bisa membayangkan array ini sebagai tumpukan buku - buku dimeja belajar. lebih jelasnya sebaiknya lihat dulu contoh script berikut:

```
#!/bin/bash
#array1

buah=(Melon,Apel,Durian);
echo ${buah[*]};
```

Hasilnya:

```
[fajar@linux$]./array1.
Melon,Apel,Durian
```

anda lihat bahwa membuat tipe array di Bash begitu mudah, secara otomatis array buah diciptakan dan string Melon menempati index pertama dari array buah, perlu diketahui bahwa array di Bash dimulai dari index 0, jadi array buah mempunyai struktur seperti berikut:

```
buah[0] berisi Melon
buah[1] berisi Apel
buah[2] berisi Durian
```

0,1,2 adalah index array, berarti ada 3 elemen pada array buah, untuk menampilkan isi semua elemen array gunakan perintah subtitusi seperti pada contoh diatas, dengan index berisi "*" atau "@". dengan adanya index array tentunya kita dapat mengisi array perindexnya dan menampilkan isi array sesuai dengan index yang diinginkan. anda lihat contoh berikut:

```
#!/bin/bash
#array2

bulan[0]=31
bulan[1]=28
bulan[2]=31
bulan[3]=30
bulan[4]=31
bulan[5]=30
bulan[6]=31
bulan[7]=31
bulan[8]=30
bulan[9]=31
bulan[10]=30
```

```
bulan[11]=31
echo "Banyak hari dalam bulan November adalah ${bulan[10]} hari"
```

Hasilnya:

```
[fajar@linux$]./array2
Banyak hari dalam bulan November adalah 30 hari
```

sebenarnya kita dapat mendeklarasikan array secara eksplisit menggunakan statement declare

contohnya:

```
declare –a myarray
```

mendeklarasikan variabel myarray sebagai array dengan opsi -a, kemudian anda sudah dapat memberinya nilai baik untuk semua elemen atau hanya elemen tertentu saja dengan perulangan yang telah kita pelajari pengisian elemen array dapat lebih dipermudah, lihat contoh :

```
#!/bin/bash
#array3

#deklarasikan variabel array
declare –a angka

#clear
i=0;
while [ $i –le 4 ];
do
let isi=$i*2;
angka[$i]=$isi;
let i=$i+1;
done

#tampilkan semua elemen array
#dengan indexnya berisi "*" atau "@"
echo ${angka[*]};

#destroy array angka (memory yang dipakai dibebaskan kembali)
unset angka
```

Hasilnya:

```
[fajar@linux$]./array3
0 2 4 6 8
```

Subrutin atau Fungsi

merupakan bagian script atau program yang berisi kumpulan beberapa statement yang melaksanakan tugas tertentu. dengan subrutin kode script kita tentunya lebih sederhana dan terstruktur, karena sekali fungsi telah dibuat dan berhasil maka akan dapat digunakan kapan saja kita inginkan. beberapa hal mengenai fungsi ini adalah:

- Memungkinkan kita menyusun kode script ke dalam bentuk modul-modul kecil yang lebih efisien dengan tugasnya masing-masing.
- Mencegah penulisan kode yang berulang - ulang.

untuk membuat subrutin shell telah menyediakan keyword function seperti pada bahasa C, akan tetapi ini bersifat optional (artinya boleh digunakan boleh tidak).

syntax:

```
function nama_fungsi() { perintah; }
```

nama_fungsi adalah pengenal (identifier) yang aturan penamaannya sama seperti pemberian nama variabel, setelah fungsi dideklarasikan atau dibuat anda dapat memaggilnya dengan menyebutkan nama fungsinya. lebih jelasnya lihat contoh script fungsi1 berikut:

```
#!/bin/bash

function say_hello() {
    echo "Hello, apa kbabar"
}
```

```
#panggil fungsi
say_hello;
```

```
#panggil sekali lagi
say_hello;
```

Hasilnya:

```
[fajar@linux$]./fungsi1
Hello, apa khabar

Hello, apa khabar
```

jika keyword function disertakan maka kita boleh tidak menggunakan tanda kurung (), tetapi jika keyword function tidak disertakan maka tanda kurung harus digunakan, lihat contoh berikut:

```
#!/bin/bash

function say_hello{
  echo "Hello,apa khabar"
}

balas(){
  echo "Baik-baik saja";
  echo "Bagaimana dengan anda ?";
}


#panggil fungsi say_hello
say_hello;
```

```
#panggil fungsi balas
balas;
```

Hasilnya:

```
[fajar@linux$]./fungsi2
Hello, apa khabar
Baik-baik saja
Bagaimana dengan anda ?
```

Mengirim argumen sebagai parameter ke fungsi

tentunya suatu fungsi lebih berdaya guna apabila dapat menerima argumen yang dikirim oleh pemanggilnya dan memproses argumen tsb didalam fungsinya, fungsi yang kita buat pada bash shell tentunya dapat melakukan hal tsb, apabila pada pemanggilan fungsi kita menyertakan argumen untuk diproses fungsi tsb, maka bash akan menyimpan argumen - argumen tsb pada parameter posisi 1,2,3,dst..., nah dengan memanfaatkan parameter posisi tsb tentunya kita dapat mengambil nilai yang dikirim. lebih jelasnya anda lihat contoh berikut:

```
#!/bin/bash

function hello{
   if [ -z $1 ]; then
      echo "Hello, apa khabar anda"
   else
      echo "Hello $1, apa khabar";
   fi
}

#masukkan nama anda disini
echo -n "Nama anda :";
read nama

#panggil fungsi dan kirim isi variabel nama ke fungsi untuk dicetak
hello $nama;
```

Hasilnya:

```
[fajar@linux$]./fungsi3
Nama anda : pinguin
Hello pinguin, apa khabar
```

lihat fungsi hello, sebelum mencetak pesan kita melakukan pemeriksaan dengan if terhadap parameter posisi $1 apabila kosong maka pesan "Hello, apa khabar anda" yang akan ditampilkan, tetapi jika ada string yang kita input maka string tersebut akan dicetak di dalam blok else pada fungsi. argumen pertama diteruskan ke variabel 1, argumen kedua pada variabel 2, dst.. jika argumen yang dikirim lebih dari satu.

Cakupan Variabel

secara default variabel - variabel yang digunakan dalam script adalah variabel bersifat global, maksud global adalah bahwa variabel tsb dikenal dan dapat diakses oleh semua fungsi dalam script, tetapi bash menyediakan keyword local yang berfungsi membatasi cakupan (scope) suatu variabel agar dikenal hanya

oleh fungsi yang mendeklarasikannya.coba lihat contoh berikut:

```
#!/bin/bash

proses(){
   echo "Isi variabel a=$a";
}

a=2;
proses();
proses $a
```

Hasilnya:

```
Isi variabel a=2
Isi variabel a=2
```

coba anda tambahkan local a pada fungsi proses menjadi

```
proses(){
   local a;
   echo –e "a didalam fungsi, a=$a";
}

a=10;
proses()

echo "a diluar fungsi, a=$a"
proses $a
```

Hasilnya:

```
a didalam fungsi, a=
a diluar fungsi,  a=10
a didalam fungsi  a=
```

nah jelas perbedaannya jika mendeklarasikan variabel memakai keyword local menyebabkan variabel tersebut hanya berlaku pada fungsi yang mendekalarasikannya. pada contoh dalam fungsi proses variabel a dideklarasikan sebagai variabel local dan tidak diberi nilai.

Diakhir dokumentasi ini saya menyertakan contoh script sederhana untuk melakukan entry data-data KPLI (Kelompok Pencinta Linux Indonesia) dan menyimpannya ke sebuah file. perintah-perintah shell dan beberapa utility yang digunakan adalah:

- apa yang telah anda pelajari diatas
- utility test, touch
- operator redirection ">>" untuk menambah data
- sleep, grep (global regular expression parser), cut, cat, | (pipa), sort dan more
- tput untuk menempatkan cursor pada koordinat tertentu (baris kolom)

sebagai latihan silahkan mengembangkan sendiri script dibawah ini:

```
#!/bin/bash
#-----------------------------------------------------------------------
#(C) Moh.fajar Makassar 2001, contoh script buat para linuxer
#file ini adalah public domain, silahkan mendistribusikan kembali
#atau mengubahnya asalkan anda mengikuti aturan - aturan dari GPL
#

menu(){
  clear
  tput cup 2 8;
  echo "SIMPLE DATABASE KPLI"
  tput cup 3 11;
  echo "1. Entry Data"
  tput cup 4 11;
  echo "2. Cari  Data"
  tput cup 5 11;
  echo "3. Cetak Data"
  tput cup 6 11;
  echo "4. Exit"
  tput cup 7 9;
  read -p "Pilihan anda [1-4] :" pil;
  while [ -z $pil ] || [ $pil -lt 1 ] || [ $pil -gt 4 ];
  do
     tput cup 7 9
     read -p "Pilihan anda [1-4] :" pil;
  done
}

entry()
{
  tput cup 9 27
  echo "Enrty data"
  tput cup 11 27
  echo -n "Nama KPLI :";
  read nama;

  while  [ -z $nama ] || grep  $nama $data -q -i;
  do
      tput cup 13 27
      echo "Ops Tidak boleh kosong atau $nama sudah ada";
      sleep 3
      clear
      tput cup 11 27
      echo -n "Nama KPLI :";
      read nama;
  done

  tput cup 12 27
  echo -n "Kota      :";
  read kota;
  tput cup 13 27
  echo -n "Alamat    :";
  read alamat;
  tput cup 14 27
  echo -n "Email     :";
  read email;
  tput cup 16 27
  echo "Rekam data ke file"
  if !(echo $nama:$kota:$alamat:$email>>$data); then
       echo "Ops, gagal merekam ke file"
       exit 1;
  fi
  sleep 3;
}
```

```
cari(){
  tput cup 9 27
  echo "Cari data per record"
```

```
  tput cup 11 27
  echo -n "Nama KPLI   :";
  read nama;
  while [ -z $nama ];
  do
    tput cup 13 27
    echo "Ops, nama tidak boleh kosong"
    sleep 3;
    tput cup 11 27
    echo -n "Nama KPLI   :";
    read nama;
  done

  if found=`grep $nama $data -n -i`; then
     tput cup 12 27
     echo -n "Kota        :";
         echo "$found" | cut -d: -f3
         tput cup 13 27
         echo -n "Alamat      :";
         echo "$found" | cut -d: -f4
     tput cup 14 27
         echo -n "Mail        :";
         echo "$found" | cut -d: -f5
         tput cup 16 27
         echo -n "Record ke- $found" | cut -d: -f1
      else
         tput cup 13 27
         echo "Ops, data tidak ditemukan";
      fi
}

cetak()
{
  tput cup 12 27
  echo "Tampilkan Data"
  tput cup 13 27
  echo -n "1->Ascendig, 2->Descending :"
  read mode
  clear;
  if [ -z $mode ] || [ $mode -eq 1 ]; then
     cat $data | sort  | more -d
  elif [ $mode -eq 2 ]; then
     cat $data | sort -r | more -d
  else
  cat $data | sort | more -d
  fi
}

#block utama

  data="mydata"

  if !(test -e $data); then
    if !(touch $data); then
       echo "gagal buat file database"
       exit 1
    fi
  fi

  lagi='y'
  while [ $lagi == 'y' ] || [ $lagi == 'Y' ]
  do
  menu;
    case $pil in
       1) entry
          ;;
       2) cari;
          ;;
       3) cetak
          ;;
```

```
      4) clear;
         exit 0;
         ;;
      *)
         echo "$pil, tidak ada dalam pilihan"
         ;;
   esac

 tput cup 18 27
 echo -n "Ke Menu (y/t): ";
 read lagi;
 done

 clear
```

tentunya kemampuan script ini dapat kita tambahkan dengan mudah sehingga mendekati program database sesungguhnya, utility seperti tr, paste, egrep, lpr, dll.. cukup baik dan membantu untuk digunakan.

Referensi

- http://pemula.linux.or.id/programming/bash-shell.html
- http://www.gnu.org/software/bash/manual/bashref.html
- http://www.tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html
- http://www.tldp.org/LDP/abs/html/

Pranala Menarik

- Shell
- Sekitar Shell