

Modul Praktikum Sistem Operasi



Jurusan Teknik Informatika

Rudhi Wahyudi Febrianto, S.Kom., M.Kom
Sekolah Tinggi Teknologi Bandung

Modul VI

Manajemen Proses

SASARAN

Setelah mempelajari materi dalam bab ini, mahasiswa diharapkan mampu:

1. Memahami konsep proses pada sistem operasi Linux.
2. Menampilkan beberapa cara menampilkan hubungan proses parent dan child.
3. Menampilkan status proses dengan beberapa format berbeda.
4. Melakukan pengontrolan proses pada shell.
5. Memahami penjadwalan prioritas.

PRAKTIKUM

1. Login sebagai user.
2. Download program C++ untuk menampilkan bilangan prima yang bernama `primes`.
3. Lakukan percobaan-percobaan di bawah ini kemudian analisa hasil percobaan.
4. Selesaikan soal-soal latihan.

Percobaan 1 : Status Proses

1. Pindah ke *command line terminal* (tty2) dengan menekan Ctrl+Alt+F2 dan login ke terminal sebagai user.
2. Instruksi `ps` (*process status*) digunakan untuk melihat kondisi proses yang ada. PID adalah Nomor Identitas Proses, TTY adalah nama terminal dimana proses tersebut aktif, STAT berisi S (*Sleeping*) dan R (*Running*), COMMAND merupakan instruksi yang digunakan.

```
$ ps
```

3. Untuk melihat faktor/elemen lainnya, gunakan option `-u` (user). %CPU adalah presentasi CPU time yang digunakan oleh proses tersebut, %MEM adalah presentasi system memori yang digunakan proses, SIZE adalah jumlah memori yang digunakan, RSS (*Real System Storage*) adalah jumlah memori yang digunakan, START adalah kapan proses tersebut diaktifkan

```
$ ps -u
```

4. Mencari proses yang spesifik pemakai. Proses diatas hanya terbatas pada proses milik pemakai, dimana pemakai teresbut melakukan login

```
$ ps -u <user>
```

5. Mencari proses lainnya gunakan opsi *a* (*all*) dan *au* (*all user*)

```
$ ps -a
```

```
$ ps -au
```

6. Logout dan tekan Alt+F7 untuk kembali ke mode grafis

Percobaan 2 : Menampilkan Hubungan Proses Parent dan Child

1. Pindah ke *command line terminal* (tty2) dengan menekan **Ctrl+Alt+F2** dan login ke terminal sebagai user.

2. Ketik **ps -eH** dan tekan **Enter**. Opsi **e** memilih semua proses dan opsi **H** menghasilkan tampilan proses secara hierarki. Proses child muncul dibawah proses parent. Proses child ditandai dengan awalan beberapa spasi.

```
$ ps -eH
```

3. Ketik **ps -e f** dan tekan **Enter**. Tampilan serupa dengan langkah 2. Opsi **-f** akan menampilkan status proses dengan karakter grafis (**** dan **_**)

```
$ ps -e f
```

4. Ketik **pstree** dan tekan **Enter**. Akan ditampilkan semua proses pada sistem dalam bentuk hirarki parent/child. Proses parent di sebelah kiri proses child. Sebagai contoh proses `init` sebagai parent (*ancestor*) dari semua proses pada sistem. Beberapa child dari `init` mempunyai child. Proses `login` mempunyai i proses `bash` sebagai child. Proses `bash` mempunyai proses child `startx`. Proses `startx` mempunyai child `xinit` dan seterusnya.

```
$ pstree
```

5. Ketik **pstree | grep mingetty** dan tekan **Enter**. Akan menampilkan semua proses `mingetty` yang berjalan pada system yang berupa *console virtual*. Selain menampilkan semua proses, proses dikelompokkan dalam satu baris dengan suatu angka sebagai jumlah proses yang berjalan.

```
$ pstree | grep mingetty
```

6. Untuk melihat semua PID untuk proses gunakan opsi **-p**.
-

```
$ pstree -p
```

7. Untuk menampilkan proses dan ancestor yang tercetak tebal gunakan opsi **-h**.

```
$ pstree -h
```

Percobaan 3 : Menampilkan Status Proses dengan Berbagai Format

1. Pindah ke *command line terminal* (tty2) dengan menekan **Ctrl+Alt+F2** dan login ke terminal sebagai user.

2. Ketik **ps -e | more** dan tekan **Enter**. Opsi **-e** menampilkan semua proses dalam bentuk 4 kolom : PID, TTY, TIME dan CMD.

```
$ ps -e | more
```

Jika halaman penuh terlihat prompt `--More--` di bagian bawah screen, tekan **q** untuk kembali ke prompt perintah.

3. Ketik **ps ax | more** dan tekan **Enter**. Opsi **a** akan menampilkan semua proses yang dihasilkan terminal (TTY). Opsi **x** menampilkan semua proses yang tidak dihasilkan terminal. Secara logika opsi ini sama dengan opsi **-e**. Terdapat 5 kolom : PID, TTY, STAT, TIME dan COMMAND.

```
$ ps ax | more
```

Jika halaman penuh terlihat prompt `--More--` di bagian bawah screen, tekan **q** untuk kembali ke prompt perintah.

4. Ketik **ps -e f | more** dan tekan **Enter**. Opsi **-e f** akan menampilkan semua proses dalam format daftar penuh.

```
$ ps ef | more
```

Jika halaman penuh terlihat prompt `--More--` di bagian bawah screen, tekan **q** untuk kembali ke prompt perintah.

5. Ketik **ps -eo pid, cmd | more** dan tekan **Enter**. Opsi **-eo** akan menampilkan semua proses dalam format sesuai definisi user yaitu terdiri dari kolom PID dan CMD.

```
$ ps -eo pid,cmd | more
```

Jika halaman penuh terlihat prompt `--More--` di bagian bawah screen, tekan **q** untuk kembali ke prompt perintah.

6. Ketik **ps -eo pid,ppid,%mem,cmd | more** dan tekan **Enter**. Akan menampilkan kolom PID, PPID dan %MEM. PPID adalah proses ID dari proses parent. %MEM menampilkan persentasi memory system yang digunakan proses. Jika proses hanya menggunakan sedikit memory system akan ditampilkan 0.

```
$ ps -eo pid,ppid,%mem,cmd | more
```

7. **Logout** dan tekan **Alt+F7** untuk kembali ke mode grafis

Percobaan 4 : Mengontrol Proses Pada Shell

1. Pindah ke *command line terminal* (tty2) dengan menekan **Ctrl+Alt+F2** dan login ke terminal sebagai user.

2. Gunakan perintah `yes` yang mengirim output `y` yang tidak pernah berhenti

```
$ yes
```

Untuk menghentikannya gunakan **Ctrl-C**.

3. Belokkan standart output ke `/dev/null`

```
$ yes > /dev/null
```

Untuk menghentikannya gunakan **Ctrl-C**.

4. Salah satu cara agar perintah `yes` tetap dijalankan tetapi shell tetap digunakan untuk hal yang lain dengan meletakkan proses pada *background* dengan menambahkan karakter `&` pada akhir perintah.

```
$ yes > /dev/null &
```

Angka dalam `"* +"` merupakan **job number** diikuti PID.

5. Untuk melihat status proses gunakan perintah `jobs`.

```
$ jobs
```

6. Untuk menghentikan job, gunakan perintah `kill` diikuti *job number* atau PID proses. Untuk identifikasi job number, diikuti prefix dengan karakter `"%"`.

```
$ kill %<nomor job> contoh: kill %1
```

7. Lihat status job setelah diterminasi

```
$ jobs
```

Percobaan 5 : Menghentikan dan Memulai Kembali Job

1. Cara lain meletakkan job pada *background* dengan memulai job secara normal (pada *foreground*), **stop** job dan memulai lagi pada *background*

```
$ yes > /dev/null
```

Hentikan sementara job (*suspend*), bukan menghentikannya (*terminate*), tetapi menghentikan sementara job sampai di restart. Untuk menghentikan sementara job gunakan **Ctrl-Z**.

2. Untuk restart job pada *foreground*, gunakan perintah *fg*.

```
$ fg
```

3. Shell akan menampilkan nama perintah yang diletakkan di *foreground*. Stop job lagi dengan **Ctrl-Z**. Kemudian gunakan perintah *bg* untuk meletakkan job pada *background*.

```
$ bg
```

Job tidak bisa dihentikan dengan **Ctrl-Z** karena job berada pada *background*. Untuk menghentikannya, letakkan job pada *foreground* dengan *fg* dan kemudian hentikan sementara dengan **Ctrl-Z**.

```
$ fg
```

4. Job pada *background* dapat digunakan untuk menampilkan teks pada terminal, dimana dapat diabaikan jika mencoba mengerjakan job lain.

```
$ yes &
```

Untuk menghentikannya tidak dapat menggunakan **Ctrl-C**. Job harus dipindah ke *foreground*, baru dihentikan dengan cara tekan **fg** dan tekan **Enter**, kemudian dilanjutkan dengan **Ctrl-Z** untuk menghentikan sementara.

5. Apabila ingin menjalankan banyak job dalam satu waktu, letakkan job pada *foreground* atau *background* dengan memberikan job ID

```
$ fg %2 atau $ %2
```

```
$ bg %2
```

6. tekan **fg** dan tekan **Enter**, kemudian dilanjutkan dengan **Ctrl-Z** untuk menghentikan sementara.
7. Lihat job dengan perintah **ps -fae** dan tekan **Enter**. Kemudian hentikan proses dengan perintah **kill**.

```
$ ps -fae
```

```
$ kill -9 <NomorPID>
```

8. **Logout** dan tekan **Alt+F7** untuk kembali ke mode grafis

Percobaan 6 : Percobaan dengan Penjadwalan Prioritas

1. Login sebagai root.
 2. Buka 3 terminal, tampilkan pada screen yang sama.
 3. Pada setiap terminal, ketik **PS1 = "\w:"** diikuti **Enter**. **\w** menampilkan path pada direktori home.
 4. Karena login sebagai root, maka akan ditampilkan **~:** pada setiap terminal. Untuk setiap terminal ketik **pwd** dan tekan **Enter** untuk melihat bahwa Anda sedang berada pada direktori **/root**.
 5. Buka terminal lagi (keempat), atur posisi sehingga keempat terminal terlihat pada screen.
 6. Pada terminal keempat, ketik **top** dan tekan **Enter**. Maka program **top** akan muncul. Ketik **i**. **Top** akan menampilkan proses yang aktif. Ketik **lmt**. **Top** tidak lagi menampilkan informasi pada bagian atas dari screen. Pada percobaan ini, terminal ke empat sebagai jendela **Top**.
 7. Pada terminal 1, bukalah program executable C++ dengan mengetik program **yes** dan tekan **Enter**.
 8. Ulangi langkah 7 untuk terminal 2.
 9. Jendela **Top** akan menampilkan dua program **yes** sebagai proses yang berjalan. Nilai %CPU sama pada keduanya. Hal ini berarti kedua proses mengkonsumsi waktu proses yang sama dan berjalan sama cepat. PID dari kedua proses akan berbeda, misalnya 3148 dan 3149. Kemudian gunakan terminal 3 (yang tidak menjalankan **primes** maupun Jendela **Top**) dan ketik **renice 19 <PID terminal 1>** (contoh : **renice 19 3148**) dan diikuti **Enter**. Hal ini berarti mengganti penjadwalan prioritas dari proses ke 19.
 10. Tunggu beberapa saat sampai program **top** berubah dan terlihat pada jendela **Top**. Pada kolom **STAT** memperlihatkan **N** untuk proses 3148. Hal ini berarti bahwa penjadwalan prioritas untuk proses 3148 lebih besar (lebih lambat) dari 0. Proses 3149 berjalan lebih cepat.
 11. Program **top** juga mempunyai fungsi yang sama dengan program **renice**. Pilih Jendela **Top** dan tekan **r**. Program **top** terdapat prompt **PID to renice:** tekan **3148** (ingat bahwa Anda harus mengganti 3148 dengan PID Anda sendiri) dan tekan **Enter**. Program **top** memberikan prompt **Renice PID 3148 to value:** tekan **-19** dan tekan **Enter**.
-

12. Tunggu beberapa saat sampai `top` berubah dan lihat nilai %CPU pada kedua proses. Sekarang proses 3148 lebih cepat dari proses 3149. Kolom status menunjukkan `<` pada proses 3148 yang menunjukkan penjadwalan prioritas lebih rendah (lebih cepat) dari nilai 0.
13. Pilih terminal 3 (yang sedang tidak menjalankan `yes` atau program `top`) dan ketik **`nice -n -10 yes`** dan tekan **Enter**. Tunggu beberapa saat agar program `top` berubah dan akan terlihat proses `primes` ketiga. Misalnya PID nya 4107. Opsi -10 berada pada kolom NI (penjadwalan prioritas).
14. Jangan menggunakan mouse dan keyboard selama 10 detik. Program `top` menampilkan proses yang aktif selain program `yes`. Maka akan terlihat proses `top` terdaftar tetapi %CPU kecil (dibawah 1.0) dan konsisten. Juga terlihat proses berhubungan dengan dekstop grafis seperti X, panel dll.
15. Pindahkan mouse sehingga kursor berubah pada screen dan lihat apa yang terjadi dengan tampilan `top`. Proses tambahan akan muncul dan nilai %CPU berubah sebagai bagian grafis yang bekerja. Satu alasan adalah bahwa proses 4107 berjalan pada penjadwalan prioritas tinggi. Pilih jendela **Top**, ketik **`r. PID to renice`** : muncul prompt. Ketik **`4107`** (ubahlah 4107 dengan PID Anda) dan tekan **Enter**. **`Renice PID 4107 to value:`** muncul prompt. Ketik **`0`** dan tekan **Enter**. Sekarang pindahkan mouse ke sekeliling screen. Lihat perubahannya.
16. Tutup semua terminal window.
17. Logout dan login kembali sebagai user.

LATIHAN

1. Masuk ke tty2 dengan **`Ctrl+Alt+F2`**. Ketik **`ps -au`** dan tekan **Enter**. Kemudian perhatikan keluaran sebagai berikut :
 - a. Sebutkan nama-nama proses yang bukan root
 - b. Tulis PID dan COMMAND dari proses yang paling banyak menggunakan CPU time
 - c. Sebutkan buyut proses dan PID dari proses tersebut
 - d. Sebutkan beberapa proses daemon
 - e. Pada prompt login lakukan hal- hal sebagai berikut :

```
$ csh
```

```
$ who
$ bash
$ ls
$ sh
$ ps
```

- f. Sebutkan PID yang paling besar dan kemudian buat urutan proses sampai ke PPID = 1.
2. Cobalah format tampilan ps dengan opsi berikut dan perhatikan hasil tampilannya :
- **-f** daftar penuh
 - **-j** format job
 - **j** format job control
 - **l** daftar memanjang
 - **s** format sinyal
 - **v** format virtual memory
 - **X** format register i386
3. Lakukan urutan pekerjaan berikut :
- a. Gunakan perintah `find` ke seluruh direktori pada sistem, blokirkan output sehingga daftar direktori dialihkan ke file `directories.txt` dan daftar pesan error dialihkan ke file `errors.txt`
 - b. Gunakan perintah `sleep 5`. Apa yang terjadi dengan perintah ini ?
 - c. Jalankan perintah pada *background* menggunakan `&`
 - d. Jalankan `sleep 15` pada *foreground* , hentikan sementara dengan Ctrl- Z dan kemudian letakkan pada *background* dengan `bg`. Ketikkan `jobs`. Ketikkan `ps`. Kembalikan job ke *foreground* dengan perintah `fg`.
 - e. Jalankan `sleep 15` pada *background* menggunakan `&` dan kemudian gunakan perintah `kill` untuk menghentikan proses diikuti *job number*.
 - f. Jalankan `sleep 15` pada *background* menggunakan `&` dan kemudian gunakan `kill` untuk menghentikan sementara proses. Gunakan `bg` untuk melanjutkan menjalankan proses.
-

- g. Jalankan `sleep 60` pada *background* 5 kali dan terminasi semua pada dengan menggunakan perintah `killall`.
- h. Gunakan perintah `ps`, `w` dan `top` untuk menunjukkan semua proses yang sedang dieksekusi.
- i. Gunakan perintah `ps -aeH` untuk menampilkan hierarki proses. Carilah init proses. Apakah Anda bisa identifikasi sistem daemon yang penting ?
Dapatkan Anda identifikasi shell dan subproses ?
- j. Kombinasikan `ps -fae` dan `grep`, apa yang Anda lihat ?
- k. Jalankan proses `sleep 300` pada *background*. Log off komputer dan log in kembali. Lihat daftar semua proses yang berjalan. Apa yang terjadi pada proses `sleep` ?

LAPORAN RESMI

1. Analisa hasil percobaan yang Anda lakukan.
 2. Kerjakan latihan diatas dan analisa hasil tampilannya.
 3. Berikan kesimpulan dari praktikum ini.
-

Modul VII

Pemrograman Shell 1

SASARAN

Setelah mempelajari materi dalam bab ini, mahasiswa diharapkan mampu:

1. Memahami shell pada sistem operasi Linux.
2. Menggunakan feature history pada Bash Shell.
3. Mengubah feature history pada Bash Shell.
4. Mengubah prompt shell.
5. Melakukan konfigurasi Bash Shell untuk menjalankan skrip secara otomatis.
6. Membuat dan mengeksekusi shell script sederhana melalui editor vi.
7. Memahami job control.
8. Memahami stack.
9. Menggunakan alias.

PRAKTIKUM

1. Login sebagai user.
2. Bukalah Console Terminal dan lakukan percobaan-percobaan di bawah ini kemudian analisa hasil percobaan.
3. Selesaikan soal-soal latihan.

Percobaan 1 : *Profile*

1. File `.bash_profile` dijalankan pada home direktori pemakai yang login. File `.bash_profile` adalah *hidden file*, sehingga untuk melihatnya gunakan opsi `a` pada instruksi `ls`.

```
$ ls -a
```

```
$ more .bash_profile
```

2. File `.bash_logout` akan dieksekusi sesaat sebelum logout, berfungsi sebagai *house clearing jobs*, artinya membersihkan semuanya, misalnya menghapus temporary file atau job lainnya. Melihat file `.bash_logout` dengan instruksi

```
$ cat .bash_logout
```

Percobaan 2 : Menggunakan Feature History Bash

1. Bash shell menyimpan "history" perintah yang digunakan sebelumnya. Anda dapat mengakses history dalam beberapa cara. Cara paling mudah adalah menggunakan **Panah Atas**. Maka perintah sebelumnya akan ditampilkan.
2. Berikutnya, berikan Bash shell beberapa perintah untuk diingat. Masukkan perintah berikut dan tekan **Enter** pada setiap baris.

```
$ cd
```

```
$ ls -l /etc
```

```
$ ls -l
```

```
$ whoami
```

```
$ who
```

3. Untuk memeriksa apakah perintah ini ditambahkan pada history, dapat menggunakan perintah `history` untuk melihat semua perintah yang pernah dimasukkan.

```
$ history
```

4. Anda dapat memilih perintah sebelumnya dengan menggunakan **Panah Atas**, tetapi hal ini tidak efisien untuk perintah yang semakin bertambah banyak. Cara yang mudah menggunakan nomor pada perintah history atau mencarinya. Untuk memilih dan mengeksekusi perintah dengan nomor, masukkan kunci ! diikuti nomor perintah.

```
$ !<Nomor Perintah> Contoh : !780
```

5. Anda dapat mencari perintah dengan menyertakan perintah yang diinginkan. Misalnya **!?etc?!** akan menjalankan perintah `ls -l /etc` yang sebelumnya digunakan.

```
$ !?etc?
```

6. Kemudian gunakan perintah history, maka akan terlihat perintah `ls -l /etc` yang kedua dan bukan **!?etc?**

```
$ history
```

7. Apabila string tidak ditemukan pada perintah history maka akan terdapat pesan error.

```
$ !?wombat99?
```

8. Jika diketikkan **!who** maka yang dijalankan adalah perintah `who`. Tetapi bila Anda ketikkan **!whoa** maka yang dijalankan adalah perintah `whoami`.

```
$ !who
```

```
$ !whoa
```

9. Anda bisa menggantikant string pada perintah `history`, terutama pada perintah yang panjang. Misalnya ketik **cat /bin/bash | strings | grep shell | less** dan tekan **Enter**. Maka akan menampilkan semua string pada file `/bin/bash` yang berisi kata "shell". Untuk keluar tekan q. Jika ingin menampilkan kata "alias", maka Anda tidak perlu mengetik perintah yang panjang lagi, tetapi cukup ketik **^shell^alias^** dan tekan **Enter** maka akan menggantikan kata "shell" dengan "alias".

```
$ cat /bin/bash | strings | grep shell | less
```

```
$ ^shell^alias^
```

Percobaan 3 : Mengubah Feature History Bash

1. Bash shell akan menyimpan perintah `history` meskipun telah log out dan log in kembali. File `.bash_history` menyimpan file history yang terdapat pada home directory.

```
$ cd
```

2. Lihat beberapa baris pada file `.bash_history` dengan ketik **tail .bash_history** dan tekan **Enter**. File ini bukan file yang up to date.

```
$ tail .bash_history
```

3. Ketik **history** dan tekan **Enter**. Maka akan terlihat baris terakhir adalah perintah `history` dan baris sebelumnya adalah `tail .bash_history`. Perintah `history` bersifat up to date, karena disimpan pada memory sistem.

```
$ history
```

4. Ketik perintah berikut

```
$ echo „Ini perintah saya“
```

5. Log out dan log in kembali sebagai user yang sama. Ketik **history** dan tekan **Enter**. Maka perintah `echo "Ini perintah saya"` akan berada pada baris terakhir. Lihat file `.bash_history`, maka perintah tsb akan terdapat pada file `.bash_history`.

```
$ history
```

```
$ tail .bash_history
```

6. Ketik **history|less** untuk melihat perintah history terakhir pada screen. Tekan spacebar untuk melihat file lebih banyak. Untuk keluar tekan q

```
$ history|less
```

7. Untuk melihat berapa banyak perintah history yang ada pada file ketik berikut dan output yang keluar serupa di bawah ini

```
$ wc -l .bash_history
```

```
1000 .bash_history
```

8. Output menunjukkan bahwa 1000 perintah history disimpan pada file history. Untuk melihat jangkauan (limit) perintah history digunakan variable **HISTSIZE**. Untuk melihat jangkauan history ketik sebagai berikut

```
$ set|grep HISTSIZE
```

9. Bila ingin memperbesar jangkauan file history, maka ubahlah variable **HISTSIZE** pada skrip startup yang disebut **.bashrc** pada home directory.

```
$ echo „HISTSIZE=5000“ >> .bashrc
```

10. Log out dan log in kembali sebagai user yang sama. Lihat perubahan variabel **HISTSIZE**.

```
$ set|grep HISTSIZE
```

11. Ketikkan perintah history beberapa kali, maka perintah ini akan disimpan pada BASH history meskipun yang diketikkan perintahnya sama.

12. Anda dapat melakukan konfigurasi BASH agar tidak menambah perintah ke history jika perintah yang diketikkan sama dengan sebelumnya. Hal ini dilakukan dengan menambahkan variabel **HISTCONTROL** dan diberikan nilai **ignoredups** pada file **.bashrc**

```
$ echo „HISTCONTROL=ignoredups“ >> .bashrc
```

13. Log out dan log in kembali sebagai user yang sama. Ketikkan history beberapa kali dan perhatikan berapa kali history muncul.

Percobaan 4 : Mengubah Prompt Shell

1. Prompt Bash shell dikonfigurasi dengan men-setting nilai variabel PS1. Selain menampilkan string statik sebagai prompt, Anda dapat menampilkan menjadi dinamis. Contohnya, apabila ingin

menunjukkan *current directory* atau *current time*. Ketik **PS1='\t:'** dan tekan **Enter** untuk menampilkan waktu sistem dalam format 24 jam sebagai prompt Bash. Format dalam HH:MM:SS

```
$ PS1="\t: "
```

2. Untuk menampilkan format 12 jam dengan indikator am dan pm ketik sebagai berikut

```
$ PS1="\t: "
```

3. Kebanyakan orang menginginkan prompt Bash menampilkan *current working directory*. Direktory dapat ditampilkan dalam bentuk keseluruhan path atau hanya nama direktory. Karakter \w menampilkan hanya nama direktory. Jika *current directory* adalah home directory, maka tampil prompt

```
~:
```

```
$ PS1="\w: "
```

4. Ketik **cd /usr/sbin** untuk melihat prompt /usr/sbin:

```
$ cd /usr/sbin
```

5. Ketik **PS1='\W:'** untuk melihat prompt sbin:

```
$ PS1="\W: "
```

6. Ada beberapa prompt BASH lain yang dapat diubah, yaitu PS2, PS3 dan PS4. Prompt PS2 digunakan sebagai prompt sekunder. Untuk melihat bagaimana penggunaannya, ketik **echo 'Hello** (tanpa diakhiri penutup quote) dan tekan **Enter**. Simbol lebih besar dari (>) akan muncul. Hal ini memberitahukan bahwa BASH menunggu Anda menyelesaikan perintah. Ketik penutup quote (') dan tekan **Enter**. Perintah ini akan menyelesaikan prompt PS2, kata **"Hello,"** muncul diikuti dengan prompt PS1 pada baris baru.

```
$ echo "Hello
```

```
>"
```

7. Anda dapat mengubah prompt PS2 seperti mengubah prompt PS1. Ketik perintah berikut :

```
$ PS2="Selesai memasukkan perintah Anda: "
```

8. Kemudian ketik echo 'Hello (tanpa diakhiri penutup quote) dan tekan Enter. Pada baris berikutnya akan muncul **Selesai memasukkan perintah Anda:**. Kemudian ketikkan penutup quote (') dan tekan **Enter**. Jika perintah selesai, maka kata **Hello** akan muncul diikuti prompt PS1 pada baris baru.

```
$ echo "Hello
```

Selesai memasukkan perintah Anda:"

9. Prompt BASH dapat ditampilkan berwarna dengan melakukan setting *colorsetting string* . Sebagai contoh, prompt BASH di-set dengan `\w\$,` akan menampilkan *current working directory* yang diikuti `$` (atau `#` jika anda login sebagai root). Untuk setting warna menjadi biru ketikkan berikut :

```
$ PS1="\033[0;34m\w\$ \033[0;37m"
```

10. Untuk mendapatkan prompt warna merah ketikkan berikut :

```
$ PS1="\033[0;31m\w\$ \033[0;37m"
```

30=hitam, 31=merah, 32=hijau, 34=biru, 35=ungu, 36=cyan, 37=putih.

11. Bila menginginkan beberapa warna, ketikkan perintah berikut :

```
$ PS1="\033[0;31m\w\033[0;32m\$ \033[0;37m"
```

12. Anda bisa menampilkan atribut visual seperti lebih terang, berkedip dan warna kebalikannya. Untuk menampilkan prompt yang lebih terang, atribut control diganti 1, seperti perintah berikut :

```
$ PS1="\033[1;34m\w\033[1;32m\$ \033[0;37m"
```

13. Untuk menampilkan prompt dengan warna berkebalikan, atribut control diganti 7, seperti perintah berikut :

```
$ PS1="\033[7;34m\w\033[7;32m\$ \033[0;37m"
```

14. Untuk menampilkan prompt berkedip, atribut control diganti 5, seperti perintah berikut :

```
$ PS1="\033[5;34m\w\033[5;32m\$ \033[0;37m"
```

Percobaan 5 : Menambahkan Otomatisasi ke Prompt Shell

1. Pastikan Anda berada di home directory

```
$ cd ~
```

2. Buatlah skrip sederhana untuk mengurut daftar file. Anda dapat menggunakan teks editor, tetapi karena hanya satu baris, gunakan perintah `echo` untuk membuat file.

```
$ echo "sort ~/list > ~/r13; mv ~/r13 ~/list" > ~/sorter
```

3. Buatlah file skrip diatas menjadi file executable

```
$ chmod +x sorter
```

4. Jalankan program sorter diatas setiap shell Bash menampilkan prompt PS1. Untuk melakukannya, buatlah variable **PROMPT_COMMAND** dimana nilainya adalah nama dari program `sorter`.

```
$ PROMPT_COMMAND=~/sorter
```

5. Ketikkan **echo 'John Smith:13001'>>list** dan tekan **Enter**. Jika file `list` tidak ada, akan dibuat secara otomatis, tetapi jika sudah ada, string 'John Smith:13001' akan ditambahkan.

```
$ echo "John Smith:13001">>list
```

6. Ketik **cat list** dan tekan **Enter**. Maka Anda akan melihat isi file `list`. Pada saat ini, file mungkin mempunyai hanya satu baris sehingga tidak dapat dilihat apakah file sudah terurut.

```
$ cat list
```

7. Masukkan beberapa perintah serupa dengan point 5 tetapi dengan nama dan nomor yang berbeda. Kemudian ketik **cat list** dan tekan **Enter**.

```
$ echo "Anita:13002">>list
```

```
$ echo "Samantha:13003">>list
```

```
$ echo "Patrik:13004">>list
```

```
$ echo "Sponse Bob:13005">>list
```

```
$ echo "Lisa:13006">>list
```

```
$ echo "Squid:13007">>list
```

8. Apabila Anda tidak menginginkan Shell Bash menampilkan file terurut sepanjang waktu, Anda tidak perlu menambahkan variable `PROMPT_COMMAND=~/.sorter` pada file konfigurasi seperti `.bashrc`. Bila Anda ingin BASH berhenti menjalankan program `sorter`, maka ketikkan variable **PROMPT_COMMAND=** dan tekan **Enter** atau log out dan login kembali.

```
$ PROMPT_COMMAND=
```

Percobaan 6 : Membuat Bash-script dan menjalankannya

1. Membuat file `p1.sh`

```
$ vi p1.sh
```

```
echo "Program bash Script"
```

2. Mengubah program menjadi executable

```
$ ls -l p1.sh
```

```
$ chmod +x p1.sh
```

```
$ ls -l p1.sh
```

3. Menjalankan script

```
$ bash p1.sh
```

```
$ sh p1.sh
```

```
$ . p1.sh
```

```
$ ./p1.sh
```

4. Konvensi dalam pembuatan script shell dinyatakan sebagai `#!/bin/bash`. Tambahkan pada file `p1.sh` konvensi tersebut

```
$ vi p1.sh
```

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

```
echo "Program bash script"
```

5. Buatlah file `p2.sh`

```
$ vi p2.sh
```

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

```
echo "Program 2 bash script"
```

6. Menjalankan beberapa program shell dalam satu baris instruksi yang dipisahkan dengan tanda ;

```
$ cat p1.sh ; cat p2.sh
```

```
$ ./p1.sh ; ./p2.sh
```

Percobaan 7 : Job Control

1. Proses foreground

```
$ ps x
```

2. Proses background

```
$ ps x > hasil &
```

3. Setiap job mempunyai PID yang tunggal (unique). Untuk melihat jobs yang aktif

```
$ jobs
```

4. Buatlah file `ploop.sh`. File ini tidak akan pernah berhenti kecuali ditekan Ctrl-C

```
$ vi ploop.sh
```

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

```
while [ true ]
```

```
do
```

```
    sleep 10
```

```
    echo "Hallo"
```

done

5. Buatlah file ploop.sh menjadi executable. Jalankan program, akan ditampilkan kata Hallo setiap 10 detik. Untuk keluar program, tekan Ctrl-C (^C)

```
$ chmod +x ploop.sh
```

```
$ ./ploop.sh
```

Percobaan 8 : Manipulasi Stack untuk Direktori

1. Instruksi `dirs` digunakan untuk melihat stack direktori, pada output hanya ditampilkan direktori home ~

```
$ dirs
```

2. Membuat 3 buah direktori

```
$ mkdir marketing sales support
```

3. Instruksi `dirs` digunakan untuk melihat stack direktori, pada output hanya ditampilkan direktori home ~

```
$ dirs
```

Percobaan 9 : Alias

1. Alias adalah mekanisme untuk memberi nama alias pada satu atau sekelompok instruksi. Untuk melihat alias yang sudah terdaftar pada system :

```
$ alias
```

2. Membuat beberapa alias

```
$ alias del="rm -i"
```

```
$ alias h="history"
```

3. Gunakan instruksi hasil alias

```
$ ls
```

```
$ del hasil
```

```
$ h | more
```

4. Untuk menghapus alias gunakan instruksi `unalias`

```
$ unalias del
```

```
$ del files (Terdapat Pesan Kesalahan, mengapa ?)
```

LATIHAN

1. Eksekusi seluruh profile yang ada :

- a. Edit file profile `/etc/profile` dan tampilkan pesan sebagai berikut :

```
echo "Profile dari /etc/profile"
```

- b. Asumsi nama anda `student`, maka edit semua profile yang ada yaitu :

```
/home/student/.bash_profile
```

```
/home/. student/.bash_login
```

```
/home/student/.profile
```

```
/home/student/.bashrc
```

- c. Ganti nama `/home/student` dengan nama anda sendiri. Pada setiap file tersebut, cantumkan instruksi `echo`, misalnya pada `/home/`

```
student/.bash_profile:
```

```
echo "Profile dari .bash_profile"
```

- d. Lakukan hal yang sama untuk file lainnya, sesuaikan tampilan dengan nama file yang bersangkutan.

2. Jalankan instruksi substitute user, kemudian keluar dengan perintah `exit` sebagai berikut :

```
$ su student
```

```
$ exit
```

kemudian gunakan opsi – sebagai berikut :

```
$ su - student
```

```
$ exit
```

Jelaskan perbedaan kedua utilitas tersebut.

3. Logout

- a. Edit file `.bash_logout`, tampilkan pesan dan tahan selama 5 detik, sebelum eksekusi `logout`

```
Echo "Terima kasih atas sesi yang diberikan"
```

```
Sleep 5
```

```
Clear
```

- b. Edit file `.bash_logout`, tampilkan pesan dan tahan selama 4 detik, sebelum eksekusi `logout`

4. History

- a. Ganti nilai `HISTSIZE` dari 1000 menjadi 20

```
$ HISTSIZE=20
```

```
$ h
```

- b. Gunakan fasilitas history dengan mengedit instruksi baris ke 5 dari instruksi yang terakhir dilakukan.

```
$ !-5
```

- c. Ulangi instruksi yang terakhir. Gunakan juga `^P` dan `^N` untuk bernavigasi pada history buffer

```
$ !!
```

- d. Ulangi instruksi pada history buffer nomor tertentu, misalnya nomor 150

```
$ !150
```

- e. Ulangi instruksi dengan prefix `ls`

```
$ !ls
```

```
$ !?ls?
```

Jelaskan perbedaan instruksi diatas

5. Prompt String (PS)

- a. Edit file `.bash_profile`, ganti prompt `PS1` dengan `'>'`. Instruksi `export` diperlukan dengan parameter nama variabel tersebut, agar perubahan variabel `PS1` dikenal oleh semua shell

```
PS1="> "
```

```
export PS1
```

Eksperimen hasil `PS1` :

```
$ PS1="\! > "
```

```
69 > PS1="\d > "
```

```
Mon Sep 23 > PS1="\t > "
```

```
10:10:20 > PS1="Saya=\u > "
```

```
Saya=stD02001 > PS1="\w >"
```

```
~ > PS1="\h >"
```

- b. Ubahlah warna shell prompt dengan warna biru dan berkedip.

6. Bash script

- a. Buat 3 buah script p1.sh, p2.sh, p3.sh dengan isi masing-masing :

```
p1.sh
```

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

```
echo "Program p1"
```

```
ls -l
```

```
p2.sh
```

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

```
echo "Program p2"
```

```
who
```

```
p3.sh
```

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

```
echo "Program p3"
```

```
ps x
```

- b. Jalankan script tersebut sebagai berikut dan perhatikan hasilnya :

```
$ ./p1.sh ; ./p3.sh ; ./p2.sh
```

```
$ ./p1.sh &
```

```
$ ./p1.sh $ ./p2.sh & ./p3.sh &
```

```
$ ( ./p1.sh ; ./p3.sh ) &
```

7. Jobs

- a. Buat shell- script yang melakukan loop dengan nama pwaktu.sh, setiap 10 detik, kemudian menyimpan tanggal dan jam pada file hasil.

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

```
while [ true ]
```

```
do
```

```
date >> hasil
```

```
sleep 10
```

done

- b. Jalankan sebagai background; kemudian jalankan satu program (utilitas find) di background sebagai berikut :

```
$ jobs
```

```
$ find / -print > files 2>/dev/null &
```

```
$ jobs
```

- c. Jadikan program ke 1 sebagai foreground, tekan ^Z dan kembalikan program tersebut ke background

```
$ fg %1
```

```
$ bg
```

- d. Stop program background dengan utilitas kill

```
$ ps x
```

```
$ kill [Nomor PID]
```

LAPORAN RESMI

1. Analisa hasil percobaan yang Anda lakukan.
 2. Kerjakan latihan diatas dan analisa hasil tampilannya.
 3. Berikan kesimpulan dari praktikum ini.
-

Modul VIII

Pemrograman Shell 2

SASARAN

Setelah mempelajari materi dalam bab ini, mahasiswa diharapkan mampu:

1. Mempelajari elemen dasar shell script
2. Membuat program shell interaktif
3. Menggunakan parameter dalam program
4. Mempelajari test kondisi serta operator logic yang terkait dengan instruksi test
5. Mengenal variable built-in dari shell
6. Membuat aplikasi dengan shell menggunakan konstruksi if-then-else
7. Menggunakan struktur case – esac.
8. Loop dengan while, for, do while.
9. Membuat fungsi dan mengetahui cara memanggil fungsi tersebut.

PRAKTIKUM

1. Login sebagai user.
2. Bukalah Console Terminal dan lakukan percobaan-percobaan di bawah ini kemudian analisa hasil percobaan.
3. Selesaikan soal-soal latihan.

Percobaan 1 : Membuat Shell Script

1. Buatlah file prog01.sh dengan editor vi

```
$ vi prog01.sh
#!/bin/sh
# Program shell
#
var1=x
var2=8
```

2. Untuk menjalankan shell, gunakan notasi TITIK di depan nama program

```
$ . prog01.sh
```

3. Untuk menjalankan shell, dapat juga dengan membuat executable file dan dieksekusi relatif dari current directory

```
$ chmod +x prog01.sh
$ ./prog01.sh
```

Percobaan 2 : Variabel

1. Contoh menggunakan variable pada shell interaktif

```
$ VPT= Teknik Informatika
$ echo $VPT
```

2. Pemisahan 2 kata dengan spasi menandakan eksekusi 2 buah instruksi. Karakter \$ harus ada pada awal nama variable untuk melihat isi variable tersebut, jika tidak, maka echo akan mengambil parameter tersebut sebagai string.

```
$ VPT2= Teknik Informatika Amikom (Terdapat pesan error)
$ VPT2=" Teknik Informatika Amikom"
$ echo VPT2
$ echo $VPT2
```

3. Menggabungkan dua variable atau lebih

```
$ V1= Teknik Informatika
$ V2=": "
$ V3=Amikom
$ V4=$V1$V2$V3
$ echo $V4
```

4. Menggabungkan isi variable dengan string yang lain. Jika digabungkan dengan nama variable yang belum didefinisikan (kosong) maka instruksi echo menghasilkan string kosong. Untuk menghindari kekeliruan, nama variable perlu diproteksi dengan { } dan kemudian isi variable tersebut digabung dengan string.

```
$ echo $V3
$ echo $V3Amikom
$ echo ${V3}Amikom
```

5. Variabel dapat berisi instruksi, yang kemudian bila dijadikan input untuk shell, instruksi tersebut akan dieksekusi

```
$ CMD=who
$ $CMD
$ CMD="ls -l"
$ $CMD
```

6. Modifikasi file `prog01.sh` berikut

```
$ vi prog01.sh
#!/bin/sh
V1= Teknik Informatika
V2=": "
V3=UMRAH
echo "Pemrograman shell"
echo $V1$V2$V3
V3=ITS
echo $V1$V2 di $V3
```

7. Cara sederhana mengeksekusi shell adalah dengan menggunakan notasi titik di depan nama shell script tersebut. Bila direktori actual tidak terdaftar dalam PATH, maka command tersebut tidak dapat ditemukan. Bila script belum executable, script tidak dapat dieksekusi.

```
$ . prog01.sh
$ prog01.sh (Terdapat pesan error)
$ ./prog01.sh (Terdapat pesan error)
$ chmod +x prog01.sh
$ ./prog01.sh
```

Percobaan 3 : Membaca Keyboard

1. Menggunakan instruksi `read`

```
$ read nama
Dony
$ echo $nama
```

2. Membaca nama dan alamat dari keyboard

```
$ vi prog02.sh
#!/bin/sh
# prog02.sh
# membaca nama dan alamat
echo "Nama Anda : "
read nama
echo "Alamat : "
read alamat
echo "Kota : "
read kota
echo
echo "Hasil adalah : $nama, $alamat di $kota"
```

3. Eksekusi program prog02.sh

```
$ . prog02.sh
Nama Anda :
Dony
Alamat :
Jl Nglempong Sari IV
Yogyakarta
Hasil adalah : Dony, Jl Nglempong Sari IV di Yogyakarta
```

4. Instruksi echo secara otomatis memberikan baris baru, maka untuk menghindari hal tersebut disediakan opsi -n, yang menyatakan kepada echo untuk menghilangkan baris baru. Modifikasi program prog02.sh

```
$ vi prog02.sh
#!/bin/sh
# prog02.sh
# membaca nama dan alamat
echo -n "Nama Anda : "
read nama
```

```
echo -n "Alamat : "  
read alamat  
echo -n "Kota : "  
read kota  
echo  
echo "Hasil adalah : $nama, $salamat di $kota"
```

5. Eksekusi program prog02.sh

```
$ . prog02.sh  
Nama Anda : Dony  
Alamat : Jl Nglempong Sari IV  
Kota : Yogyakarta  
Hasil adalah : Dony, Jl Nglempong Sari IV di Yogyakarta
```

6. Variabel kosong adalah variable yang tidak mempunyai nilai. Variabel ini didapat atas assignment atau membaca dari keyboard atau variable yang belum didefinisikan

```
$ read nama  
<CR>  
$ echo $nama  
$ A=  
$ B=""  
$ C=$A$B  
$ echo $C
```

7. Variabel dapat disubstitusikan dengan hasil eksekusi dari sebuah instruksi. Pada contoh dibawah , instruksi pwd dieksekusi lebih dahulu dengan sepasang Back Quate (tanda kutip terbalik). Hasil dari eksekusi tersebut akan masuk sebagai nilai variable DIR

```
$ pwd  
$ DIR=`pwd`  
$ echo $DIR
```

8. Buatlah shell script prog03.sh

```
$ vi prog03.sh  
#!/bin/sh
```

```
# prog03.sh
#
NAMA=`whoami`
echo Nama Pengguna Aktif adalah $NAMA
tanggal=`date | cut -c1-10`
echo Hari ini tanggal $tanggal
```

9. Eksekusi prog03.sh

```
$ . prog03.sh
```

Percobaan 4 : Parameter

1. Membuat shell script prog04.sh

```
$ vi prog04.sh
#!/bin/sh
# prog04.sh versi 1
# Parameter passing
#
echo "Nama program adalah $0"
echo "Parameter 1 adalah $1"
echo "Parameter 2 adalah $2"
echo "Parameter 3 adalah $3"
```

2. Eksekusi prog04.sh tanpa parameter, dengan 2 parameter, dengan 4 parameter

```
$ . prog04.sh
$ . prog04.sh Dony Ariyus
$ . prog04.sh Dony Ariyus Randyka Fransisco
```

3. Membuat shell script prog04.sh versi 2 dengan memberikan jumlah parameter

```
$ vi prog04.sh
#!/bin/sh
# prog04.sh versi 2
# Parameter passing
#
echo "Jumlah parameter yang diberikan adalah $#"
```

```
echo "Nama program adalah $0"
echo "Parameter 1 adalah $1"
echo "Parameter 2 adalah $2"
echo "Parameter 3 adalah $3"
```

4. Eksekusi `prog04.sh` tanpa parameter dan dengan 4 parameter

```
$ . prog04.sh
$ . prog04.sh Dony Ariyus Randyka Pransisco
```

5. Membuat shell script `prog04.sh` versi 3 dengan menambahkan total parameter dan nomor proses id (PID)

```
$ vi prog04.sh
#!/bin/sh
# prog04.sh versi 3
# Parameter passing
#
echo "Jumlah parameter yang diberikan adalah $#"
```

```
echo "Nama program adalah $0"
echo "Parameter 1 adalah $1"
echo "Parameter 2 adalah $2"
echo "Parameter 3 adalah $3"
echo "Total parameter adalah $*"
echo "PID proses shell ini adalah $$"
```

6. Eksekusi `prog04.sh` dengan 4 parameter

```
$ . prog04.sh Dony Ariyus Randyka Pransisco
```

Percobaan 5 : Status Exit

1. String tidak diketemukan, maka status exit adalah 1

```
$ grep xyz /etc/passwd
$ echo $?
```

2. String diketemukan, maka status exit adalah 0

```
$ grep <user> /etc/passwd
$ echo $?
```

Percobaan 6 : Konstruksi If

1. Instruksi dengan exit status 0

```
$ who
$ who | grep <user>
$ echo $?
```

2. If membandingkan exit status dengan 0, bila sama, maka blok program masuk ke dalam blok then-fi

```
$ if [ $? = 0 ]
> then
> echo      "Pemakai tersebut sedang aktif"
> fi
```

3. Nomor (1) dan (2) diatas dapat disederhanakan dengan

```
$ if who|grep <user> >/dev/null
> then
> echo      okay
> fi
```

Percobaan 7 : Konstruksi If Then Else

1. Membuat shell script prog05.sh

```
$ vi prog05.sh
#!/bin/sh
# prog05.sh
# Program akan memberikankonfirmasi apakah nama
# user sedang aktif atau tidak
#
echo -n "Berikan nama pemakai : "
read nama
if who | grep $nama > /dev/null
then
    echo "$nama sedang aktif"
else
```

```
        echo "$nama tidak aktif"
    fi
```

2. Jalankan `prog05.sh`, masukkan nama pemakai yang aktif yang tampil pada instruksi `who` dan coba juga untuk nama pemakai yang tidak aktif

```
$ who
$ . prog05.sh [nama=<user>]
$ . prog05.sh [nama=studentOS]
```

Percobaan 8 : Instruksi Test

1. Menggunakan instruksi test, perhatikan spasi antara

```
$ NAMA=Dony
$ test $NAMA = dony
$ echo $?
$ test $NAMA = Chevin
$ echo $?
```

2. Aplikasi test dengan konstruksi if

```
$ vi prog06.sh
#!/bin/sh
# prog06.sh
echo -n "NAMA = "
read NAMA
if test "$NAMA" = dony
then
echo "Selamat Datang $NAMA"
else
echo "Anda bukan dony, sorry!"
fi
```

3. Jalankan program `prog06.sh` dengan memasukkan `NAMA = amir` dan `NAMA = <CR>` perhatikan hasil tampilannya

```
$ . prog06.sh [NAMA = Dony]
$ . prog06.sh [NAMA = <CR>] (Terdapat pesan error)
```

4. Modifikasi `prog06.sh` dengan menggunakan notasi untuk test

```
$ vi prog06.sh
#!/bin/sh
# prog06.sh
echo -n "NAMA = "
read NAMA
if [ "$NAMA" = Dony ]
then
echo "Selamat Datang $NAMA"
else
echo "Anda bukan Dony, sorry!"
fi
```

5. Jalankan program `prog06.sh` dengan memasukkan `NAMA = amir`

```
$ . prog06.sh [NAMA = Dony]
```

Percobaan 9 : Notasi `&&` dan `||`

1. Bila file `prog01.sh` ada (TRUE), maka jalankan program berikutnya. File `prog01.sh` ada, karena itu exit status adalah TRUE, hasil operasi AND masih tergantung pada hasil eksekusi instruksi ke 2, dan dengan demikian instruksi `echo` akan dijalankan.

```
$ [ -f prog01.sh ] && echo "Prog01.sh ada"
```

2. File `prog99.sh` tidak ada, karena itu exit status adalah FALSE dan instruksi `echo` tidak dijalankan

```
$ [ -f prog99.sh ] && echo "Prog99.sh ada"
```

3. Bila `prog01.sh` ada maka jalankan shell script tersebut

```
$ [ -f prog01.sh ] && . prog01.sh
```

4. Bila `prog01.sh` ada maka jalankan program berikutnya. File `prog01.sh` memang ada, karena itu exit status adalah TRUE, dan karena sudah TRUE maka instruksi `echo` tidak lagi dijalankan

```
$ [ -f prog01.sh ] || echo "Dieksekusi tidak ?"
```

5. File `prog99.sh` tidak ada, karena itu exit status adalah FALSE, hasil masih tergantung atas exit status instruksi ke dua, karena itu instruksi `echo` dijalankan

```
$ [ -f prog99.sh ] || echo "Dieksekusi tidak ?"
```

6. File `prog99.sh` tidak ada, maka tampilkan pesan error `$ [-f prog99.sh] || echo "Sorry, prog99.sh tidak ada"`

Percobaan 10 : Operator Bilangan Bulat untuk Test

1. Menggunakan operator dengan notasi test

```
$ i=5
$ test "$i" -eq 5
$ echo $?
```

2. Menggunakan operator dengan notasi `[]` (penganti notasi test)

```
$ [ "$i" -eq 5 ]
$ echo $?
```

Percobaan 11 : Operatot Logical dan Konstruksi Elif

1. Buatlah file `prog07.sh`

```
$ vi prog07.sh
#!/bin/sh
# prog07.sh
echo -n "INCOME = "
read INCOME
if [ $INCOME -ge 0 -a $INCOME -le 10000 ]
then
    BIAYA=10
elif [ $INCOME -gt 10000 -a $INCOME -le 25000 ]
then
    BIAYA=25
else
    BIAYA=35
fi
echo "Biaya = $BIAYA"
```

2. Jalankan file `prog07.sh` dan masukkan untuk `INCOME=5000, 20000, 28000`

```
$ . prog07.sh [INCOME=5000]
```

```
$ . prog07.sh [INCOME=20000]
```

```
$ . prog07.sh [INCOME=28000]
```

Percobaan 12 : Hitungan Aritmetika

1. Menggunakan utilitas expr

```
$ expr 5 + 1
```

```
$ A=5
```

```
$ expr $A + 2
```

```
$ expr $A - 4
```

```
$ expr $A * 2 (Ada Pesan Error)
```

```
$ expr $A \* 2
```

```
$ expr $A / 6 +10
```

```
$ expr 17 % 5
```

2. Substitusi isi variable dengan hasil utilitas expr

```
$ A=5
```

```
$ B=`expr $A + 1`
```

```
$ echo $B
```

Percobaan 13 : Instruksi Exit

1. Buat shell script prog08.sh

```
$ vi prog08.sh
```

```
#!/bin/sh
```

```
if [ -f prog01.sh ]
```

```
then
```

```
    exit 3
```

```
else
```

```
    exit -1
```

```
fi
```

2. Jalankan script prog08.sh dan periksa status exit

```
$ . prog08.sh
```

```
$ echo $?
```

Percobaan 14 : Konstruksi Case – Esac

1. Buatlah file prog09.sh dengan editor vi

```
$ vi prog09.sh
```

```
#!/bin/sh
# Prog: prog09.sh
echo "1. Siapa yang aktif"
echo "2. Tanggal hari ini"
echo "3. Kalender bulan ini"
echo -n " Pilihan : "
read PILIH
case $PILIH in
1)
    echo "Yang aktif saat ini"
    who
    ;;
2)
    echo "Tanggal hari ini"
    date
    ;;
3)
    echo "Kalender bulan ini"
    cal
    ;;
*)
    echo "Salah pilih !!"
    ;;
esac
```

2. Jalankan program prog09.sh, cobalah beberapa kali dengan inputan yang berbeda

```
$ . prog09.sh
```

3. Buatlah file prog10.sh yang merupakan bentuk lain dari case

```
$ vi prog10.sh
#!/bin/sh
# Prog: prog10.sh
```

```
echo -n "Jawab (Y/T) : "  
read JWB
```

```
case $JWB in  
y | Y | ya | Ya | YA ) JWB=y ;;  
t | T | tidak | Tidak | TIDAK ) JWB=t ;;  
esac
```

4. Jalankan program prog10.sh, cobalah beberapa kali dengan inputan yang berbeda

```
$ . prog10.sh
```

5. Modifikasi file prog10.sh yang merupakan bentuk lain dari case

```
$ vi prog10.sh  
#!/bin/sh  
# Prog: prog10.sh
```

```
echo -n "Jawab (Y/T) : \c"  
read JWB
```

```
case $JWB in  
[yY] | [yY][aA] ) JWB=y ;;  
[tT] | [tT]idak ) JWB=t ;;  
*) JWB=? ;;  
esac
```

6. Jalankan program prog10.sh, cobalah beberapa kali dengan inputan yang berbeda

```
$ . prog10.sh
```

Percobaan 15 : Konstruksi for-do-done

1. Buatlah file prog11.sh

```
$ vi prog11.sh  
#!/bin/sh  
# Prog: prog11.sh
```

```
for NAMA in Dony Ariyus Randyka Fransisco
do
    echo "Nama adalah : $NAMA"
done
```

2. Jalankan program prog11.sh

```
$ . prog11.sh
```

3. Buatlah file prog12.sh yang berisi konstruksi for dan wildcard, program ini akan menampilkan nama file yang berada di current direktori

```
$ vi prog12.sh
#!/bin/sh
# Prog: prog12.sh
```

```
for F in *
do
    echo $F
done
```

4. Jalankan program prog12.sh

```
$ . prog12.sh
```

5. Modifikasi file prog12.sh, program ini akan menampilkan long list dari file yang mempunyai ekstensi lst

```
$ vi prog12.sh
#!/bin/sh
# Prog: prog12.sh
```

```
for F in *.lst
do
    ls -l $F
done
```

6. Jalankan program prog12.sh

```
$ . prog12.sh
```

Percobaan 16 : Konstruksi While-Do-Done

1. Buatlah file prog13.sh

```
$ vi prog13.sh
```

```
#!/bin/sh
```

```
# Prog: prog13.sh
```

```
PILIH=1
```

```
while [ $PILIH -ne 4 ]
```

```
do
```

```
    echo "1. Siapa yang aktif"
```

```
    echo "2. Tanggal hari ini"
```

```
    echo "3. Kalender bulan ini"
```

```
    echo "4. Keluar"
```

```
    echo " Pilihan : \c"
```

```
    read PILIH
```

```
    if [ $PILIH -eq 4 ]
```

```
    then
```

```
        break
```

```
    fi
```

```
    clear
```

```
done
```

```
echo "Program berlanjut di sini setelah break"
```

2. Jalankan program prog13.sh

```
$ . prog13.sh
```

Percobaan 17 : Instruksi Dummy

1. Modifikasi file prog13.sh

```
$ vi prog13.sh
```

```
#!/bin/sh
```

```
# Prog: prog13.sh
```

```
PILIH=1
```

```
while :
```

```
do
```

```
    echo "1. Siapa yang aktif"
```

```
    echo "2. Tanggal hari ini"
```

```
    echo "3. Kalender bulan ini"
```

```
    echo "4. Keluar"
```

```
    echo " Pilihan : \c"
```

```
    read PILIH
```

```
    if [ $PILIH -eq 4 ]
```

```
    then
```

```
        break
```

```
    fi
```

```
    clear
```

```
done
```

```
echo "Program berlanjut di sini setelah break"
```

2. Jalankan program prog13.sh

```
$ . prog13.sh
```

3. Buatlah file prog14.sh yang berisi instruksi dummy untuk konstruksi if

```
$ vi prog14.sh
```

```
#!/bin/sh
```

```
# Prog: prog14.sh
```

```
echo -n "Masukkan nilai : "
```

```
read A
```

```
if [ $A -gt 100 ]
```

```
then
```

```
:
```

```
else
    echo "OK !"
fi
```

4. Jalankan program `prog14.sh` beberapa kali dengan input yang berbeda

```
$ . prog14.sh
```

Percobaan 18 : Fungsi

1. Buatlah file `fungsi.sh`

```
$ vi fungsi.sh
#!/bin/sh
# Prog: fungsi.sh

F1( ) {
    echo "Fungsi F1"
    return 1
}
echo "Menggunakan Fungsi"
F1
F1
echo $?
```

2. Jalankan program `fungsi.sh`

```
$ . fungsi.sh
```

3. Menggunakan variable pada fungsi dengan memodifikasi file `fungsi.sh`

```
$ vi fungsi.sh
#!/bin/sh
# Prog: fungsi.sh

F1( )
{
    Honor=10000
    echo "Fungsi F1"
```

```

        return 1
    }
    echo "Menggunakan Fungsi"
    F1
    F1
    echo "Nilai balik adalah $?"
    echo "Honor = $Honor"

```

4. Jalankan program fungsi.sh

```
$ . fungsi.sh
```

5. Menggunakan variable pada fungsi dengan memodifikasi file fungsi.sh

```
$ vi fungsi.sh
#!/bin/sh
# Prog: fungsi.sh

```

```

F1( )
{
    local Honor=10000
    echo "Fungsi F1"
    return 1
}
echo "Menggunakan Fungsi"
F1
F1
echo "Nilai balik adalah $?"
echo "Honor = $Honor"

```

6. Jalankan program fungsi.sh

```
$ . fungsi.sh
```

LATIHAN

1. Buatlah program **salin.sh** yang menyalin file (copy) sebagai berikut : *salin.sh file-asal file-tujuan* Dengan ketentuan :
 - a. Bila file asal tidak ada, berikan pesan, salin gagal.
 - b. Bila file tujuan ada dan file tersebut adalah directory, beri pesan bahwa file tidak bisa disalin ke direktori
 - c. Bila file tujuan ada dan file biasa, beri pesan apakah file tersebut akan dihapus, bila dijawab dengan "Y", maka copy file tersebut
 - d. Bila file tujuan belum ada, lakukan copy

Untuk mengambil nama file, gunakan parameter \$1 dan \$2. Bila jumlah parameter tidak sama (\$#) dengan 2, maka beri pesan exit = -1

```
#!/bin/sh
# file: salin.sh
# Usage: salin.sh fasal ftujuan
if [ $# -ne 2 ]
then
    echo "Error, usage: salin.sh file-asal file-tujuan"

    exit -1
fi
fasal=$1
ftujuan=$2
echo "salin.sh $fasal $ftujuan"
.....
.....
```

2. Buat program yang memeriksa nama direktori, jika parameter tersebut adalah direktori, maka jalankan instruksi `ls -ld` pada direktori tersebut. Namakan program tersebut **checkdir.sh**. Gunakan notasi `[-d NamaDirektori]` dan pilih logika `&&` atau `||` pada level shell.
-

```
#!/bin/sh
# file: checkdir.sh
# Usage: checkdir.sh DirectoryName
#
if [ $# -ne 1]
then
    echo "Error, usage: checkdir.sh DirectoryName"
    exit 1
fi
[ ... ] && ...
```

3. Dengan shell script **pph.sh**, hitung PPH per tahun dengan ketentuan sebagai berikut:

- a. 10 juta pertama PPH 15%
- b. 25 juta berikutnya (sisanya) PPH 25%
- c. Bila masih ada sisa, maka sisa tersebut PPH 35%

Contoh :

Gaji 8 juta

PPH = 15% * 8 juta

Gaji 12 juta

PPH = 15% * 10 juta + 25% * (12-10) juta

Gaji 60 juta

PPH = 15% * 10 juta + 25% * 25 juta + 35% * (60-10-25) juta

Debugging : untuk melakukan tracing (debug) gunakan opsi `-x` pada eksekusi shell.

```
$ sh -x pph.sh
+ echo -n „Berikan gaji dalam ribuan rupiah : „
Berikan gaji dalam ribuan rupiah : + read gaji
20000
+ pcp=10000
+ „[„ 20000 -le 10000 „]“
```

```

++ expr 20000 - 10000
+ gaji=10000
+ pph=1500
+ pkp=25000
+ „[„ 10000 -le 25000 „]“
+ pkp=10000
++ expr 1500 + 10000 „*“ 25 / 100
+ pph=4000
+ echo „Pajak Penghasilan = 4000“
Pajak Penghasilan = 4000

```

4. Buatlah program **myprog.sh** yang memproses parameter \$1, nilai parameter harus berupa string :

```

start
stop
status
restart
reload

```

Bila buka dari string tersebut, maka berikan pesan error. Sempurnakan program di bawah ini untuk keperluan tersebut

```

#!/bin/sh
# See how we were called
case "$1" in
    start)
        echo "Ini adalah start"
        ;;
    stop)
        echo "Ini adalah stop"
        ;;
    *)
        echo $"Usage:$0 {start|stop|restart|reload|status}"

```

```
;;  
esac  
return
```

5. Buat sebuah fungsi pada script **confirm.sh** yang memberikan konfirmasi jawaban **Yes**, **No** atau **Continue**. Jika jawaban Yes, maka beri nilai balik 0, No = 1 dan Continue = 2. Modifikasi kerangka program berikut untuk memenuhi permintaan tersebut.

```
#!/bin/sh  
# Confirm whether we really want to run this service  
confirm() {  
    local YES="Y"  
    local NO="N"  
    local CONT="C"  
while :  
do  
    echo -n "(Y)es/(N)o/(C)ontinue? {Y} "  
    read answer  
    answer=`echo "$answer" | tr „[a-z]" „[A-Z]"`  
  
    if [ "$answer" = "" -o "$answer" = $YES ]  
    then  
        return 0  
    elif ...  
    then  
        return 2  
    elif ...  
    then  
        return 1  
    fi  
done  
}
```

Test fungsi diatas dengan program berikut :

```
$ vi testp.sh
. confirm.sh
confirm
if [ $? -eq 0 ]
then
    echo "Jawaban YES OK"
elif [ $? =eq 1 ]
then
    echo "Jawaban NO"
else
    echo "Jawaban CONTINUE"
fi
```

Perhatikan baris pertama, adalah loading dari fungsi confirm yang terdapat di script confirm.sh. Setelah eksekusi script tersebut, maka fungsi confirm dapat digunakan.

LAPORAN RESMI

1. Analisa hasil percobaan yang Anda lakukan.
 2. Kerjakan latihan diatas dan analisa hasil tampilannya.
 3. Berikan kesimpulan dari praktikum ini.
-