

# Pengunci Terminal dengan Python

Sering meninggalkan *login* di terminal? Sering *diisengin* rekan kerja? Kenapa tidak membuat penguncinya sendiri?

**M**asalah keamanan komputer adalah hal yang kompleks. Seseorang dengan niat jahat dan kemampuan teknis yang mencukupi dapat membobol sistem komputer, baik dengan memanfaatkan kelemahan sistem ataupun menggunakan berbagai *tool* yang tersedia. Di luar semua itu, sebenarnya, bagian terbesar dari keamanan komputer terletak di sisi penggunaannya sendiri. Bagaimana mungkin suatu sistem yang seharusnya aman dapat tetap aman apabila penggunaannya selalu meninggalkan login-nya sehingga siapapun dapat mempergunakan sesi login tersebut untuk melakukan kejahatan?

Antisipasi dengan mengatur variabel `TMOUT` mungkin membantu. Akan tetapi, hal tersebut sekaligus dapat menjadi hal yang menyebalkan. Bayangkan saja, Anda mungkin menjalankan beberapa terminal *console* dan bekerja pada salah satunya secara aktif. Anda mungkin butuh untuk berpindah ke terminal lain dalam selang waktu yang tidak pasti. Apabila variabel `TMOUT` diberikan nilai beberapa menit misalnya, maka ketika Anda bekerja di terminal yang satu, terminal yang lain mungkin saja telah melakukan *logout* otomatis. Apabila `TMOUT` diberikan nilai yang terlalu lama, maka bisa-bisa menjadi celah bagi terganggunya keamanan sistem Anda. Untuk itu, sebuah pengunci terminal mungkin dapat membantu dalam kasus tertentu.

## Signal

Suatu pengunci terminal pada dasarnya harus memperhatikan bagaimana mengatur signal yang dikirimkan kepada dirinya. Pengunci tersebut, harus tahan akan segala signal mematikan kecuali

signal yang memang tidak dapat ditahan. Sebelum kita membahas tentang signal di Linux, ada baiknya untuk mengenal signal dan penanganannya terlebih dahulu.

Signal adalah salah satu cara paling tua dari komunikasi antarproses. Seperti yang kita ketahui, Linux adalah sistem operasi *multitasking*, di mana mampu menangani berbagai proses. Signal sendiri dapat dikirimkan secara manual ataupun secara otomatis dan bekerja secara asinkron.

Kita bisa menganalogikan signal dalam kehidupan sehari-hari. Anda sedang bekerja memindahkan barang dari suatu lokasi ke lokasi lainnya. Kemudian teman Anda berseru kepada Anda dan mengatakan bahwa sudah saatnya bagi Anda untuk makan siang dan beristirahat sejenak. Dalam hal ini, signal berupa panggilan dari teman Anda dan dikirimkan secara manual. Setelah Anda makan dan beristirahat, pekerjaan pun diteruskan kembali. Setelah bekerja selama satu jam, langit menjadi mendung dan secara tiba-tiba, hujan deras pun turun. Apa boleh dikata, Anda pun harus kembali menunda pekerjaan memindahkan barang tersebut sekali lagi. Akan tetapi, hujan, sebuah signal lain yang dikirimkan kepada Anda, adalah signal yang otomatis dan tidak dikirimkan secara manual.

Kita mengenal berbagai macam signal sesuai fungsinya. Ada signal yang bertujuan untuk membunuh proses. Ada signal yang dikirimkan oleh kernel karena kerusakan *hardware* tiba-tiba dan lain sebagainya. Dalam signal yang membunuh proses misalnya, kita masih dapat membaginya menjadi beberapa berdasarkan keampuhannya. Ada proses yang membandel dan menolak `SIGTERM` dan `SIGINT`, misalnya. Akan tetapi, tidak berdaya ketika menerima



signal `SIGKILL`. Kembali ke panggilan makan siang pada analogi sebelumnya, tentunya Anda masih dapat menolak panggilan makan pertama. Anda masih dapat juga menolak panggilan kedua, apalagi kalau perut Anda masih belum berdemonstrasi minta makan. Panggilan ketiga pun lewat begitu saja. Akhirnya teman Anda memberikan ultimatum: makan sekarang atau makanan akan dihabiskan. Karena Anda butuh makan, maka mau tidak mau, Anda akan menunda pekerjaan Anda dan menuruti panggilan makan.

Pengiriman signal, selain menggunakan program *kill*, umumnya mungkin sudah sering kita lakukan. Penekanan kombinasi tombol `CTRL-C` dan `CTRL-Z` misalnya. `CTRL-C` akan mengirimkan `SIGINT` (Signal Interrupt), dan penekanan `CTRL-Z` akan mengirimkan `SIGTSTP` (*Signal Terminal Stop*).

Pada saat proses menerima signal, ia bisa melakukan salah satu tindakan berikut ini:

- Mengikuti tindakan default akibat pengiriman signal. Tindakan default adalah tindakan yang telah ditentukan. Sebagai contoh, jika suatu proses dikenakan `SIGTERM` (signal terminasi), maka proses tersebut akan diterminasi karena tindakan default dari `SIGTERM` adalah terminasi proses.
- Mengabaikan signal. Dalam proses-proses yang cukup penting, pengabaian beberapa signal adalah tindakan yang cukup bijak. Hal ini agar proses tidak diterminasi tanpa sengaja oleh penggunaannya. Tidak semua signal dapat diabaikan. `SIGKILL` dan `SIGSTOP` tidak dapat diabaikan.
- Membelokkan signal yang dikirimkan. Sebuah proses yang dikenakan signal tertentu bisa menangkap signal

tersebut, untuk kemudian melakukan tindakan sendiri daripada mengikuti tindakan default signal tersebut. Contohnya adalah penekanan kombinasi tombol CTRL-C pada modus interaktif Python yang tidak mengakibatkan Python diterminasi, melainkan ditangani sendiri oleh Python dengan tindakan sendiri, dalam hal ini membangkitkan eksepsi *KeyboardInterrupt*. Tidak semua signal dapat dibelokkan. SIGKILL dan SIGSTOP tidak dapat dibelokkan.

## Contoh program sederhana

Sebelum kita masuk ke program pengunci terminal, kita akan melihat contoh program sederhana untuk mengirimkan signal, mengikuti tindakan default, mengabaikan serta membelokkan signal yang dikirimkan.

Pengiriman signal dapat dilakukan secara implisit ataupun eksplisit. Secara implisit, kita menggunakan fungsi `alarm()` yang terdapat di dalam modul `signal`.

```
#!/usr/bin/python

import signal

def main():
    signal.alarm(2)
    while 1:
        print 'waiting for alarm...'

if __name__ == '__main__':
    main()
```

Program ini akan mencetak tulisan *waiting for alarm...* terus menerus selama 2 detik, sebelum dihentikan dengan pengiriman `SIGALRM` secara implisit.

Pengiriman signal secara eksplisit dilakukan dengan memanggil fungsi `kill()` yang terdapat di modul `os`.

```
#!/usr/bin/python

import sys
import os
import signal

def main():
    if len(sys.argv) == 3:
        try:
            os.kill(int(sys.argv[2]),
```

```
        signal.__dict__[sys.argv[1]])
    except (KeyError, ValueError,
            OSError), msg:
        sys.exit(msg)
    else:
        sys.exit('usage: %s <signal>
        <pid> ' %sys.argv[0])

if __name__ == '__main__':
    main()
```

Program ini berfungsi seperti program `kill(1)`. Contoh perbedaannya adalah kita harus selalu memasukkan signal yang ingin dikirimkan ke suatu proses. Kita juga tidak dapat menampilkan daftar signal.

Setelah mengirimkan signal, berikutnya kita akan membahas contoh untuk mengikuti tindakan default, mengabaikan, serta membelokkan signal tertentu. Contoh signal yang akan kita abaikan serta belokkan adalah `SIGINT`, yang umumnya dikirimkan ketika pengguna menekan kombinasi tombol CTRL-C.

```
#!/usr/bin/python

import signal
import sys

def registerSignal():
    signal.signal(signal.SIGINT,
                  signal.SIG_DFL)

def main():
    registerSignal()
    n = 0
    while 1:
        n += 1
        print n

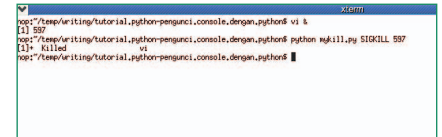
if __name__ == '__main__':
    main()
```

Program ini mengikuti tindakan default signal apabila suatu signal dikirimkan kepadanya. Hal ini adalah tingkah laku program kebanyakan.

```
#!/usr/bin/python

import signal
import sys

def registerSignal():
```



### ▲ Versi lain program `kill(1)`



### ▲ Program pengunci terminal

```
signal.signal(signal.SIGINT,
              signal.SIG_IGN)

def main():
    registerSignal()
    n = 0
    while 1:
        n += 1
        print n

if __name__ == '__main__':
    main()
```

Dengan mengganti `SIG_DFL` dengan `SIG_IGN`, kita dapat mengabaikan `SIGINT` dengan mudah. Untuk keluar dari program tersebut, tekanlah kombinasi tombol CTRL-\. Dan berikutnya, kita akan melihat bagaimana menangani `SIGINT` yang dikirimkan kepada kita.

```
#!/usr/bin/python

import signal
import sys
import time

def signalHandler(signum, frame):
    print 'signal number: %d' %signum
    print dir(frame)
    time.sleep(1)

def registerSignal():
    signal.signal(signal.SIGINT,
                  signalHandler)

def main():
    registerSignal()
    n = 0
    while 1:
        n += 1
        print n
```

```
if __name__ == '__main__':
    main()
```

Program ini akan membelokkan SIGINT yang dikirimkan kepadanya. Alih-alih mengikuti tindakan default SIGINT (proses diterminasi), program ini mencetak nomor signal dan data tambahan, kemudian melakukan penundaan selama 1 detik.

## Program pengunci terminal

Dengan demikian, kita dapat melihat bahwa program untuk mengunci terminal bukanlah program yang sulit untuk dibuat. Prinsip utamanya hanyalah mencoba untuk mengabaikan semua signal. Penggunaan password juga mutlak agar program lebih mudah digunakan.

Sebagai catatan, *screenshot* diambil dari *xterminal*. Hal ini semata-mata dimaksudkan agar screenshot mudah diambil. Efektifnya, program pengunci berjalan di terminal console murni.

Berikut ini adalah *source code* selengkapnya:

```
#!/usr/bin/python

import signal
import sys
import getpass

line = '-' * 80

def registerSignal():
    for i in range(0, signal.SIGINT):
        try:
            signal.signal(i,
                signal.SIG_IGN)
        except:
            pass

def main():
    print 'nopy terminal lock'
    passwd = getpass.getpass('Enter
unlock password: ')
    registerSignal()
    print 'terminal locked'
    print line

    while 1:
        try:
            trypasswd = getpass.
                getpass('Enter password: ')
```

```
        if trypasswd == passwd:
            print 'terminal
            unlocked'
            sys.exit(0)
        else:
            print 'invalid
            password'
        except EOFError:
            continue

if __name__ == '__main__':
    main()
```

signal.SIGINT adalah konstanta yang berisikan nilai maksimal signal pada sistem operasi. Pada saat program dijalankan, maka program akan meminta *password*. Harap diperhatikan untuk tidak menjalankan registerSignal() sebelum pengguna memasukkan password. Dengan demikian, pengguna masih dapat menekan kombinasi tombol CTRL-C untuk membatalkan program pengunci terminal.

Pengunci terminal ini hanya akan keluar dengan terpaksa apabila dikenakan signal SIGKILL (9) dan SIGSTOP (19)

yang memang tidak dapat diabaikan. Untuk mencoba, jalankanlah shell script berikut ini:

```
for i in `seq 1 8`; do kill -s 801; done
for i in `seq 10 18`; do kill -s 801; done
for i in `seq 20 63`; do kill -s 801; done
```

Sebagai catatan, 801 adalah pid dari program pengunci terminal tersebut. Seharusnya, shell script tersebut tidak akan mengakibatkan program pengunci terminal diterminasi.

Aturlah \$PATH agar Anda dapat menjalankan program ini di mana saja. Penambahan tindakan antisipatif agar pengguna tidak salah mengetikkan password tanpa sengaja ketika pertama kali diminta dengan meminta pengguna untuk memasukkan password sekali lagi mungkin harus ditambahkan. Tentunya tidak lucu apabila pengguna terkunci oleh program penguncinya sendiri.

Dengan contoh ini, kita melihat bahwa pemrograman Linux menggunakan Python sangatlah menyenangkan untuk dilakukan. Selamat mencoba!

**Noprianto** ([noprianto@infolinux.co.id](mailto:noprianto@infolinux.co.id))



# Python dan File Sistem

Sebagai bahasa pemrograman *general purpose*, Python dapat pula digunakan untuk bermain-main dengan file sistem Linux.

Linux adalah sistem operasi yang menganggap hampir segala sesuatu adalah file. Sederhananya, harddisk Anda hanyalah sebuah file bagi Linux. Dalam hidup sehari-hari pun, kita tidak dapat melepaskan diri dari file. Baik itu mengetik dokumen, mendengarkan lagu, sampai proses *boot* Linux.

Karena itulah, kita mengenal sangat banyak utiliti yang berhubungan dengan file. Mulai dari mendaftar file di suatu direktori, mengubah kepemilikan suatu file, sampai mengetahui tipe file. Apabila Anda seorang *developer* Python, Anda pun dapat membuat sendiri utiliti tersebut. Berikut ini kita akan membahas tentang seputar file di Linux dan implementasinya dengan Python.

## File di Linux

Di Linux, kita mengenal bermacam-macam tipe file. Untuk bermain-main dengan file sistem, ada baiknya bagi kita untuk membedakan bagaimana Linux memandang suatu file. Berikut ini adalah kategorisasi file di Linux:

### → file biasa

File ini adalah file yang paling umum dan mudah ditemukan di mana-mana. File ini pula yang seringkali dimaksud apabila kita membicarakan istilah file. File mp3, video klip, dokumen yang kita simpan bahkan kernel termasuk file tipe ini. File tipe ini adalah file yang berorientasi *byte*.

### → Direktori

Direktori juga lazim ditemukan di mana-mana. Sederhananya, direktori adalah file yang mengandung file lain. Bekerja dengan direktori di Python sangatlah mudah. Peng-*copy*-an satu direktori beserta seluruh isi dan strukturnya dapat dilakukan dengan satu perintah saja. Apabila Anda melakukan perintah `ls -al`, karakter paling kiri dari hak akses akan bernilai `d`.

### → File spesial blok

File yang satu ini pengoperasiannya dilakukan dalam satuan blok. Yang termasuk file ini di antaranya harddisk, cdrom, dan floppy. Apabila Anda melakukan perintah `ls -al`, karakter paling kiri dari hak akses akan bernilai `b`.

### → File spesial karakter

File yang satu ini pengoperasiannya dilakukan dalam satuan karakter. Sederhananya, transfer data dilakukan per karakter. Yang termasuk file ini di antaranya terminal, mouse, dan printer. Apabila Anda melakukan perintah `ls -al`, karakter paling kiri dari hak akses akan bernilai `c`.

### → Symbolic link

Dalam dunia Windows, file yang satu ini lebih populer dengan istilah *shortcut*. *Symbolic link* adalah file yang menunjuk ke file lain. Ukurannya hanyalah sebesar alamat file yang ditunjuknya. *Hardlink* tidak termasuk file yang satu ini. Apabila Anda melakukan perintah `ls -al`, karakter paling kiri dari hak akses akan bernilai `l`.

### → Pipe

Sederhananya, file ini bisa diibaratkan sebagai pipa, di mana mengambil *output* suatu proses menjadi *input* dan mengeluarkannya kembali untuk menjadi input proses lainnya. Analogi dalam kehidupan sehari-hari adalah pipa yang disambungkan ke kran air. Pipa mengambil input dari keluaran kran dan melewati air ke tempat lainnya. File tipe ini dibagi lagi menjadi dua, pipa tak bernama (*unnamed pipe*) dan pipa bernama (*named pipe*). Pipa bernama disebut juga *fifo*. Apabila Anda melakukan perintah `ls -al`, karakter paling kiri dari hak akses suatu pipa bernama (*fifo*) akan bernilai `p`.

Berikut ini adalah contoh penggunaan pipa tak bernama yang mungkin sudah sangat sering dilakukan:

```
cat a | sort
```



Keluaran dari program `cat a` akan diambil sebagai input oleh pipa tak bernama, kemudian menyalurkannya menjadi input bagi program `sort`. Proses ini terjadi di dalam struktur data kernel, dan karena kita tidak dapat menemukannya file sistem, maka pipa ini disebut sebagai pipa tak bernama. Berbeda halnya dengan pipa bernama yang kehadirannya di file sistem dapat ditemukan. Untuk membuat *fifo*, kita dapat menggunakan program `mkfifo`.

### → socket

File ini memiliki fungsi kurang lebih sama seperti pipe. Perbedaannya adalah *socket* lebih diarahkan untuk koneksi jaringan, baik dalam satu *host* ataupun dalam jaringan fisik. Apabila Anda melakukan perintah `ls -al`, karakter paling kiri dari hak akses akan bernilai `s`.

## Mendapatkan tipe file

Untuk mendapatkan tipe suatu file di Linux, cara selain melihat keluaran dari program `ls` adalah menggunakan program `file`. Program `file` dapat secara akurat menentukan tipe dan informasi ekstra dari suatu file.

Bagaimana caranya mendapatkan tipe suatu file menggunakan Python? Berikut ini adalah langkah-langkahnya:

- Gunakan `os.stat()` untuk mendapatkan informasi file. `os.stat()` akan mengembalikan suatu *tuple*, di mana untuk saat ini, `os.stat()[0]` yang akan kita perhatikan. `os.stat()[0]` adalah mode (*inode protection bits*) dari suatu file
- gunakan modul `stat` yang berfungsi untuk menindaklanjuti hasil keluaran dari `os.stat()`. Anda juga bisa melakukan operasi AND antara mode file dengan konstanta di modul `stat`. Anda bisa juga menggunakan fungsi-fungsi di module `stat`.



```
root@temp/writing/tutorial_python-python_da_files#stat python FileModel.py
python FileModel.py is a regular file with ('user has r/w/', 'user can read', 'user can write', 'group has r/w/', 'group can read', 'others have r/w/', 'others can read')
```

◀ Mendapatkan hak akses dan modifier

```
root@temp/writing/tutorial_python-python_da_files#stat python FileModel.py
stat: 0x00000000
  inode: 53288
  device: 770
  number of hardlinks: 1
  owner's uid = 1000
  owner's gid = 100
  size in bytes = 679
  time of last access (atime) = Mon Jun 23 16:10:26 2003
  time of last modification (mtime) = Mon Jun 23 16:10:10 2003
  time of last change (ctime) = Mon Jun 23 16:10:10 2003
  root@temp/writing/tutorial_python-python_da_files#
```

### Mendapatkan informasi internal file

Berikut ini adalah beberapa fungsi untuk mendapatkan tipe file yang terdapat dalam modul stat:

- **S\_ISDIR(mode)**

Berguna untuk memeriksa apakah suatu file bertipe direktori.

- **S\_ISCHR(mode)**

Berguna untuk memeriksa apakah suatu file bertipe file spesial karakter.

- **S\_ISBLK(mode)**

Berguna untuk memeriksa apakah suatu file bertipe file spesial blok.

- **S\_ISREG (mode)**

Berguna untuk memeriksa apakah suatu file bertipe file biasa.

- **S\_ISLNK(mode)**

Berguna untuk memeriksa apakah suatu file bertipe *symbolic link*.

- **S\_ISFIFO(mode)**

Berguna untuk memeriksa apakah suatu file bertipe *fifo*.

- **S SOCK(mode)**

Berguna untuk memeriksa apakah suatu file bertipe *socket*.

Contoh:

```
>>> import os
>>> from stat import *
>>> a = os.stat('/bin/lis')
>>> ms = a[0]
>>> S_ISDIR(ms)
0
>>> S_ISREG(ms)
1
>>> S_ISCHR(ms)
```

```
0
>>> S_ISBLK(ms)
0
>>> S_ISLNK(ms)
0
>>> S_ISFIFO(ms)
0
>>> S_ISSOCK(ms)
0
```

### Mendapatkan informasi internal file

Suatu file tidak hanya memiliki informasi berupa tipe file. Masih banyak informasi internal dari suatu file yang bisa didapatkan. Sebelumnya, kita telah menggunakan fungsi `stat()` dari modul `os`. Saat ini, kita akan membahas lebih jauh keluaran dari `os.stat()`.

Berikut ini adalah tuple yang dikembalikan oleh `os.stat()`:

```
os.stat(path) -> (mode, ino, dev, nlink, uid,
gid, size, atime, mtime, ctime)
```

Penjelasan masing-masing anggota tuple yang dikembalikan oleh `os.stat()`

- **mode** adalah informasi mode file.

Secara teknis diartikan sebagai inode protection bits. Mode file yang didapatkan ini dapat digunakan untuk mendapatkan access bit, modifier dan tipe file. Untuk selanjutnya, mode akan banyak dipakai.

- **Ino** adalah inode suatu file. Di Linux, setiap file memiliki inode yang dapat

diartikan sebagai identitas file. Apabila seorang mahasiswa memiliki nomor induk mahasiswa, maka inode adalah nomor induk file di file sistem.

- **Dev** akan memberikan informasi *device*.
- **nlink** akan memberikan informasi mengenai jumlah hardlink yang menunjuk pada file tersebut. Harap diingat, jumlah hardlink dan bukannya symbolic link yang dicatat di sini.
- **Uid** akan memberikan informasi uid pemilik file.
- **gid** akan memberikan informasi gid pemilik file.
- **size** akan mengembalikan informasi ukuran suatu file dalam *long integer*.
- **atime** akan memberikan informasi berupa waktu terakhir akses suatu file. Waktu akses adalah waktu di mana kita melakukan akses pada suatu file, seperti membuka file untuk diedit dan waktu saat dijalankan untuk file *executable*.
- **Ctime** akan memberikan informasi berupa waktu terakhir perubahan suatu file. Waktu perubahan adalah waktu di mana kita melakukan perubahan pada file, seperti mengubah hak akses file. Jika Anda melakukan perubahan pada isi file, maka secara otomatis, waktu perubahan ini juga ikut berubah.
- **Mtime** akan memberikan informasi berupa waktu terakhir modifikasi suatu file. Waktu modifikasi adalah waktu di mana kita melakukan perubahan pada tubuh file, seperti mengubah isi file.

Contoh:

```
>>> a = os.stat('/bin/lis')
>>> for i in a:
...     print i
...
33261
9
770
1
0
0
43784
1056357504
1016464201
1053454562
```

Contoh program: fileinfo1.py

```
#!/usr/bin/python

import os
import sys
import time

def main():
    if len(sys.argv) == 2:
        try:
            a = os.stat(sys.argv[1])
        except OSError, msg:
            sys.exit(msg)
        lstats = \
            ('mode (protection bits)', 'inode', 'device', \
             'number of hardlinks', 'owner's uid', 'owner's gid', \
             'size in bytes', 'time of last access (atime)', \
             'time of last modifications (mtime)', \
             'time of last change (ctime)')
        for i in range(0, len(lstats) - 3):
            print '%s = %s' % (lstats[i], a[i])
        for i in range(len(lstats) - 3, len(lstats)):
            print '%s = %s' % (lstats[i], time.asctime(time.localtime(a[i])))
        else:
            sys.exit('nothing to do')

if __name__ == '__main__':
    main()
```

Contoh ini menggunakan lstat(), yang fungsinya serupa dengan os.stat()

## Mendapatkan hak akses dan modifier

Masih menggunakan informasi yang di dapat dari os.stat(), kita akan mendapatkan hak akses dan modifier suatu file, seperti sticky bit, SUID, dan SGID. Berikut adalah langkah-langkah untuk mendapatkan hak akses dan modifier:

- Mengambil hasil keluaran dari os.stat()[0].
- Gunakan module stat untuk menindaklanjuti keluaran os.stat().

- Lakukan operasi *bitwise* AND antara mode file yang didapat dengan beberapa konstanta dalam modul stat, di mana jika hasil operasi AND tersebut mengembalikan nilai bukan nol, maka file akan memiliki karakteristik-karakteristik sesuai dengan konstanta-konstanta tersebut.

Contoh:

```
>>> import os
>>> from stat import *
>>> a = os.stat('/bin/lis')
>>> ma = a[0]
>>> ma & S_ISUID
0
>>> ma & S_ISGID
0
>>> ma & S_ISVTX
0
>>> ma & S_IRWXU
448
>>> ma & S_IRUSR
256
>>> ma & S_IWUSR
128
>>> ma & S_IXUSR
64
>>> ma & S_IRWXG
40
>>> ma & S_IRGRP
32
>>> ma & S_IWGRP
0
>>> ma & S_IXGRP
8
```

```
>>> ma & S_IRWXO
5
>>> ma & S_IROTH
4
>>> ma & S_IWOTH
0
>>> ma & S_IXOTH
1
>>>
```

Contoh program: filemode1.py

```
#!/usr/bin/python

import os
import sys
from stat import *

def main():
    if len(sys.argv) != 2:
        sys.exit('nothing to do')
    else:
        try:
            ma = os.stat(
                sys.argv[1])[0]
        except OSError, msg:
            sys.exit(msg)

        tType = (\
            (S_ISREG, 'regular file'), \
            (S_ISDIR, 'directory'), \
            (S_ISBLK, 'block special file'), \
            (S_ISCHR, 'character special file'), \
            (S_ISLNK, 'symbolic link'), \
```

Berikut ini adalah daftar konstanta untuk hak akses dan modifier:

Konstanta	Nilai	Arti
S_ISUID	2048	SUID
S_ISGID	1024	SGID
S_ISVTX	512	Sticky bit
S_IRWXU	448	user memiliki r/w/x
S_IRUSR	256	user memiliki hak baca ( r )
S_IWUSR	128	user memiliki hak tulis ( w )
S_IXUSR	64	user memiliki hak execute ( x )
S_IRWXG	56	group memiliki r/w/x
S_IRGRP	32	group memiliki hak baca ( r )
S_IWGRP	16	group memiliki hak tulis ( w )
S_IXGRP	8	group memiliki hak execute ( x )
S_IRWXO	7	orang lain memiliki r/w/x
S_IROTH	4	orang lain memiliki hak baca ( r )
S_IWOTH	2	orang lain memiliki hak tulis ( w )
S_IXOTH	1	orang lain memiliki hak execute ( x )

```

(S_ISFIFO,'FIFO'),\
(S_ISSOCK,'socket'))

tMode = (\
(S_ISUID,'SUID'),\
(S_ISGID,'SGID'),\
(S_ISVTX,'sticky'),\
(S_IRWXU,'user has r/w/x'),\
(S_IRUSR,'user can read'),\
(S_IWUSR,'user can write'),\
(S_IXUSR,'user can execute'),\
(S_IRWXG,'group has r/w/x'),\
(S_IRGRP,'group can read'),\
(S_IWGRP,'group can read'),\
(S_IXGRP,'group can
execute'),\
(S_IRWXO,'others have r/
w/x'),\
(S_IROTH,'others can read'),\
(S_IWOTH,'others can write'),\
(S_IXOTH,'others can
execute'))

for i in tType:
    if i[0](ma):
        fileType = i[1]
        break

IModeRes = []
for i in tMode:
    if ma & i[0] != 0:
        IModeRes.append(i[1])

print '%s is a %s with %s'\
      %(sys.argv[1], fileType,
IModeRes)

if __name__ == '__main__':
    main()

```

## Mengubah kepemilikan file (chown)

Sebagai pengguna Linux, sebuah sistem operasi *multiuser*, tentunya kita sering bermain-main dengan `chown(1)`. Program tersebut, sesuai namanya, bertujuan untuk mengubah kepemilikan file.

Python sendiri menyediakan fungsi untuk mengubah kepemilikan suatu file. Berikut ini adalah prototipe fungsinya:

```
chown (path, uid, gid) -> none
```

`path` adalah nama file yang ingin diubah kepemilikannya, sementara `uid` adalah uid

pemilik yang baru, dan `gid` adalah gid yang baru.

Dengan modal fungsi tersebut, kita akan membuat klon dari program `chown`. Sedikit sentuhan perlu ditambahkan, agar pengguna dapat memasukkan nama user atau group, dan bukannya uid atau gid (dalam format bilangan).

Contoh program: `chown1.py`

```

#!/usr/bin/python

import os
import sys
import pwd
import grp

def main():
    if len(sys.argv) != 3:
        sys.exit('nothing to do')
    else:
        try:
            uid, gid =
os.stat(sys.argv[2])[4:6]
            owner = list(sys.argv[1])
            if owner.count(':') == 0:
                newUid =

```

```


pwd.getpwnam(sys.argv[1])[2]
                newGid = gid
            else:
                newUid, newGid =
sys.argv[1].split(':')[0:]
                newUid =
pwd.getpwnam(newUid)[2]
                newGid =
grp.getgrnam(newGid)[2]

            os.chown(sys.argv[2],
newUid, newGid)

        except (KeyError, OSError,
ValueError), msg:
            sys.exit(msg)

if __name__ == '__main__':
    main()

```

Demikianlah beberapa contoh untuk mengutak-atik file sistem dengan Python. Anda dapat mengembangkannya lebih jauh lagi. Barangkali Anda ingin membuat saingan Nautilus, Konqueror, atau Windows Explorer?  **Noprianto** ([noprianto@infolinux.co.id](mailto:noprianto@infolinux.co.id))

# MORE... SPACE

# RELIABILITY & TIME

# LESS... & MONEY

**LINUX, FreeBSD, and Win 2000 Hosting**

Features :

- Unlimited data transfer
- Complete control panels
- POP3 email, FTP access
- SSH, CGI, SQL...
- and much more...
- Start from Rp. 19.500,-/ month
- Free Setup \*)
- 2 Months Free \*\*)

**Server Hosting**

Features :

- Location NOC Jakarta - Indonesia (IIX)
- Size server : 1 U Rackmount
- Bandwidth : 128 kbps
- IP Address : 8 (max)
- Colocation : Rp. 1.000.000,-/ month

**ALSO**

- Colocation & Dedicated Server in USA
- Domain Name Register
- Benefit Reseller Program

**Limited Offer :**  
Dedicated Server  
Rp. 1.250.000,-/ mo

**"IT'S NEVER BEEN EASIER  
TO TAKE YOUR BUSINESS ONLINE"**

Note : \*) Transfer (restriction apply)  
\*\*) 1 year payment

**CAKRAWEB**  
Supporting You to a Web Success

Cyber Building (d/h Elektrindo) 10 th Floor  
Jl. Kuningan Barat No. 8 Jakarta Selatan 12710  
Phone. (021) 526 8000 Fax. (021) 52 66 444  
<http://www.cakraweb.com> - [info@cakraweb.com](mailto:info@cakraweb.com)