

Membangun Personal Linux Assistant Sendiri

Khawatir terjadi perubahan di sistem Anda? Terlalu sibuk untuk mengamatinya secara berkala? Program pemantau lain terlalu kompleks? Kurang personal? Kenapa tidak membangunnya sendiri? Cukup modal bash dan cron!

Anda adalah seorang administrator jaringan kecil. Dalam kesehariannya, Anda memiliki banyak kesibukan sambil tetap harus menjaga agar server Anda tidak diganggu oleh pihak-pihak tidak bertanggung jawab. Mengamati sendiri secara berkala dan manual tentunya hanya akan menghabiskan waktu. Sementara, pilihan program pemantau sistem terkadang cukup kompleks. Anda butuh yang berfungsi sesuai dengan keinginan Anda. Apabila Anda memiliki waktu luang barang satu atau dua jam, mari membangun *Personal Linux assistant* sendiri.

Pertama-tama yang kita perlukan adalah mengetahui secara persis apa yang ingin kita monitor secara berkala. Sebagai seorang pemilik sistem, adalah wajib untuk mengetahui perubahan mana yang mungkin terjadi. Mungkin ada di antara Anda yang peduli akan perubahan beberapa file penting. Ada pula yang ingin mengamati perubahan *hardware* ataupun berbagai *peripheral* fisik server. Ada pula yang merasa perlu untuk mengamati jumlah proses atau batas terendah memori yang tersisa.

Tulisan ini akan memberikan beberapa contoh modul untuk mengamati perubahan informasi CPU, perubahan perangkat PCI, informasi beberapa file, dan pengamatan pada batas terendah memory yang tersisa. Modul? Ya, aplikasi yang ingin kita bangun, *Personal Linux assistant*, akan bekerja secara modular. Penambahan modul tidak memerlukan pengubahan sama sekali di aplikasi utama. Dan selanjutnya seorang asisten, PLA pun dapat menyajikan laporan untuk kita. Kita akan membuat PLA langkah demi langkah.

1. Fitur-fitur PLA

PLA yang ingin kita bangun akan bekerja dalam *command line interface*. Untuk meningkatkan fleksibilitas, PLA dapat kita minta untuk mencatat perubahan terbaru yang sebenarnya (membangun database baru). PLA juga dapat diminta untuk memonitor perubahan secara manual. Versi PLA yang akan kita bangun akan terus mengamati perubahan dan tidak menghapus data pengamatan sebelumnya.

Sifat yang satu ini akan berakibat pada banyaknya jumlah direktori yang menampung informasi perubahan. Setiap kali melakukan pengamatan, PLA akan membuat direktori baru dengan nama direktori adalah tanggal dan waktu saat itu, kemudian mengisinya dengan catatan perubahan. Data perubahan dibandingkan antara data saat itu dengan data asli yang disimpan. Untuk mencegah agar tidak ada sampah laporan yang terlalu menumpuk, maka PLA juga harus mampu untuk menghapus semua laporan.

Kemudian, pengguna dapat meminta PLA untuk memberikan laporan secara interaktif. PLA hanya akan melaporkan perubahan yang terjadi. Dalam hal ini, kita akan memanfaatkan program *diff*. Dan yang terakhir, PLA dapat menampilkan informasi versi, hak cipta, dan lisensi.

2. Struktur direktori PLA

Kita akan memiliki satu buah file *executable*, dengan nama file adalah *pla*. Kemudian, di direktori yang sama, kita akan memiliki direktori *init*, yang berfungsi untuk menampung data asli. PLA dapat diminta untuk memperbarui data di direktori *init* tersebut. Kemudian,



ada pula direktori *monitor* yang akan menampung data pengamatan. Dan direktori *var*, yang menyimpan modul PLA. Berikut ini adalah visualisasi struktur direktori PLA:

```
+ init
+ monitor
+ var
-pla
```

3. Struktur modul PLA

Sebuah modul PLA harus memiliki minimal dua fungsi, yakni fungsi *init()* dan fungsi *monitor()*. Fungsi-fungsi tambahan dapat dibuat sendiri oleh pengembang modul. Modul dapat diberikan nama apa saja, asalkan mencerminkan fungsinya. Modul juga harus menyeleksi kapan menjalankan fungsi *init()* ataupun *monitor()*. Kita akan merancang agar pembuatan modul menjadi semudah mungkin.

Pada prinsipnya, fungsi *init()* akan membuat data yang baru. Sementara, fungsi *monitor()* akan membandingkan data aktif dengan data yang dihasilkan dari fungsi *init()*. Fungsi *init()* hanya akan dijalankan apabila pengguna secara eksplisit meminta.

4. Membuat aplikasi utama: *pla*

Aplikasi utama harus dijalankan dengan memberikan aksi atau parameter tertentu. Aksi yang saat ini dimiliki adalah *init*, *monitor*, *purge*, *report*, dan *version*. Aksi *init* akan menyebabkan terjadinya pembuatan database baru. Aksi *monitor* akan memonitor perubahan. Aksi *purge* akan menyebabkan dihapusnya semua laporan perubahan. Aksi *report* akan menjalankan tugas pelaporan. Sementara aksi *version* akan menampilkan versi, hak

cipta, dan informasi lisensi. Apabila pengguna menjalankan pla tanpa menyebutkan aksi, maka kita akan menampilkan cara penggunaan, dan keluar sesudahnya.

Berikut ini adalah *source* lengkap dari program utama. Penjelasan akan diberikan setelahnya.

```
#!/bin/sh

COPYRIGHT="(c) Noprianto
(nop@keant.org), June 2003, GPL"
VERSION="pla 0.1a"
VAR="/root/NOP/pla/var/"
DATE=$(date)

case $1 in
init|monitor)
    echo "ok, $1."
    for plas in $(ls $VAR)
    do
        . $VAR/$plas $1 "$DATE"
    done
    ;;
purge)
    echo -n "purging data "
    rm -rf monitor
    mkdir monitor
    echo .
    ;;
report)
    reportdir=$(ls -rt monitor | tail -1)
    working="monitor/$reportdir"
    files=$(ls "$working")
    echo "reporting files in $working"
    PS3="[CTRL-C to abandon] "
    select menu in $files
    do
        test -f "$working/$menu" &&
        cat "$working/$menu"
    done
    ;;
version)
    echo $VERSION
    echo $COPYRIGHT
    ;;
*)
    echo "usage: `basename $0`
init|monitor|purge|report|version"
    exit 1
    ;;
esac
```

Variabel \$DATE dimaksudkan untuk mencatat tanggal aktif. Sementara variabel \$COPYRIGHT dan \$VERSION berguna untuk informasi versi, hak cipta, dan lisensi.

Selanjutnya, kita akan memeriksa isi dari parameter pertama. Apabila parameter pertama bernilai init atau monitor, maka blok berikut ini akan dijalankan:

```
init|monitor)
    echo "ok, $1."
    for plas in $(ls $VAR)
    do
        . $VAR/$plas $1 "$DATE"
    done
    ;;
```

Blok kode tersebut akan menjalankan setiap modul yang ada di direktori var. Kita memberikan dua parameter, yaitu init atau monitor dan \$DATE. Dengan adanya teks init atau monitor yang dikirimkan ke modul, maka modul dapat menjalankan fungsi yang bersesuaian.

Apabila parameter pertama bernilai purge, maka kode berikut ini akan dijalankan:

```
purge)
    echo -n "purging data "
    rm -rf monitor
    mkdir monitor
    echo .
    ;;
```

Kode tersebut secara sederhana akan menghapus direktori monitor dan seluruh isinya. Setelah itu, kita membuat kembali direktori monitor. Selanjutnya, apabila parameter pertama bernilai report, kode berikut ini akan dijalankan:

```
report)
    reportdir=$(ls -rt monitor | tail -1)
    working="monitor/$reportdir"
    files=$(ls "$working")
    echo "reporting files in $working"
    PS3="[CTRL-C to abandon] "
    select menu in $files
    do
        test -f "$working/$menu" && cat
        "$working/$menu"
    done
    ;;
```

Ini adalah blok kode terpanjang. Kita memiliki banyak direktori yang menyimpan perubahan. Masing-masing harus diurutkan terlebih dahulu.

Pengurutannya berdasarkan waktu agar kita bisa mendapatkan direktori yang paling baru dibuat. Isi dari direktori ini akan dilaporkan.

Baris reportdir=\$(ls -rt monitor | tail -1) dimaksudkan untuk mengurutkan keluaran ls berdasarkan waktu mulai yang paling lama ke yang paling baru. Keluaran tersebut segera ditangkap oleh program tail. Di sini, program tail kita gunakan untuk mengambil satu baris terakhir.

Baris working="monitor/\$reportdir" dan files=\$(ls "\$working") dimaksudkan untuk mendapatkan file yang terdapat di direktori kerja.

Sementara baris PS3="[CTRL-C to abandon] " dimaksudkan untuk mengubah *prompt* yang akan digunakan oleh perintah *select*. Dan akhirnya, kita menjalankan kode berikut untuk menampilkan pilihan bagi pengguna. Kita juga akan memeriksa terlebih dahulu apakah file yang ingin ditampilkan terdapat di file sistem.

```
select menu in $files
do
    test -f "$working/$menu" && cat
    "$working/$menu"
done
```

Harap diperhatikan, file-file tersebut hanya menyimpan perubahan. Apabila tidak ada perubahan yang terjadi, maka file tersebut adalah file kosong semata.

5. Membuat modul PLA: cpuinfo, pci, file_integrity dan memory

Di kesempatan kali ini, kita akan membuat empat modul contoh untuk PLA. Modul pertama adalah cpuinfo, yang berfungsi untuk mengamati perubahan pada CPU. Perubahan pada CPU memang hal yang tidak wajar, akan tetapi tetap menyenangkan untuk dibuat. Terutama karena pembuatannya sangatlah mudah dan singkat. Modul berikutnya adalah modul pci, yang berfungsi untuk mengamati perubahan pada pci komputer. Kedua modul ini

mengambil informasi dari `/proc`. Sementara modul ketiga, `file_integrity` sedikit lebih panjang. Modul yang satu ini mengamati perubahan pada file-file tertentu. Modul terakhir, `memory`, yang berfungsi untuk mengamati batas terendah `memory` tersisa, juga mengambil informasi dari `/proc`.

Kita akan mulai dengan modul pertama: `cpuinfo`. Berikut ini adalah *source code* selengkapnya. Penjelasan akan diberikan setelah *source code*.

```
function init()
{
    cp /proc/cpuinfo init/
    chmod u+w init/cpuinfo
}

function monitor()
{
    working="monitor/$1"
    if ! [ -d "$working" ]
    then
        mkdir "$working"
    fi
    diff /proc/cpuinfo init/cpuinfo >
"$working/cpuinfo"
}

case $1 in
init)
    init
    ;;

monitor)
    monitor "$2"
    ;;

esac
```

Pertama-tama, karena setiap modul menerima dua parameter yang dikirimkan dari program utama `pla`, maka kita harus mengetahui fungsi apa yang harus dipanggil berdasarkan parameter yang dikirimkan tersebut. Blok kode `case` berikut ini akan mengerjakannya:

```
case $1 in
init)
    init
    ;;

monitor)
    monitor "$2"
    ;;

esac
```

```
;;

esac
```

Sementara itu, pada fungsi `init`, kita hanya meng-*copy*-kan file `/proc/cpuinfo` ke direktori `init`. Pengubahan permissi dimaksudkan untuk mencegah pemilik kehilangan hak tulis (kita mungkin perlu menimpa file ini ketika melakukan `init` selanjutnya).

```
function init()
{
    cp /proc/cpuinfo init/
    chmod u+w init/cpuinfo
}
```

Pada fungsi `monitor`, pertama-tama kita akan memeriksa terlebih dahulu ada atau tidaknya direktori kerja untuk menyimpan hasil laporan. Apabila tidak ditemukan, maka direktori tersebut akan dibuat terlebih dahulu. Blok kode berikut ini akan mengerjakan pemeriksaan eksistensi direktori tersebut.

```
if ! [ -d "$working" ]
then
    mkdir "$working"
fi
```

Setelah itu, menggunakan program `diff`, kita akan memeriksa apakah file `/proc/cpuinfo` saat ini (file yang sebenarnya), sama dengan file `cpuinfo` yang terletak di direktori `init`. Hasil pengecekan akan disimpan di direktori laporan.

```
diff /proc/cpuinfo init/cpuinfo >
"$working/cpuinfo"
```

File-file di direktori kerja tempat menyimpan laporan inilah yang akan dibaca ketika kita memanggil:

```
pla report
```

Selanjutnya, kita akan membahas modul `pci`. Modul `pci` pada dasarnya hanyalah mengubah file yang dibaca menjadi file `/proc/pci`. Perbedaan selanjutnya dengan modul `cpuinfo` adalah nama file laporan yang dihasilkan. Berikut ini adalah *source code* selengkapnya. Penjelasan dapat merujuk ke penjelasan modul `cpuinfo`.

```
function init()
{
    cp /proc/pci init/
    chmod u+w init/pci
}

function monitor()
{
    working="monitor/$1"
    if ! [ -d "$working" ]
    then
        mkdir "$working"
    fi
    diff /proc/pci init/pci > "$working/pci"
}

case $1 in
init)
    init
    ;;

monitor)
    monitor "$2"
    ;;

esac
```

Kemudian, kita akan sampai pada modul ketiga, yaitu modul `file_integrity`. Di contoh modul ini, kita akan membandingkan keluaran dari perintah `ls -al` terhadap file-file yang ingin diperiksa integritasnya. Tidak lupa pula kita memasukkan keluaran dari perintah `md5sum` terhadap file-file tersebut untuk meningkatkan akurasi.

Kenapa integritas file harus dicatat? Salah satu kejahatan komputer adalah mengganti fungsi program tertentu dengan fungsi yang lain. Bagaimana kalau program `ls` yang pasti Anda jalankan setiap kali ternyata juga berusaha untuk memenuhi `harddisk` Anda dengan file sampah di samping tugas utamanya? Bagaimana pula kalau program `rm` yang Anda jalankan dengan tujuan menghapus file tertentu ternyata juga berfungsi ganda dan turut menghapus file penting di sistem?

Berikut ini adalah *source* selengkapnya. Penjelasan akan diberikan setelah *source code*.

```
function do_stat()
{
```

```

ls -al /bin/ls > $1
md5sum /bin/ls >> $1
ls -al /bin/rm >> $1
md5sum /bin/rm >> $1
ls -al /bin/mv >> $1
md5sum /bin/mv >> $1
}

function init()
{
    do_stat init/file_integrity
    chmod u+w init/file_integrity
}

function monitor()
{
    working="monitor/$1"
    if ! [ -d "$working" ]
    then
        mkdir "$working"
    fi

    tempfile=/tmp/${file_integrity}
    do_stat $tempfile
    diff $tempfile init/file_integrity >
"$working/file_integrity"
    rm -f $tempfile
}

case $1 in
init)
    init
    ;;

monitor)
    monitor "$2"
    ;;

esac

```

Fungsi tambahan `do_stat()` berikut ini adalah fungsi yang akan mencatat integritas file dan melaporkan ke file tertentu, sesuai parameter yang diterima. Tambahkan baris `ls` dan `md5sum` untuk file-file lain yang ingin Anda tambahkan. Bukan fungsi yang optimal, tapi setidaknya, it works (tm).

```

function do_stat()
{
    ls -al /bin/ls > $1
    md5sum /bin/ls >> $1
    ls -al /bin/rm >> $1

```

```

md5sum /bin/rm >> $1
ls -al /bin/mv >> $1
md5sum /bin/mv >> $1
}

Fungsi init() sama seperti fungsi init() pada dua modul sebelumnya. Sementara, fungsi monitor() sedikit lebih panjang dibandingkan fungsi monitor() pada dua modul sebelumnya.

tempfile=/tmp/${file_integrity}
do_stat $tempfile
diff $tempfile init/file_integrity >
"$working/file_integrity"
rm -f $tempfile

```

Blok kode tersebut dimaksudkan untuk memanggil fungsi `do_stat()` dan menyimpan hasilnya ke dalam file sementara di direktori `/tmp`. Kemudian, file tersebut akan dibandingkan dengan file `file_integrity` di direktori `init`. Setelah perbandingan selesai, file sementara tersebut kemudian dihapus.

Dan yang terakhir, kita akan membahas tentang modul `memory`, yang berfungsi untuk mengamati batas terendah `memory` tersisa. Berikut ini adalah source selengkapannya. Penjelasan akan diberikan setelahnya.

```

function init()
{
    echo 7000 > init/memory
    chmod u+w init/memory
}

function monitor()
{
    working="monitor/$1"
    if ! [ -d "$working" ]
    then
        mkdir "$working"
    fi

    MEMFREENOW='cat /proc/meminfo | grep -w MemFree | awk '{print $2}''
    MEMFREE='cat init/memory'
    if [ $MEMFREENOW -lt $MEMFREE ]
    then
        echo "On $1, MemFree is less than $MEMFREE" > "$working/memory"
    else
        echo "MemFree normal" >

```

```

"$working/memory"
fi
}

case $1 in
init)
    init
    ;;

monitor)
    monitor "$2"
    ;;

esac

```

Untuk mengambil `memory` yang tersisa, kita dapat membaca file `/proc/meminfo`. Baris berikut ini akan melakukan *parsing* dan mengambil nilainya:

```
MEMFREENOW='cat /proc/meminfo |
grep -w MemFree | awk '{print $2}''
```

6. Menjalankan pla secara berkala

Sampai saat ini, PLA telah dapat digunakan dengan memberikan tugas-tugas tertentu. Tapi, cukupkah semua itu? Belum. Kita tidak ada waktu untuk memonitor secara berkala. Kita butuh menjalankan PLA secara otomatis dan berkala. Untuk itu, *cron* adalah tool yang paling cocok untuk digunakan.

Login-lah sebagai `root` dan jalankanlah perintah berikut ini:

```
crontab -e
```

Program ini akan meluncurkan editor teks. Berikan baris berikut untuk menjalankan PLA setiap menit sekali:

```
***** cd /root/NOP/pla/ ; ./pla monitor
```

Gantilah `/root/NOP/pla` sesuai dengan konfigurasi Anda. Setiap kali *cron* dijalankan, maka sebuah mail akan dikirimkan kepada *root*. Apabila Anda menginginkan agar mail dikirimkan kepada user lain, maka tambahkan variabel `MAILTO=<nama_user_ain>` sebelum penjadwalan.

Demikianlah perkenalan kita dengan PLA. Kembangkan sendiri modul Anda, dan perbaikilah program PLA bagi Anda yang berminat. ¹

Noprianto (noprianto@infolinux.co.id)