

# **Praktikum Sistem Operasi**

Auriza Akbar

2017



---

## DAFTAR ISI

---

|   |  |    |
|---|--|----|
| i | PEMROGRAMAN SISTEM UNIX                      | 1  |
| 1 | PROSES                                       | 3  |
|   | Intro  | 3  |
|   | Tim Praktikum                                | 3  |
|   | Peraturan                                    | 3  |
|   | LMS  | 4  |
|   | Proses                                       | 4  |
|   | Apa itu proses?                              | 4  |
|   | Apa itu proses?                              | 4  |
|   | Bagaimana cara menjalankan program?          | 4  |
|   | Bagaimana cara menjalankan program?          | 4  |
|   | Shell  | 5  |
|   | Apa itu <i>shell</i> ?                       | 5  |
|   | Apa itu <i>shell</i> ?                       | 5  |
|   | <i>Kernel</i> vs <i>Shell</i>                | 5  |
|   | <i>Kernel</i> vs <i>Shell</i>                | 5  |
|   | <i>Kernel</i> vs <i>Shell</i>                | 5  |
|   | Contoh <i>kernel</i>                         | 5  |
|   | Contoh <i>shell</i>                          | 6  |
|   | Bagaimana <i>shell</i> bisa membuat proses?  | 6  |
|   | <code>system()</code>                        | 6  |
|   | Fungsi <code>system()</code>                 | 6  |
|   | Contoh                                       | 6  |
|   | Hierarki proses                              | 7  |
|   | Latihan                                      | 7  |
|   | <code>exec()</code>                          | 7  |
|   | Fungsi <code>exec()</code>                   | 7  |
|   | Contoh <code>execlp()</code>                 | 7  |
|   | Contoh <code>execvp()</code>                 | 8  |
|   | Hierarki proses                              | 8  |
|   | Latihan                                      | 8  |
|   | <code>system()</code> vs <code>exec()</code> | 9  |
| 2 | FORK, WAIT                                   | 11 |
|   | <code>fork()</code>                          | 11 |
|   | Fungsi <code>fork()</code>                   | 11 |
|   | Contoh                                       | 11 |
|   | Latihan                                      | 11 |
|   | <i>Parent</i> atau <i>Child</i> ?            | 12 |
|   | Konkurensi                                   | 12 |
|   | Latihan                                      | 13 |

|   |  |    |
|---|--|----|
|   | <code>wait()</code>                    | 13 |
|   | Fungsi <code>wait()</code>             | 13 |
|   | Contoh                                 | 13 |
|   | Mendapatkan Status Proses <i>Child</i> | 14 |
|   | Zombie                                 | 14 |
|   | Proses Zombie                          | 14 |
|   | Contoh                                 | 14 |
|   | Mengamati Zombie                       | 15 |
|   | Latihan                                | 15 |
|   | Latihan Buku                           | 15 |
|   | Latihan Tambahan                       | 16 |
| 3 | SIGNAL                                 | 17 |
|   | Sinyal                                 | 17 |
|   | Jenis sinyal                           | 17 |
|   | <code>signal()</code>                  | 20 |
|   | Fungsi <code>signal()</code>           | 20 |
|   | Contoh                                 | 20 |
|   | Penjelasan                             | 20 |
|   | <code>kill()</code>                    | 20 |
|   | Fungsi <code>kill()</code>             | 20 |
|   | Contoh                                 | 21 |
|   | Penjelasan                             | 21 |
|   | <code>pause()</code>                   | 21 |
|   | Fungsi <code>pause()</code>            | 21 |
|   | Contoh                                 | 21 |
|   | Penjelasan                             | 22 |
|   | Tugas                                  | 22 |
|   | Tugas Bonus                            | 22 |
| 4 | UNIX SHELL                             | 25 |
|   | Shell                                  | 25 |
|   | Cara Kerja Shell                       | 25 |
|   | Membuat Proses Baru                    | 25 |
|   | Tugas                                  | 25 |
|   | UNIX Shell                             | 25 |
|   | History Feature                        | 26 |
| 5 | THREAD                                 | 27 |
|   | Thread                                 | 27 |
|   | Thread                                 | 27 |
|   | POSIX Thread                           | 27 |
|   | Membuat Thread                         | 27 |
|   | Menunggu Thread                        | 28 |
|   | Mengakhiri Thread                      | 28 |
|   | Contoh                                 | 28 |
|   | Satu Thread Tanpa Argumen              | 28 |
|   | Dua Thread Tanpa Argumen               | 28 |
|   | Banyak Thread Tanpa Argumen            | 29 |

|    |                              |    |
|----|------------------------------|----|
|    | Satu Thread Dengan Argumen   | 29 |
|    | Dua Thread Dengan Argumen    | 30 |
|    | Banyak Thread Dengan Argumen | 30 |
|    | Latihan                      | 30 |
|    | Jumlah Array                 | 30 |
|    | Jumlah Array (Satu Thread)   | 31 |
|    | Jumlah Array (Dua Thread)    | 31 |
|    | Jumlah Array (Empat Thread)  | 31 |
| 6  | SINKRONISASI THREAD          | 33 |
|    | Critical Section             | 33 |
|    | Critical Section             | 33 |
|    | Sinkronisasi                 | 33 |
|    | Mutual Exclusion             | 33 |
|    | Mutex                        | 33 |
|    | Fungsi Mutex                 | 34 |
|    | Latihan                      | 34 |
|    | Semaphore                    | 35 |
|    | Semaphore                    | 35 |
|    | Jenis Semaphore              | 36 |
|    | Fungsi Semaphore             | 36 |
|    | Latihan                      | 36 |
|    | Tugas                        | 36 |
|    | Array Sum                    | 36 |
| ii | ADMINISTRASI SISTEM UNIX     | 39 |
| 7  | UNIX INTRO                   | 41 |
|    | Pendahuluan                  | 41 |
|    | Mengapa CLI?                 | 41 |
|    | Server SO                    | 41 |
|    | Login Server                 | 41 |
|    | Sumber Tambahan              | 41 |
|    | Perintah Dasar               | 42 |
|    | ssh                          | 42 |
|    | echo                         | 42 |
|    | hostname                     | 43 |
|    | uname                        | 43 |
|    | uptime                       | 43 |
|    | date                         | 43 |
|    | cal                          | 43 |
|    | whoami                       | 43 |
|    | who                          | 44 |
|    | w                            | 44 |
|    | last                         | 44 |
|    | write                        | 44 |
|    | mesg                         | 44 |
|    | mail                         | 44 |

## Daftar Isi

|    |                                     |    |
|----|-------------------------------------|----|
|    | passwd                              | 44 |
|    | logout                              | 45 |
|    | poweroff                            | 45 |
|    | reboot                              | 45 |
|    | Tombol <i>shortcut</i>              | 45 |
|    | Tugas                               | 46 |
| 8  | FILE DAN DIREKTORI                  | 47 |
|    | File dan Direktori                  | 47 |
|    | pwd                                 | 47 |
|    | cd                                  | 47 |
|    | ls                                  | 47 |
|    | touch                               | 48 |
|    | mkdir                               | 48 |
|    | cp                                  | 48 |
|    | mv                                  | 48 |
|    | rm                                  | 48 |
|    | rmdir                               | 49 |
|    | <i>Dotfile</i>                      | 49 |
|    | Simbol                              | 49 |
|    | <i>Path</i>                         | 49 |
|    | Tugas                               | 50 |
| 9  | PERMISSION, LINK, PIPE, REDIRECTION | 51 |
|    | <i>Ownership</i>                    | 51 |
|    | <i>Permission</i>                   | 51 |
|    | su                                  | 51 |
|    | chown                               | 52 |
|    | chmod                               | 52 |
|    | Format chmod                        | 52 |
|    | Contoh chmod                        | 54 |
|    | Link                                | 54 |
|    | ln                                  | 54 |
|    | <i>Stream</i> Standar               | 54 |
|    | <i>Pipe</i>                         | 56 |
|    | <i>Redirect</i>                     | 57 |
| 10 | PENCARIAN DAN PEMROSESAN TEKS       | 59 |
|    | Pencarian                           | 59 |
|    | man                                 | 59 |
|    | which                               | 59 |
|    | locate                              | 60 |
|    | find                                | 60 |
|    | xargs                               | 60 |
|    | grep                                | 60 |
|    | Pemrosesan Teks                     | 61 |
|    | editor (nano)                       | 61 |
|    | vi                                  | 61 |
|    | pager (less)                        | 61 |

|                             |    |  |
|-----------------------------|----|--|
| cat                         | 61 |  |
| split                       | 63 |  |
| sort                        | 63 |  |
| uniq                        | 63 |  |
| head                        | 63 |  |
| tail                        | 63 |  |
| tr                          | 64 |  |
| sed                         | 64 |  |
| cut                         | 64 |  |
| paste                       | 64 |  |
| Ekspresi Reguler            | 64 |  |
| Ekspresi Reguler (Regex)    | 64 |  |
| Referensi Singkat           | 65 |  |
| Referensi Singkat           | 65 |  |
| Tugas                       | 65 |  |
| 11 PROCESS AND JOB CONTROL  | 67 |  |
| Proses                      | 67 |  |
| ps                          | 67 |  |
| pstree                      | 67 |  |
| top                         | 67 |  |
| htop                        | 67 |  |
| pgrep                       | 68 |  |
| kill                        | 68 |  |
| pmap                        | 68 |  |
| lsof                        | 68 |  |
| nice                        | 68 |  |
| renice                      | 70 |  |
| Job Control                 | 70 |  |
| Process State               | 70 |  |
| Background Process          | 70 |  |
| jobs                        | 71 |  |
| fg                          | 71 |  |
| bg                          | 71 |  |
| 12 SHELL SCRIPTING          | 73 |  |
| <i>Shell Scripting</i>      | 73 |  |
| Contoh                      | 73 |  |
| Referensi                   | 73 |  |
| Ekspansi                    | 74 |  |
| <i>Pathname Expansion</i>   | 74 |  |
| <i>Pathname Expansion</i>   | 74 |  |
| <i>Parameter Expansion</i>  | 74 |  |
| <i>Command Substitution</i> | 74 |  |
| <i>Arithmetic Expansion</i> | 75 |  |
| Parameter Khusus            | 75 |  |
| Ekspresi                    | 75 |  |
| [                           | 75 |  |

|                                |    |
|--------------------------------|----|
| [                              | 76 |
| seq                            | 76 |
| read                           | 76 |
| Kontrol Aliran                 | 77 |
| Percabangan                    | 77 |
| Kasus                          | 77 |
| Perulangan                     | 77 |
| Perulangan                     | 77 |
| Fungsi                         | 78 |
| Contoh                         | 78 |
| Percabangan                    | 78 |
| Kasus                          | 78 |
| Perulangan                     | 78 |
| Fungsi                         | 79 |
| Perulangan dan Percabangan     | 79 |
| Latihan                        | 79 |
| Frekuensi Kata Terbanyak       | 79 |
| Identifikasi Penyerang         | 80 |
| Cek Tugas Email                | 80 |
| Tugas: <i>Spelling Checker</i> | 81 |
| 13 PERINTAH TAMBAHAN           | 83 |
| Kompresi                       | 83 |
| tar                            | 83 |
| gzip                           | 83 |
| gunzip                         | 83 |
| tar.gz                         | 84 |
| tar.gz                         | 84 |
| zip                            | 84 |
| unzip                          | 84 |
| pgp                            | 85 |
| Konversi                       | 85 |
| convert                        | 85 |
| avconv                         | 86 |
| pandoc                         | 86 |
| tesseract                      | 87 |
| espeak                         | 87 |
| dot                            | 87 |
| figlet                         | 88 |
| Info Sistem                    | 88 |
| lscpu                          | 88 |
| lshw                           | 89 |
| lspci                          | 89 |
| lsusb                          | 89 |
| lsblk                          | 89 |
| lslocks                        | 89 |
| lsof                           | 90 |



|        |    |
|--------|----|
| df     | 90 |
| du     | 90 |
| free   | 90 |
| vmstat | 90 |
| bmon   | 91 |



---

## DAFTAR GAMBAR

---

|           |                                       |    |
|-----------|---------------------------------------|----|
| Gambar 1  | Tata tertib mahasiswa IPB             | 3  |
| Gambar 2  | Komunikasi antarproses pada Linux     | 18 |
| Gambar 3  | Sinyal pada UNIX                      | 19 |
| Gambar 4  | <i>Don't share mutable state</i>      | 34 |
| Gambar 5  | Raspberry Pi                          | 42 |
| Gambar 6  | UNIX permissions                      | 52 |
| Gambar 7  | Membuka akses untuk publik            | 53 |
| Gambar 8  | Inode                                 | 55 |
| Gambar 9  | Direktori dan symlink                 | 55 |
| Gambar 10 | File descriptor                       | 56 |
| Gambar 11 | Pipe                                  | 57 |
| Gambar 12 | Bagian halaman manual                 | 59 |
| Gambar 13 | Petunjuk singkat vi                   | 62 |
| Gambar 14 | <i>Real programmers</i>               | 62 |
| Gambar 15 | <i>Regex saves the day</i>            | 66 |
| Gambar 16 | Don't SIGKILL                         | 69 |
| Gambar 17 | <i>Process information filesystem</i> | 70 |
| Gambar 18 | <i>Process state</i>                  | 71 |



Bagian I

PEMROGRAMAN SISTEM UNIX



## PROSES

### INTRO

#### *Tim Praktikum*

- Auriza Rahmad Akbar
- M Mukhibillah Asshidiqy
- Kurnia Saputra
- Lu William Hanugra
- Selfi Qisthina

#### *Peraturan*

- Pakaian sopan, tidak ketat
  - pelanggaran lebih dari 3 kali: sanksi sedang (nilai 0)
- Kehadiran minimal 80%
- Toleransi terlambat 15 menit
- Tidak membawa makanan ke lab



Gambar 1: Tata tertib mahasiswa IPB

## PROSES

### *LMS*

- <https://lms.ipb.ac.id/course/view.php?id=154>
  - *key*: so2017
- Buku acuan:
  - Silberschatz *et al.* 2013. *Operating System Concepts*. Ed ke-9.
- Proporsi nilai praktikum:
  - UTSP: 30%
  - UASP: 30%
  - Tugas: 40%

## PROSES

*Apa itu proses?*

*Apa itu proses?*

Program yang sedang berjalan.

A program in execution.<sup>1</sup>

*Bagaimana cara menjalankan program?*

Misalkan kita ingin menjalankan program Firefox. Ada berapa cara?

*Bagaimana cara menjalankan program?*

Misalkan kita ingin menjalankan program Firefox. Ada berapa cara?

Dua cara:

1. **CLI**: buka *shell*, lalu ketikkan perintah **firefox**.
2. **GUI**: klik ikon Firefox pada menu aplikasi<sup>2</sup>.

---

<sup>1</sup> Silberschatz *et al.* (2013), *Operating System Concepts*, hlm 105.

<sup>2</sup> jika ikon diklik, program akan tetap dijalankan melalui *shell*; coba cek isi *file* `/usr/share/applications/firefox.desktop`.



## SHELL

*Apa itu shell?*

*Apa itu shell?*

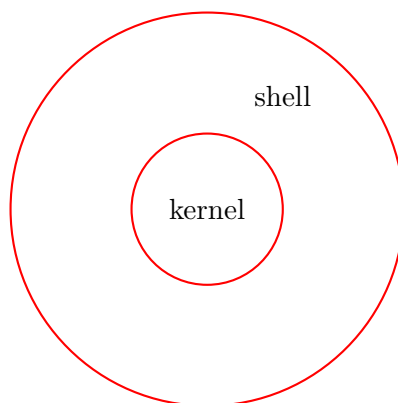
- *Shell* adalah antarmuka antara pengguna dengan *kernel*.

Kernel *vs* Shell

- *kernel* = ...
- *shell* = ...

Kernel *vs* Shell

- *kernel* = inti
- *shell* = kulit

Kernel *vs* Shell

- *Kernel* adalah inti dari sistem operasi.
- *Shell* adalah antarmuka antara pengguna dengan *kernel*.
- *Shell* bertugas untuk menjalankan aplikasi pengguna.
  - *user* → *shell* → *kernel*.

*Contoh* kernel

- UNIX
  - BSD

## PROSES

- AIX
- HP-UX
- Solaris
- Linux
- Windows NT

### *Contoh shell*

- Bourne shell (**sh**)
- Bourne-again shell (**bash**)
- Korn shell (**ksh**)
- Z shell (**zsh**)
- Windows PowerShell

### *Bagaimana shell bisa membuat proses?*

**Tugas:** baca Silberschatz *et al.* (2013), hlm 116–118 sebagai tugas sekaligus materi praktikum pekan depan.

## SYSTEM()

### *Fungsi system()*

```
int system(char *command);
```

- Menjalankan `command` dengan menjalankan *shell* terlebih dahulu<sup>3</sup>:
  - `sh -c "command"`

### *Contoh*

- Menjalankan perintah “`ps --forest`”.

```
// system.c
int main()
{
    puts("Running command");

    system("ps --forest");
}
```

---

<sup>3</sup> lihat ‘`man system`’

```

    puts("Done");
    return 0;
}

```

### *Hierarki proses*

```

..
\_ bash
  \_ ./system
    \_ sh
      \_ ps

```

- Bisa menjalankan rangkaian beberapa perintah sekaligus.
- Contoh:

```

int main()
{
    system("hostname | rev");
    return 0;
}

```

### *Latihan*

- Buat program untuk menjalankan perintah ‘ps -A’!
- Buat program untuk mencetak kalender bulan Desember!

## EXEC()

### *Fungsi exec()*

```

int execlp(char *file, char *arg, ...);
int execlp(char *file, char *arg, ...);

```

- Menggantikan proses yang ada dengan proses baru<sup>4</sup>

### *Contoh execlp()*

- Parameter perintah ditempatkan pada *list* argumen.
- Menjalankan perintah “ps --forest”:

---

<sup>4</sup> lihat ‘man exec’

```
// exec.c
int main()
{
    puts("Running command");

    execlp("ps", "ps", "--forest", NULL);

    puts("Done");
    return 0;
}
```

#### Contoh *execvp()*

- Parameter perintah disimpan pada variabel *string array*.
- Menjalankan perintah “ps --forest”:

```
// exec.c
int main()
{
    puts("Running command");

    char *args[] = {"ps", "--forest", NULL};
    execvp(args[0], args);

    puts("Done");
    return 0;
}
```

#### Hierarki proses

```
..
\_ bash
  \_ ./exec
```

Setelah pemanggilan fungsi `exec`, proses lama akan ditimpa.

```
..
\_ bash
  \_ ps --forest
```

#### Latihan

- Buat program untuk menjalankan perintah ‘ps -A’!

- Buat program untuk mencetak kalender bulan Juni!

*system() vs exec()*

- Fungsi `system()` lebih mudah digunakan
  - namun tidak efisien dalam penggunaan memori dan waktu
  - karena harus membuat dua proses baru untuk tiap perintah
- Fungsi `exec()` lebih efisien
  - langsung menimpa proses yang sudah ada
  - dipakai oleh *shell* untuk membuat proses baru



---

## FORK, WAIT

---

`FORK()`

*Fungsi `fork()`*

`pid_t fork(void);`

- Menduplikasi proses<sup>1</sup>
- *Return value:*
  - jika *parent*: PID *child*
  - jika *child*: 0
  - jika *error*: -1

*Contoh*

```
// fork.c
#include <stdio.h>
#include <unistd.h>

int main()
{
    fork();
    puts("hello");
    return 0;
}
```

*Latihan*

```
#include <stdio.h>
#include <unistd.h>
```

---

<sup>1</sup> lihat 'man 2 fork'

FORK, WAIT

```
int main()
{
    fork();
    fork();
    puts("X");
    return 0;
}
```

Berapa kali X dicetak?

Parent *atau* Child?

*Return value fork()* digunakan untuk membedakan antara proses *parent* dengan *child*.

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

int main()
{
    pid_t pid;                // process id

    pid = fork();
    if (pid == 0)
        puts("child");
    else
        puts("parent");

    return 0;
}
```

*Konkurensi*

- Proses *parent* dan *child* berjalan secara konkuren.
  - meskipun pada kode program terlihat sekuensial
- Proses *child* memiliki semua salinan variabel *parent*-nya.

```
int main() {
    pid_t pid; char *msg; int n;

    pid = fork();
    if (pid == 0) {
        msg = "child";
        n = 4;
    }
}
```



```

    } else {
        msg = "parent";
        n = 8;
    }

    while (n-- > 0) { puts(msg); sleep(1); }
    return 0;
}

```

### Latihan

Dengan menggunakan *looping* dan proses *parent* tidak mencetak apapun:

- Buatlah 4 proses *child* untuk mencetak “Hello”!
- Buatlah 5 proses *child* untuk mencetak “Hello”!
- Buatlah 9 proses *child* untuk mencetak “Hello”!

### wait()

#### Fungsi wait()

```
pid_t wait(int *status);
```

- Proses *parent* menunggu hingga salah satu proses *child* selesai<sup>2</sup>.
  - jika *child* sudah selesai, semua *resource*-nya akan dilepaskan
  - lalu proses *parent* melanjutkan eksekusi proses
- *Return value*: PID *child*.
- Argumen **status**: menyimpan *exit status* proses *child*
  - isi dengan NULL jika tidak dipakai

### Contoh

```

int main() {
    pid_t pid; char *msg; int n;

    pid = fork();
    if (pid == 0) { msg = "child"; n = 8; }
    else { msg = "parent"; n = 4; }
}

```

---

<sup>2</sup> lihat 'man 2 wait'

```

    while (n--) { puts(msg); sleep(1); }

    if (pid > 0)    // parent menunggu child selesai
        wait(NULL);

    return 0;
}

```

### Mendapatkan Status Proses Child

Gunakan parameter `status` pada fungsi `wait()` untuk menyimpan *exit status* proses *child* yang telah selesai.

```

if (pid > 0) {
    int status;
    wait(&status);
    printf("child status: %d\n", WEXITSTATUS(status));
}

```

### ZOMBIE

#### Proses Zombie

- Proses *child* sudah selesai, tetapi masih ada di memori.
- Sebab: proses *child* sudah selesai, tetapi *parent* masih berjalan dan tidak memanggil fungsi `wait()`.

#### Contoh

```

// zombie.c
int main()
{
    pid_t pid; char *msg; int n;

    pid = fork();
    if (pid == 0) { msg = "child"; n = 3;}
    else { msg = "parent"; n = 20;}

    while (n--) { puts(msg); sleep(1); }
}

```

```

    return 0;
}

```

### *Mengamati Zombie*

- Jalankan program tersebut pada terminal.
- Buka terminal baru, jalankan `ps f` untuk melihat daftar proses.
  - lakukan dengan cepat, zombie hanya muncul selama 17 detik
- Ciri zombie: status Z dan nama proses `<defunct>`.

| PID   | TTY   | STAT | TIME | COMMAND               |
|-------|-------|------|------|-----------------------|
| 12027 | pts/1 | Ss   | 0:00 | bash                  |
| 12028 | pts/1 | S+   | 0:00 | \_ ./zombie           |
| 12029 | pts/1 | Z+   | 0:00 | \_ [zombie] <defunct> |
| 12031 | pts/2 | Ss   | 0:00 | bash                  |
| 12032 | pts/2 | R+   | 0:00 | \_ ps f               |

### *Latihan*

*// Berapa kali X dicetak, gambarkan pohon prosesnya!*

```

int main()
{
    pid_t pid = fork();
    if (pid != 0)
        fork();
    fork();
    puts("X");
    return 0;
}

```

### *Latihan Buku*

- Kerjakan latihan (buku hlm 149–153) nomor:
  - 3.1
  - 3.2
  - 3.12
  - 3.13
  - 3.14
  - 3.17
- Kerjakan tanpa bantuan komputer terlebih dahulu, lalu cek jawaban anda dengan menjalankan kode program yang diberikan.

FORK, WAIT

*Latihan Tambahan*

- Kerjakan soal nomor:
  - 3.21
  - 3.22 (baca sendiri ttg *shared-memory* antarproses)

---

## SIGNAL

---

### *Sinyal*

- Bentuk komunikasi antarproses (IPC) yang paling sederhana.
- Contoh IPC yang lain<sup>1</sup>:
  - *pipe*
  - *socket*
  - *shared memory*
  - *message passing*

### *Jenis sinyal*

- Ada 31 jenis sinyal standar<sup>2</sup>.
- Beberapa sinyal dapat dikirim langsung oleh *user* ke proses *foreground* dengan menekan kombinasi tombol berikut:
  - Ctrl-C: sinyal *interrupt* (SIGINT)
  - Ctrl-Z: sinyal *terminal stop* (SIGTSTP)
  - Ctrl-\: sinyal *quit* (SIGQUIT)

```
#define SIGHUP      1    // Hangup.
#define SIGINT     2    // Interrupt.
#define SIGQUIT    3    // Quit.
#define SIGILL     4    // Illegal instruction.
#define SIGTRAP    5    // Trace trap.
#define SIGABRT    6    // Abort.
#define SIGBUS     7    // Bus error.
#define SIGFPE     8    // Floating-point exception.
#define SIGKILL    9    // Kill, unblockable.
#define SIGUSR1    10   // User-defined signal 1.
#define SIGSEGV    11   // Segmentation violation.
```

<sup>1</sup> Silberschatz *et al.* (2013), *Operating System Concepts*, hlm 130–147.

<sup>2</sup> lihat *file* /usr/include/bits/signum.h dan *man* 7 signal.

# inter-process communication

(on Linux)

by JULIA EVANS  
@b0rk



in no particular order:

① Write it to a file

② Send it over the local network

(with a HTTP request or something)

③ Use a pipe!

'program1 | program 2'

cool thing: you get buffering automatically

cool thing: you can easily switch to having the 2 programs on different machines.

④ shared memory

processes can share memory, not just threads on the same process!

see `shm_open`

cool thing: very fast/powerful (+scary :))

④ Unix domain sockets

Another way to send a stream of data.

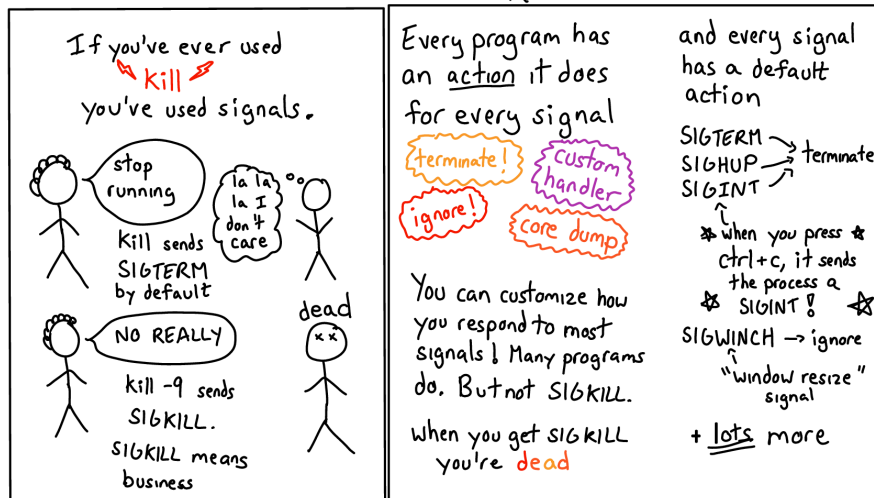
cool thing: you can send file descriptors over a unix domain socket

Gambar 2: Komunikasi antarproses pada Linux

# SIGNALS

## on Unix

julia evans  
@b0rk  
jvns.ca



Gambar 3: Sinyal pada UNIX

```
#define SIGUSR2    12    // User-defined signal 2.
#define SIGPIPE    13    // Broken pipe.
#define SIGALRM    14    // Alarm clock.
#define SIGTERM    15    // Termination.
#define SIGSTKFLT  16    // Stack fault.

#define SIGCHLD    17    // Child status has changed.
#define SIGCONT    18    // Continue.
#define SIGSTOP    19    // Stop, unblockable.
#define SIGTSTP    20    // Keyboard stop.
#define SIGTTIN    21    // Background read from tty.
#define SIGTTOU    22    // Background write to tty.
#define SIGURG     23    // Urgent condition on socket.
#define SIGXCPU    24    // CPU limit exceeded.
#define SIGXFSZ    25    // File size limit exceeded.
#define SIGVTALRM  26    // Virtual alarm clock.
#define SIGPROF    27    // Profiling alarm clock.
#define SIGWINCH   28    // Window size change.
#define SIGIO      29    // I/O now possible.
#define SIGPWR     30    // Power failure restart.
#define SIGSYS     31    // Bad system call.
```

SIGNAL

SIGNAL()

*Fungsi signal()*

```
void signal(int signum, void function(int));
```

- Untuk menangani sinyal yang masuk<sup>3</sup>.
- Jika ada `signum` yang masuk, maka `function` akan dijalankan.

*Contoh*

```
void foo(int sig) {
    printf("got signal %d\n", sig); // print signum
    signal(SIGINT, SIG_DFL);       // back to default
}

int main() {
    signal(SIGINT, foo);
    while (1) {
        puts("hello");
        sleep(1);
    }
}
```

*Penjelasan*

- Jalankan program, kemudian kirim SIGINT (tekan Ctrl-C).
- Karena ada SIGINT masuk, program memanggil fungsi `foo`.
- Kirim lagi SIGINT.
- Apa yang terjadi? Mengapa demikian?
- Apa maksud SIG\_DFL?

KILL()

*Fungsi kill()*

```
int kill(pid_t pid, int signum);
```

---

<sup>3</sup> lihat man 2 signal.



- Untuk mengirim sinyal `signal` ke proses `pid`<sup>4</sup>.

#### Contoh

```
int main()
{
    pid_t child = fork();
    if (child == 0) {
        while (1) {
            puts("child");
            sleep(1);
        }
    } else {
        sleep(5);
        kill(child, SIGTERM); // terminate
    }
    return 0;
}
```

#### Penjelasan

- *Child* akan terus mencetak tiap 1 detik.
- Setelah 5 detik, *parent* mengirim SIGTERM ke *child*.
- *Child* akan berhenti karena mendapat SIGTERM dari *parent*.

#### PAUSE()

#### Fungsi pause()

```
int pause(void);
```

- Untuk menunggu sinyal masuk<sup>5</sup>.

#### Contoh

```
void ding(int sig) { puts("ding!"); }

int main()
```

---

<sup>4</sup> lihat `man 2 kill`.

<sup>5</sup> lihat '`man 2 pause`'.

## SIGNAL

```
{
    if (fork() == 0) {
        sleep(5);
        kill(getppid(), SIGALRM);
    } else {
        signal(SIGALRM, ding);
        puts("waiting...");
        pause();
    }
    return 0;
}
```

### Penjelasan

- *Parent* menunggu sinyal masuk.
- *Child* akan mengirim SIGALRM ke *parent* setelah 5 detik.
- Setelah SIGALRM masuk, *parent* memanggil fungsi *ding*.
- Apa yang terjadi jika *parent* tidak memanggil fungsi `pause()`?

### Tugas

- Modifikasi program contoh hlm 3 pada bagian *parent*, sehingga *child* akan:
  - berjalan selama 4 detik, lalu
  - berhenti sementara (*stop*) selama 3 detik, lalu
  - lanjut lagi berjalan (*continue*) selama 2 detik, lalu
  - berhenti (*terminate*)
- Jika benar, *child* akan mencetak 6 kali.
- Kumpulkan di LMS berupa satu *file* dengan nama [NIM].c.

### Tugas Bonus

- Implementasikan sendiri fungsi `system()` anda sesuai penjelasan yang tertera pada manual<sup>6</sup>.
  - gunakan fungsi `fork()`, `execl()`, `wait()`, dan `signal()`
  - coba jalankan beberapa perintah memakai fungsi tsb
- Kumpulkan di LMS berupa satu *file* dengan nama [NIM].c.
  - **opsional**, plagiasi akan mendapat sanksi nilai -100

---

<sup>6</sup> lihat man 3 system.

- paling lambat besok pukul 06:00



---

## UNIX SHELL

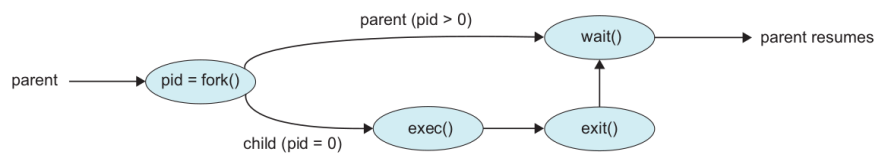
---

### SHELL

#### *Cara Kerja Shell*

1. Baca masukan perintah dari pengguna
2. Pisahkan perintah per argumen (per kata)
3. Buat proses *child* (**fork**)
4. Jalankan perintah (**exec**)
5. Tunggu sampai *child* selesai (**wait**)
6. Kembali ke nomor 1

#### *Membuat Proses Baru*



### TUGAS

#### *UNIX Shell*

- Lihat **Project 1**<sup>1</sup>, kerjakan **Part I**, yaitu membuat *shell*
  - lengkapi contoh program berikut<sup>2</sup>
- Kumpulkan di LMS dengan nama *file* **NIM\_shell.c**
  - paling lambat setelah praktikum berakhir

<sup>1</sup> Silberschatz *et al.* (2013), *Operating System Concepts*, hlm 157–159

<sup>2</sup> <https://git.io/vycRv>

*History Feature*

- Lanjutkan **Part II**, yaitu membuat fitur *history*
  - **opsional**, bonus nilai
  - plagiasi akan mendapat sanksi nilai -100
- Kumpulkan di LMS dengan nama *file* `NIM_shellhist.c`
  - paling lambat besok pukul 06:00

---

## THREAD

---

### THREAD

#### *Thread*

- *thread* adalah satuan dasar utilisasi CPU<sup>1</sup>
- tiap *thread* memiliki:
  - id, *program counter*, *register set*, dan *stack*
- dalam satu proses, *thread* berbagi:
  - segmen *code*, segmen *data*, dan sumberdaya lainnya, seperti *file*
- proses *multithreaded* memiliki beberapa *thread* yang dapat mengerjakan beberapa tugas secara bersamaan

#### *POSIX Thread*

- UNIX memakai standar POSIX<sup>2</sup> *thread* (pthread)
- saat kompilasi tambahkan *flag* `-pthread`

#### *Membuat Thread*

```
pthread_create(&thread, attr, func, arg);
```

- membuat satu `thread` dengan atribut `attr` yang akan menjalankan fungsi `func` dengan argumen `arg`<sup>3</sup>
- deklarasi fungsi tersebut:
  - `void *func(void *arg);`<sup>4</sup>

---

<sup>1</sup> Silberschatz *et al.* (2013), *Operating System Concepts*, hlm 163.

<sup>2</sup> The Portable Operating System Interface

<sup>3</sup> lihat '`man pthread_create`'

<sup>4</sup> `void*`: tipe data *generic pointer*

## THREAD

### *Menunggu Thread*

```
pthread_join(thread, &retval);
```

- menunggu *thread* selesai dan menyimpan keluarannya ke variabel `retval`<sup>5</sup>

### *Mengakhiri Thread*

```
pthread_exit(retval);
```

- mengakhiri *thread* dengan nilai keluaran `retval`<sup>6</sup>

## CONTOH

### *Satu Thread Tanpa Argumen*

```
#include <pthread.h>
#include <stdio.h>

void *hello(void *arg) {
    printf("hello\n");
    pthread_exit(NULL);
}

int main() {
    pthread_t thread;
    pthread_create(&thread, NULL, hello, NULL);
    pthread_join(thread, NULL);
    return 0;
}
```

### *Dua Thread Tanpa Argumen*

```
int main() {
    pthread_t thread1;
    pthread_t thread2;

    pthread_create(&thread1, NULL, hello, NULL);
    pthread_create(&thread2, NULL, hello, NULL);
}
```

---

<sup>5</sup> lihat 'man pthread\_join'

<sup>6</sup> lihat 'man pthread\_exit'



```

    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    return 0;
}

```

*Banyak Thread Tanpa Argumen*

```

#define N 4

int main() {
    pthread_t thread[N];
    int i;

    for (i = 0; i < N; i++)
        pthread_create(&thread[i], NULL, hello, NULL);

    for (i = 0; i < N; i++)
        pthread_join(thread[i], NULL);

    return 0;
}

```

*Satu Thread Dengan Argumen*

```

#include <pthread.h>
#include <stdio.h>

void* hello(void* arg) {
    printf("hello from thread %s\n", (char*)arg);
    pthread_exit(NULL);
}

int main() {
    pthread_t thread;
    pthread_create(&thread, NULL, hello, "0");
    pthread_join(thread, NULL);
    return 0;
}

```

## THREAD

### *Dua Thread Dengan Argumen*

```
int main() {
    pthread_t thread1;
    pthread_t thread2;

    pthread_create(&thread1, NULL, hello, "0");
    pthread_create(&thread2, NULL, hello, "1");

    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    return 0;
}
```

### *Banyak Thread Dengan Argumen*

```
#define N 4

int main() {
    pthread_t thread[N];
    char *id[N] = {"0", "1", "2", "3"};
    int i;

    for (i = 0; i < N; i++)
        pthread_create(&thread[i], NULL, hello, id[i]);

    for (i = 0; i < N; i++)
        pthread_join(thread[i], NULL);

    return 0;
}
```

## LATIHAN

### *Jumlah Array*

- lengkapi program berikut untuk menjumlahkan nilai semua elemen *array* A
- gunakan variabel global `sum` untuk menyimpan hasilnya

```
#include <stdio.h>
#define N 16
```

```

int sum = 0;

int main() {
    int A[N] = {68,34,64,95,35,78,65,93,
                51,67, 7,77, 4,73,52,91};
    // TODO: array sum
    printf("%d\n", sum);    // 954
    return 0;
}

```

#### *Jumlah Array (Satu Thread)*

- sekarang, buat satu buah *thread* untuk menjumlahkan nilai semua elemen *array* A dengan fungsi `array_sum()`
- *thread* utama hanya membuat dan menunggu *thread* ini selesai

#### *Jumlah Array (Dua Thread)*

- oke?
- sekarang gunakan 2 buah *thread* untuk menjumlahkan nilai semua elemen *array* A
- pastikan pembagian kerja antara kedua *thread* seimbang, yaitu tiap *thread* memproses  $\frac{N}{2}$  elemen

#### *Jumlah Array (Empat Thread)*

- bisa?
- sekarang gunakan 4 buah *thread* untuk menjumlahkan nilai semua elemen *array* A
- pastikan pembagian kerja antara keempat *thread* seimbang, yaitu tiap *thread* memproses  $\frac{N}{4}$  elemen
- kumpulkan di LMS



---

## SINKRONISASI THREAD

---

### CRITICAL SECTION

#### *Critical Section*

A **critical section** is a section of code that can be executed by at most **one process at a time**. The critical section exists to protect shared resources from multiple access.<sup>1</sup>

- contoh: mengubah variabel global, menulis ke *file*, dll.
- solusi: sinkronisasi

#### *Sinkronisasi*

- untuk melindungi (mengunci) sebuah *critical section*
  - hanya satu proses/*thread* dalam satu waktu yang dapat masuk
- menggunakan *mutex lock* atau *semaphore*

### MUTUAL EXCLUSION

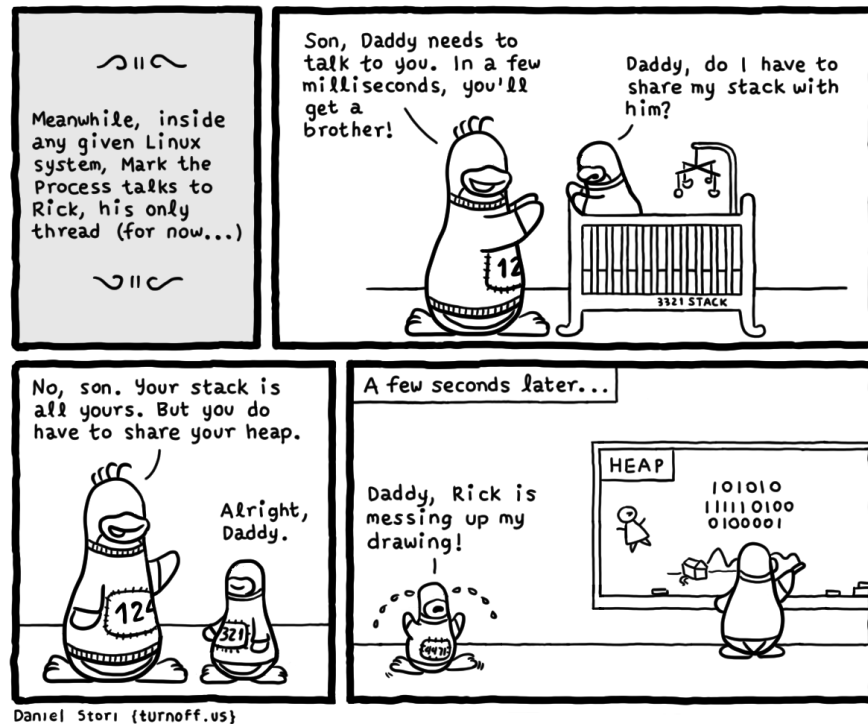
#### *Mutex*

Mutex is a key to a variable. One thread can have the key—modify the variable—at the time. When finished, the thread gives (frees) the key to the next thread in the group.<sup>2</sup>

---

<sup>1</sup> Jones (2008), *GNU/Linux Application Programming*, hlm 264.

<sup>2</sup> <http://koti.mbnet.fi/niclasw/MutexSemaphore.html>

Gambar 4: *Don't share mutable state**Fungsi Mutex*

```
#include <pthread.h>

int pthread_mutex_init(pthread_mutex_t *mutex,
                       const pthread_mutexattr_t *attr);
int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
int pthread_mutex_destroy(pthread_mutex_t *mutex);
```

- init: inialisasi mutex
- lock: mengunci *critical section*
- unlock: melepaskan kunci *critical section*
- destroy: menghapus mutex

*Latihan*

Apa yang salah dengan kode berikut ini? Perbaiki dengan menggunakan *mutex*!

```
// counting to one million
#include <stdio.h>
#include <pthread.h>
```

```

#define N 1000000
#define T 4

int count = 0;

void *counting(void *arg)
{
    int i;
    for (i = 0; i < N/T; i++)
        count++;           // critical section

    pthread_exit(NULL);
}

int main()
{
    pthread_t t[T];
    int i;

    for (i = 0; i < T; i++)
        pthread_create(&t[i], NULL, counting, NULL);

    for (i = 0; i < T; i++)
        pthread_join(t[i], NULL);

    printf("%d\n", count);    // 1000000, no?
    return 0;
}

```

## SEMAPHORE

*Semaphore*

- nilai *semaphore* *S* diinisialisasi dengan bilangan non-negatif
- terdapat dua operasi atomik yang bisa dilakukan pada *semaphore*, yaitu *wait* dan *post*<sup>3</sup>

```

wait(S) {
    while (S == 0)
        ; // busy wait
    S--;

    post(S) {

```

<sup>3</sup> Silberschatz *et al.* (2013), *Operating System Concepts*, hlm 214.

```

        S++;
    }

```

### *Jenis Semaphore*

1. *Counting semaphore*, nilai awal *semaphore* lebih dari 1
2. *Binary semaphore*, nilai awal *semaphore* adalah 1 (sama fungsinya dengan *mutex*)

### *Fungsi Semaphore*

```
#include <semaphore.h>
```

```

int sem_init(sem_t *sem, int pshared, unsigned int value);
int sem_wait(sem_t *sem);
int sem_post(sem_t *sem);
int sem_destroy(sem_t *sem);

```

- *init*: inisialisasi *sem* dengan nilai awal *value*
- *wait*:
  - jika *sem* = 0 → *block*
  - jika *sem* > 0 → *sem--*, *continue*
- *post*: *sem++*
- *destroy*: menghapus *sem*

### *Latihan*

Perbaiki latihan sebelumnya dengan menggunakan *semaphore*!

### TUGAS

#### *Array Sum*

Identifikasi *critical section* dan perbaiki kode berikut ini supaya hasilnya benar.

```

#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>

#define N 100000
#define T 4

```



```

int sum = 0;

void *array_sum(void *arg)
{
    int *array = (int*)arg;    // cast void* --> int*
    int i;

    for (i = 0; i < N/T; i++)
        sum += array[i];

    pthread_exit(NULL);
}

int main()
{
    pthread_t t[T];
    int A[N], i;

    for (i = 0; i < N; i++)
        A[i] = rand()%10;

    for (i = 0; i < T; i++)
        pthread_create(&t[i], NULL, array_sum, &A[i*N/T]);

    for (i = 0; i < T; i++)
        pthread_join(t[i], NULL);

    printf("%d\n", sum);    // 448706
    return 0;
}

```



Bagian II

ADMINISTRASI SISTEM UNIX



---

## UNIX INTRO

---

### PENDAHULUAN

#### *Mengapa CLI?*

- administrasi sistem
  - hampir semua server berbasis UNIX
  - perintah sama, meskipun berbeda OS
  - bisa otomatisasi dengan *script*
- pilihan aplikasi lebih banyak
- efektif dan efisien

#### *Server SO*

- Raspberry Pi 3 Model B<sup>1</sup>
  - 1.2 GHz quad-core ARMv8 CPU
  - 1 GiB RAM
- Raspbian Lite GNU/Linux

#### *Login Server*

- *Host*: `os.apps.cs.ipb.ac.id`
- *Username*: lihat LMS
- *Password*: 6 digit terakhir NIM

#### *Sumber Tambahan*

- UNIX commands list<sup>2</sup>

---

<sup>1</sup> <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>

<sup>2</sup> [https://en.wikipedia.org/wiki/Template:Unix\\_commands](https://en.wikipedia.org/wiki/Template:Unix_commands)



Gambar 5: Raspberry Pi

- Explain Shell<sup>3</sup>
- Regular expression checker<sup>4</sup>

## PERINTAH DASAR

### *ssh*

Login ke komputer *remote*.

**ssh** [OPTION] [USER@]HOST

- **-p**: nomor *port* untuk koneksi
- **-X**: mengaktifkan X11 *forwarding*

### *echo*

Menampilkan satu baris teks.

**echo** [OPTION] [STRING]

- **-n**: tanpa *newline* di akhir
- **-e**: mengaktifkan interpretasi *backslash escape*

---

<sup>3</sup> <http://explainshell.com>

<sup>4</sup> <http://regexr.com/>

*hostname*

Menampilkan nama *host* sistem.

`hostname` [OPTION]

- `-I`: menampilkan alamat IP *host*

*uname*

Menampilkan informasi kernel sistem.

`uname` [OPTION]

- `-a`: menampilkan semua informasi

*uptime*

Menampilkan berapa lama sistem sudah berjalan.

`uptime`

*date*

Mencetak tanggal dan waktu sistem.

`date` [+FORMAT]

*cal*

Menampilkan kalender.

`cal` [[MONTH] YEAR]

*whoami*

Menampilkan nama *user* yang sedang dipakai.

`whoami`

*who*

Menampilkan siapa saja yang sedang *log in*.

`who` [OPTION]

- `-q`: menampilkan semua *username* dan jumlahnya
- `-w`: menampilkan status *message* pengguna (+, -, atau ?)

*w*

Menampilkan siapa saja yang sedang *log in* dan apa yang dilakukannya.

`w` [USER]

*last*

Menampilkan daftar waktu *user* terakhir kali *login*.

`last`

*write*

Mengirim pesan ke *user* lain yang sedang *login*.

`write` USER [TTY]

*mesg*

Mengontrol akses masuk pesan ke terminal anda (ya/tidak).

`mesg` [y|n]

*mail*

Mengecek atau mengirim surat ke *user* lain.

`mail` [USER]

*passwd*

Mengganti *password* login.



`passwd` [OPTION] [USER]

- `-d`: menghapus *password* (*delete*)
- `-e`: membuat *password* kadaluwarsa (*expired*)

*logout*

Keluar dari sistem.

`logout`

*poweroff*

Mematikan (*shutdown*) sistem.

`sudo poweroff`

*reboot*

Me-*restart* sistem.

`sudo reboot`

*Tombol* shortcut

*Tab*

*auto-completion*

*Up dan Down*

mengakses *history* perintah yang pernah dimasukkan

*Ctrl+D*

mengakhiri teks (*end of file*, EOF)

*Ctrl+C*

mengakhiri proses yang sedang berjalan

*Ctrl+L*

membersihkan layar

*Ctrl+W*

menghapus satu kata di belakang kursor

*Tugas*

- ganti *password* anda
- kirim surat ke *user auriza*

---

## FILE DAN DIREKTORI

---

### FILE DAN DIREKTORI

#### *pwd*

Mencetak nama direktori saat ini.

`pwd`

#### *cd*

Mengganti direktori.

`cd [DIRECTORY]`

Jika tanpa parameter `DIRECTORY`, maka `cd` akan mengganti ke direktori *home* (`~`).

#### *ls*

Menampilkan daftar isi direktori.

`ls [OPTION] [FILE]`

- `-a`: tampilkan juga *dotfile*
- `-h`: mencetak ukuran dalam format yang mudah dibaca
- `-i`: cetak nomor indeks setiap *file*
- `-l`: gunakan format panjang
- `-r`: balik urutan *sorting*
- `-S`: *sorting* berdasarkan ukuran

### *touch*

Meng-*update* waktu akses dan modifikasi suatu **FILE**.

**touch** FILE

Jika **FILE** belum ada, maka **touch** akan membuat **FILE** kosong.

### *mkdir*

Membuat direktori.

**mkdir** [OPTION] DIRECTORY

- **-p**: buar direktori *parent* jika diperlukan

### *cp*

Menyalin *file* dan direktori.

**cp** [OPTION] SOURCE DEST

**cp** [OPTION] SOURCES... DIRECTORY

- **-f**: tanpa konfirmasi jika terjadi *overwrite*
- **-i**: meminta konfirmasi sebelum *overwrite*
- **-r**: salin direktori secara rekursif

### *mv*

Memindahkan (mengganti nama) *file*.

**mv** [OPTION] SOURCE DEST

**mv** [OPTION] SOURCES... DIRECTORY

- **-f**: tanpa konfirmasi jika terjadi *overwrite*
- **-i**: meminta konfirmasi sebelum *overwrite*

### *rm*

Menghapus *file* atau direktori.

**rm** [OPTION] FILE...

- **-f**: tanpa konfirmasi, abaikan jika *file* tidak ada
- **-i**: meminta konfirmasi setiap kali menghapus
- **-r**: hapus direktori dan isinya secara rekursif

*rmdir*

Menghapus direktori kosong.

`rmdir` [OPTION] DIRECTORY...

- `-p`: hapus DIRECTORY dan pendahulunya; misal: `'rmdir -p a/b/c'` sama dengan `'rmdir a/b/c a/b a'`

## Dotfile

*File* yang namanya diawali dengan tanda titik. Secara umum, *dotfile* tidak akan terlihat (*hidden*). Biasanya digunakan untuk menyimpan konfigurasi program.

*Simbol*

~

direktori *home* pengguna (`/home/$USER/`)

.

direktori saat ini

..

direktori *parent* dari direktori saat ini

/

direktori *root*, yaitu direktori paling atas

## Path

## Absolute

*path* ditulis lengkap dari direktori *root*; contoh: `'/etc'`

## Relative

*path* ditulis relatif terhadap posisi saat ini; contoh: ‘../etc’

## Tugas

- masuk ke direktori *home* anda
- buat direktori **public\_html**
- masuk ke direktori tersebut
- unduh templat resume berikut ke sini
  - <https://raw.githubusercontent.com/auriza/os-lab/master/txt/bio.html>
- ubah nama *file* menjadi **resume.html**
- edit *file* sesuai dengan data anda
- untuk melihat hasilnya, buka halaman web berikut
  - <http://os.apps.cs.ipb.ac.id/~username/resume.html>

---

## PERMISSION, LINK, PIPE, REDIRECTION

---

### Ownership

- Tiap *file* memiliki *owner*
  - hanya *superuser* yang dapat mengubah kepemilikan *file*
- Tiap *file* memiliki *permission*
  - mengatur hak akses *file* tersebut

### Permission

- Tiga jenis *permission*:

| <i>Permission</i> | <i>File</i>    | <i>Directory</i>           |
|-------------------|----------------|----------------------------|
| <b>r</b>          | <i>read</i>    | <i>list files</i>          |
| <b>w</b>          | <i>write</i>   | <i>add or remove files</i> |
| <b>x</b>          | <i>execute</i> | <i>enter the directory</i> |

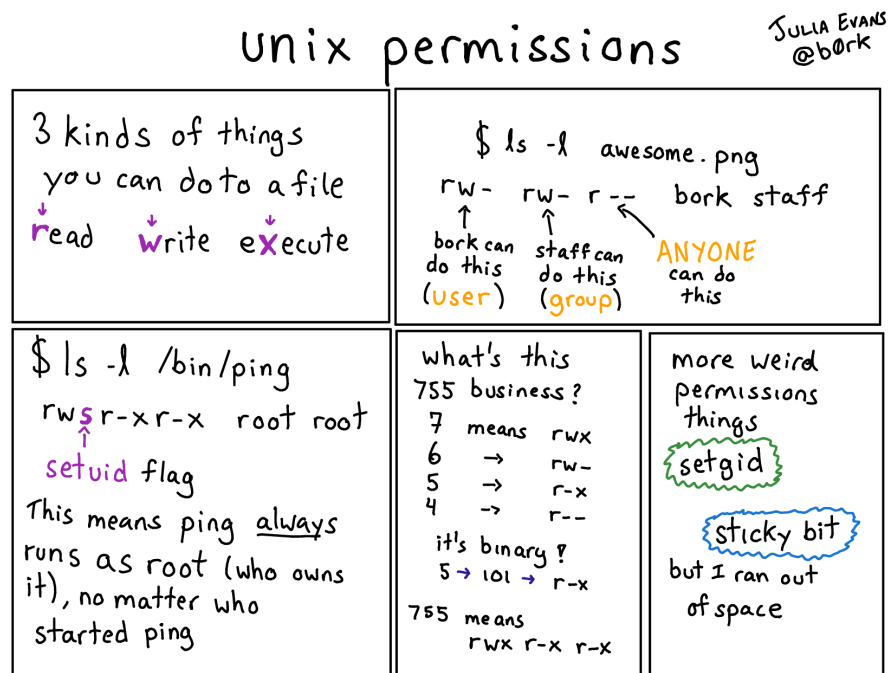
- Tiga jenis *user*:
  - *user owner* (**u**)
  - *group owner* (**g**)
  - *others* (**o**)

### **su**

Berubah menjadi *user* lain atau menjadi *superuser*.

**su** [OPTION] [USERNAME]

- **-c** **COMMAND**: menjalankan perintah **COMMAND**
- **-l**: seperti *login* langsung (*default environment*)



Gambar 6: UNIX permissions

*chown*

Mengubah kepemilikan suatu *file*.

**chown** [OPTION] [OWNER] [:GROUP] FILE

- -R: rekursif

*chmod*

Mengganti mode *permission* suatu *file*.

**chmod** [OPTION] MODE[,MODE]... FILE...

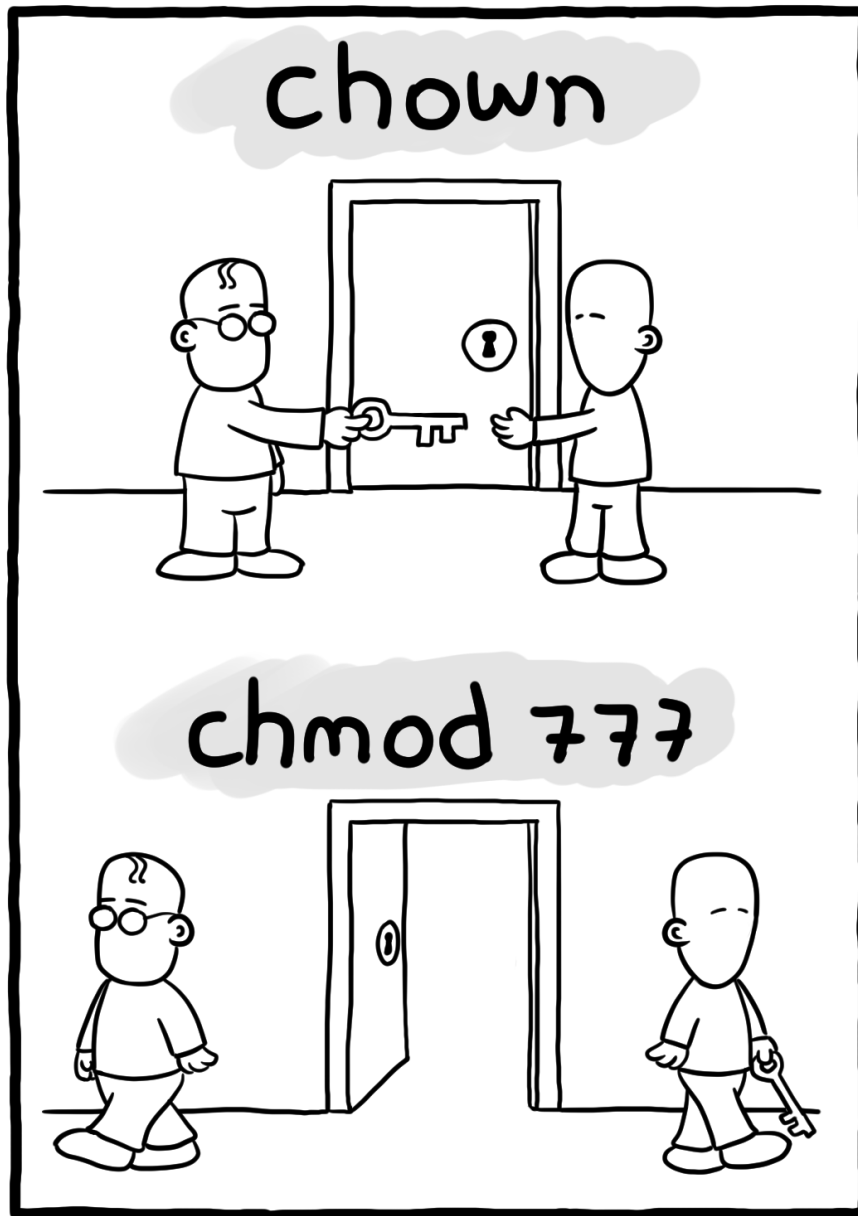
**chmod** [OPTION] OCTAL-MODE FILE...

- -R: rekursif

*Format chmod*

- Format mode simbolis:
  - [ugoa] [+ -=] [rwxX]
- Format mode numerik:
  - digit oktal = 4 (read) + 2 (write) + 1 (exec)





Daniel Stori {turnoff.us}

Gambar 7: Membuka akses untuk publik

- *Catatan:* opsi *permission* X hanya akan mengeset bit *execute* untuk direktori saja

### *Contoh chmod*

- `r--r--r--`
  - `chmod a=r FILE`
  - `chmod 444 FILE`
- `rw-rw----`
  - `chmod ug=rw,o= FILE`
  - `chmod 660 FILE`
- `rwxr-xr-x`
  - `chmod a=rx,u+w FILE`
  - `chmod 755 FILE`

### *Link*

#### 1. *Hard link*

- mengacu pada nomor indeks *file* (inode)
- tidak terpengaruh terhadap perubahan nama *file*
- namun hanya bisa dalam satu partisi yang sama

#### 2. *Symbolic link*

- mengacu pada nama *file*
- bisa lintas partisi
- bisa membuat *link* ke direktori
- namun jika nama *file* yang dirujuk berubah akan mengakibatkan *broken link*

### *ln*

Membuat *link* antar-*file*.

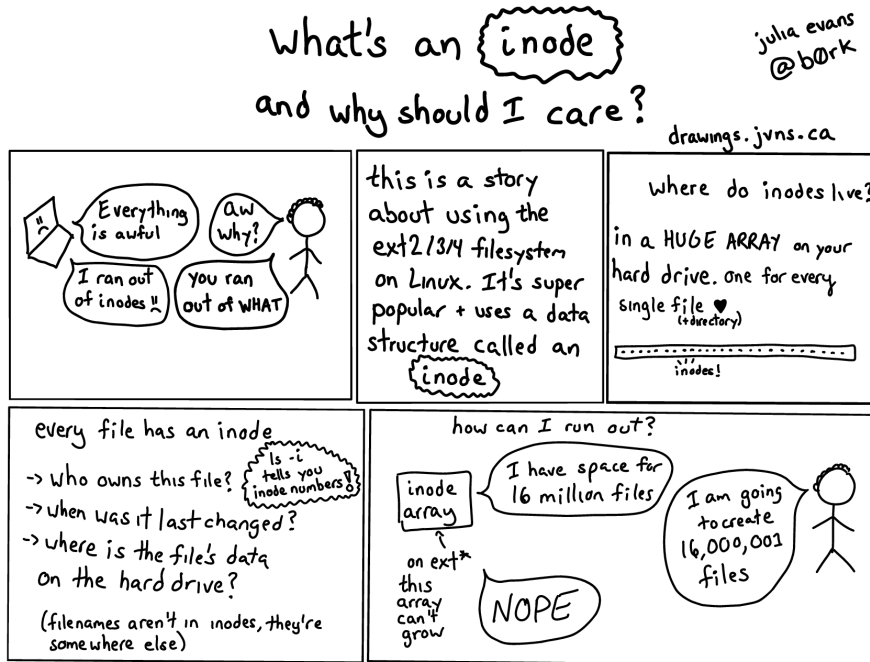
`ln [OPTION] TARGET LINK-NAME`

- `-s`: *symbolic link*

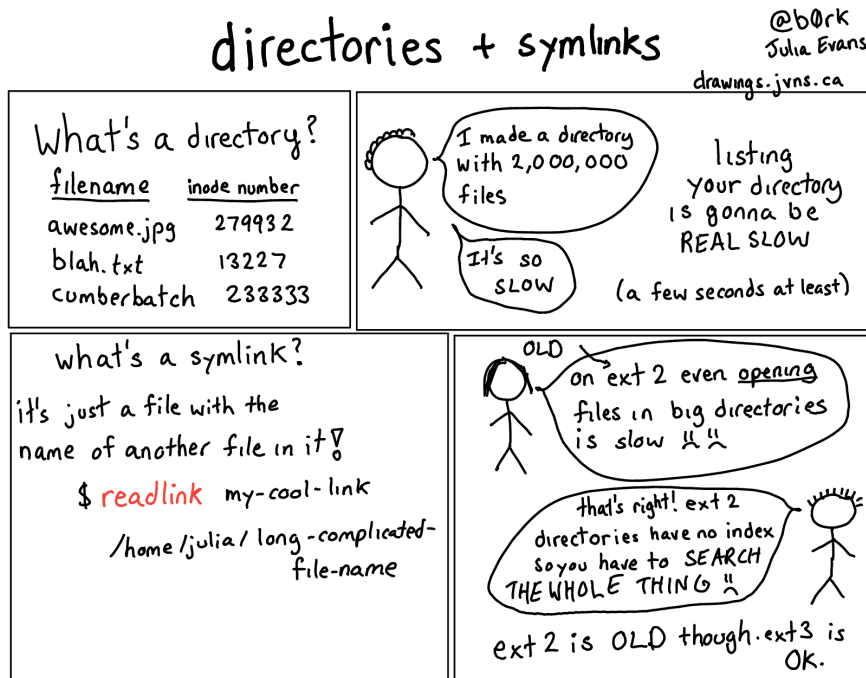
### *Stream Standar*

Setiap proses yang berjalan memiliki tiga *stream* standar I/O:

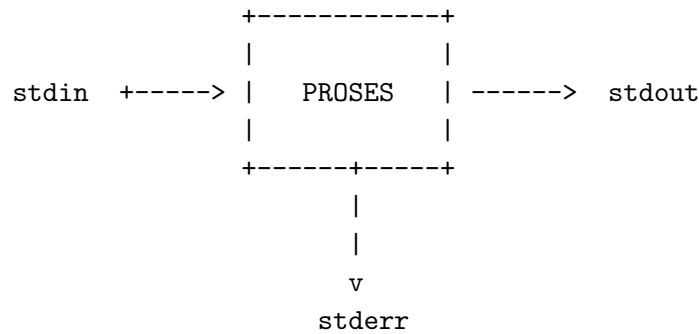
- *standard input* (`stdin`)
- *standard output* (`stdout`)
- *standard error* (`stderr`)



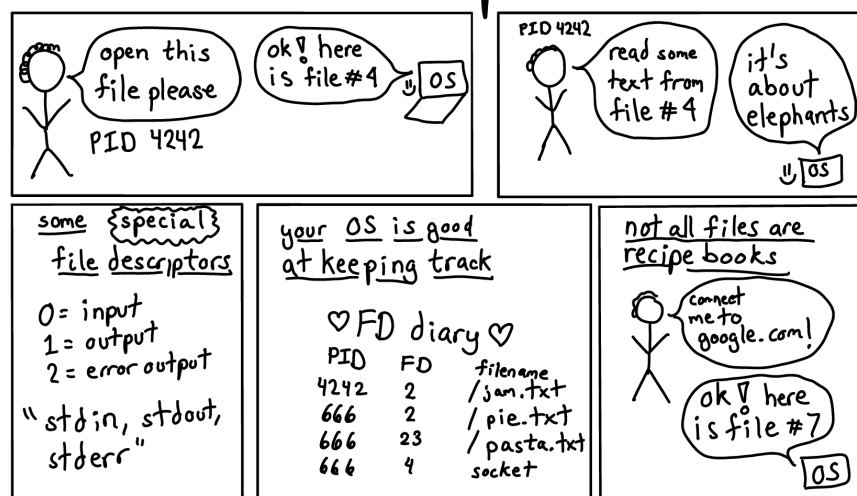
Gambar 8: Inode



Gambar 9: Direktori dan symlink



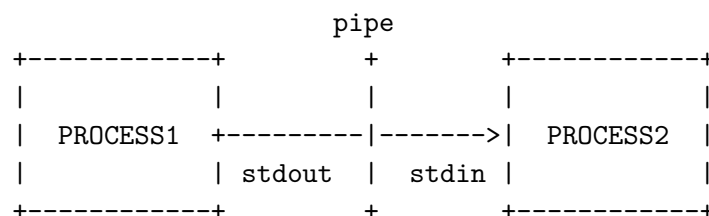
let's learn about  
♥ file descriptors ♥



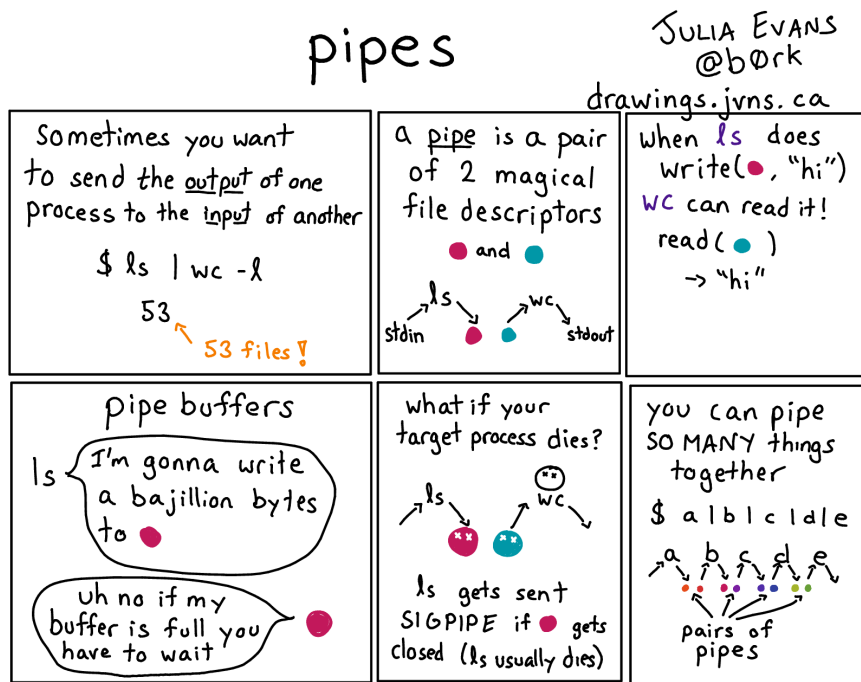
Gambar 10: File descriptor

## Pipe

- Menyalurkan *output* proses menjadi *input* proses selanjutnya
- Berguna untuk membuat *pipeline* perintah



- Contoh:
  - echo "halo" | rev
  - echo "2 + 5" | bc
  - who | wc -l



Gambar 11: Pipe

## Redirect

- Mengarahkan *stream* standar proses ke suatu *file* yang ditentukan oleh pengguna

| Karakter | Redirect                 |
|----------|--------------------------|
| <        | stdin                    |
| >        | stdout                   |
| >>       | stdout ( <i>append</i> ) |
| 2>       | stderr                   |

- Contoh:

```
date > now.txt 2> err.txt
```

```
rev < now.txt
```

```
rev < now.txt > rev.txt
```



## PENCARIAN DAN PEMROSESAN TEKS

### PENCARIAN

*man*

Mencari halaman manual suatu program, fungsi, dan sebagainya.

**man** [SECTION] PAGE

- q: (*quit*) keluar
- /PATTERN: pencarian kata
  - n: (*next*) lanjutkan pencarian kata
  - N: (*next-reverse*) lanjutkan pencarian kata mundur



Gambar 12: Bagian halaman manual

*which*

Mencari lokasi *file* program.

**which** COMMAND

*locate*

Mencari lokasi *file* berdasarkan namanya pada *database*.

`locate` [OPTION] PATTERN

- `-i`: (*insensitive*) abaikan kapitalisasi
- `-c`: (*count*) cetak jumlah *file* yang ditemukan

*find*

Mencari *file* langsung pada sebuah hierarki direktori.

`find` [PATH] [TEST]...

- `-name` PATTERN: nama *file*
- `-iname` PATTERN: nama *file* (*case insensitive*)
- `-size` [+]*N*[kMG]: ukuran *file* sebesar *N*
- `-atime` [+]*N*: *file* terakhir diakses *N* hari yang lalu
- `-mtime` [+]*N*: *file* terakhir dimodifikasi *N* hari yang lalu
- `-empty`: *file* kosong
- `-type` [df]: jenis *file* (direktori atau *file* biasa)

*xargs*

Mengubah tiap baris masukan menjadi argumen untuk suatu perintah.

`xargs` [OPTION] COMMAND

- `-L` *N*: gunakan maksimum *N* baris argumen untuk tiap perintah

*Contoh*

```
## temukan semua file backup (.bak) di direktori ini,
## lalu hapus satu-per-satu
find . -name "*.bak" | xargs -L 1 rm
```

*grep*

Mencetak baris *file* yang cocok dengan suatu pola.

`grep` [OPTION] PATTERN FILE

- `-c`: (*count*) tampilkan jumlah baris
- `-i`: (*insensitive*) abaikan kapitalisasi
- `-v`: (*invert*) kebalikan dari pola yang diberikan



- `-r`: rekursif

## PEMROSESAN TEKS

### *editor (nano)*

Membuat dan mengedit *file* teks.

`editor [OPTION] [FILE]...`

- `-i`: (*indent*) indentasi otomatis
- `-u`: (*undo*) mengaktifkan fitur *undo*
- `^O`: (*write-out*) menyimpan *file*
- `^X`: (*exit*) keluar dari `editor`

### *vi*

Editor teks untuk *programmer*.

`vi [OPTION] [FILE]...`

### *pager (less)*

Menampilkan *file* teks per halaman sesuai ukuran layar.

`pager [FILE]...`

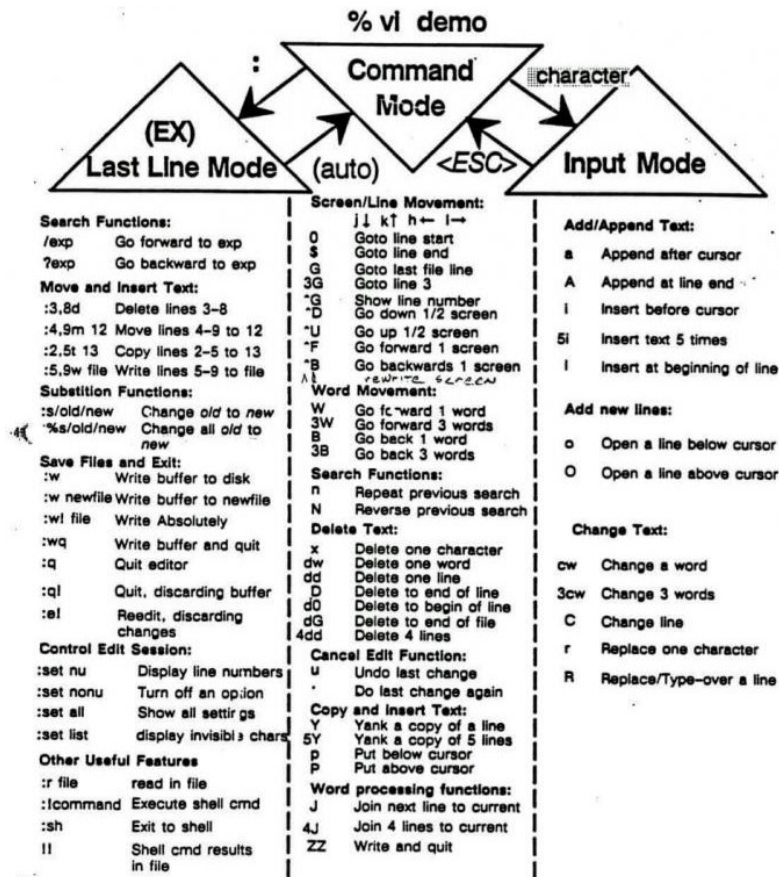
- `q`: (*quit*) keluar
- `/PATTERN`: pencarian kata
  - `n`: (*next*) lanjutkan pencarian kata
  - `N`: (*next-reverse*) lanjutkan pencarian kata mundur

### *cat*

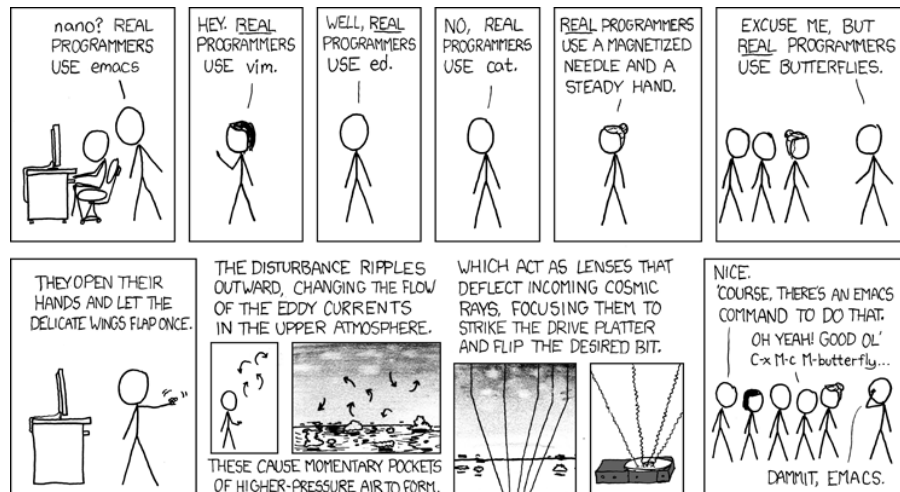
Menggabungkan *file* dan menampilkannya ke layar.

`cat [OPTION] [FILE]...`

- `-n`: (*number*) berikan nomor semua baris
- `-b`: (*blank*) berikan nomor baris yang ada isinya saja
- `-s`: (*squeeze*) hilangkan baris kosong yang berulang



Gambar 13: Petunjuk singkat vi



Gambar 14: Real programmers

*split*

Memecah *file* menjadi beberapa bagian.

**split** [OPTION] FILE [PREFIX]

- **-b** N: (*bytes*) pecah per *N* byte
- **-l** N: (*lines*) pecah per *N* baris

*sort*

Mengurutkan tiap baris pada *file* teks

**sort** [OPTION] [FILE]

- **-n**: (*numeric*) urutkan secara numerik
- **-r**: (*reverse*) urutkan terbalik

*uniq*

Menampilkan baris yang unik saja

**uniq** [OPTION] [FILE]

- **-c**: (*count*) tambahkan jumlah kemunculan di awal baris
- **-d**: (*duplicate*) hanya cetak baris yang berulang
- **-i**: (*ignore-case*) abaikan perbedaan huruf kecil/kapital
- **-u**: (*unique*) hanya cetak baris yang tidak berulang

*head*

Menampilkan bagian awal *file*.

**head** [OPTION] [FILE]

- **-n** K: (*lines*) tampilkan *K* baris pertama
- **-b** K: (*bytes*) tampilkan *K* byte pertama

*tail*

Menampilkan bagian akhir *file*.

**tail** [OPTION] [FILE]

- **-n** K: (*lines*) tampilkan *K* baris terakhir
- **-b** K: (*bytes*) tampilkan *K* byte terakhir

## *tr*

Translasi karakter dari set pertama ke set kedua.

`tr` [OPTION] SET1 [SET2]

- `-d`: (*delete*) menghapus karakter yang terdapat pada SET1
- `-s`: (*squeeze*) menghapus karakter yang berulang dari SET1

## *sed*

*Stream editor*, memanipulasi *string* dengan ekspresi reguler.

`sed` [OPTION] 's/SEARCH/REPLACE/' [FILE]

- `-e`: (*execute*) menambahkan perintah untuk dieksekusi
- `-i`: (*in-place*) mengedit *file* langsung

## *cut*

Mengambil sebagian karakter/kolom dari sebaris teks.

`cut` OPTION [FILE]

- `-c N-M`: (*characters*) cetak karakter ke-N hingga M
- `-f N-M`: (*fields*) cetak kolom ke-N hingga M
- `-d DELIM`: (*delimiter*) karakter pemisah antarkolom

## *paste*

Menggabungkan baris-baris tiap *file*.

`paste` [OPTION] [FILE]

- `-d`: (*delimiter*) karakter pemisah antarkolom
- `-s`: (*serial*) proses tiap *file* satu per satu

## EKSPRESI REGULER

### *Ekspresi Reguler (Regex)*

Regex adalah susunan karakter yang merupakan pola pencarian. Regex digunakan untuk mencari *string* tertentu pada teks.

Misal, ekspresi reguler `/G64\d{6}/` dapat mencocokkan NIM semua mahasiswa S1 Ilmu Komputer IPB.

Latihan dan informasi lebih lanjut, kunjungi <http://regexr.com>.

### *Referensi Singkat*

- Karakter
  - . karakter apapun selain *newline*
  - `[ABC]` karakter `a`, `b`, atau `c`
  - `[^ABC]` bukan karakter `a`, `b`, atau `c`
  - `[A-G]` karakter antara `a` sampai `g`
- Kelas Karakter
  - `\w` kata, `[A-Za-z0-9_]`
  - `\d` digit, `[0-9]`
  - `\s` *whitespace*
- *Anchor*
  - `^` awal baris
  - `$` akhir baris

### *Referensi Singkat*

- Jumlah
  - `*` 0 atau lebih
  - `+` 1 atau lebih
  - `?` 0 atau 1
  - `{3}` tepat 3
  - `{3,}` 3 atau lebih
  - `{3,5}` antara 3 sampai 5
- Grup
  - `(...)` membuat grup
  - `\N` referensi balik grup ke-*n*

### *Tugas*

UNIX Text Processing Contest 2017x<sup>1</sup>

<sup>1</sup> <https://www.hackerrank.com/unix-text-processing-2017x>



Gambar 15: *Regex saves the day*

---

## PROCESS AND JOB CONTROL

---

### PROSES

#### *ps*

Menampilkan cuplikan informasi proses yang sedang berjalan.

**ps** [OPTION]

- **-e**: (*every*) semua proses
- **-f**: (*full*) format lengkap
- **-L**: (*lightweight*) tampilkan informasi *thread*
- **--forest**: pohon proses

#### *pstree*

Menampilkan pohon proses.

**pstree** [OPTION] [PID|USER]

- **-n**: (*numeric*) urutkan berdasarkan PID
- **-p**: (PID) tampilkan PID

#### *top*

Memonitor proses.

**top** [OPTION]

- **-u USER**: proses milik USER tertentu saja
- **-p PID**: proses PID tertentu saja

#### *htop*

Memonitor proses secara interaktif.

`htop` [OPTION]

- `-u USER`: proses milik `USER` tertentu saja
- `-p PID`: proses `PID` tertentu saja

*pgrep*

Mendapatkan PID suatu proses berdasarkan namanya.

`pgrep` [OPTION] PATTERN

- `-u USER`: proses milik `USER` tertentu saja

*kill*

Mengirim sinyal ke suatu proses (*default* `SIGTERM`).

`kill` [OPTION] PID

- `-SIG`: mengirim sinyal `SIG`
- `-l`: (*list*) menampilkan semua daftar sinyal

*pmap*

Menampilkan *memory map* sebuah proses.

`pmap` [OPTION] PID ...

- `-x`: *extended format*

*lsof*

Menampilkan daftar *file* yang sedang dibuka oleh proses.

`lsof` [OPTION] [FILENAME]

- `-p PID`: proses `PID` tertentu saja

*nice*

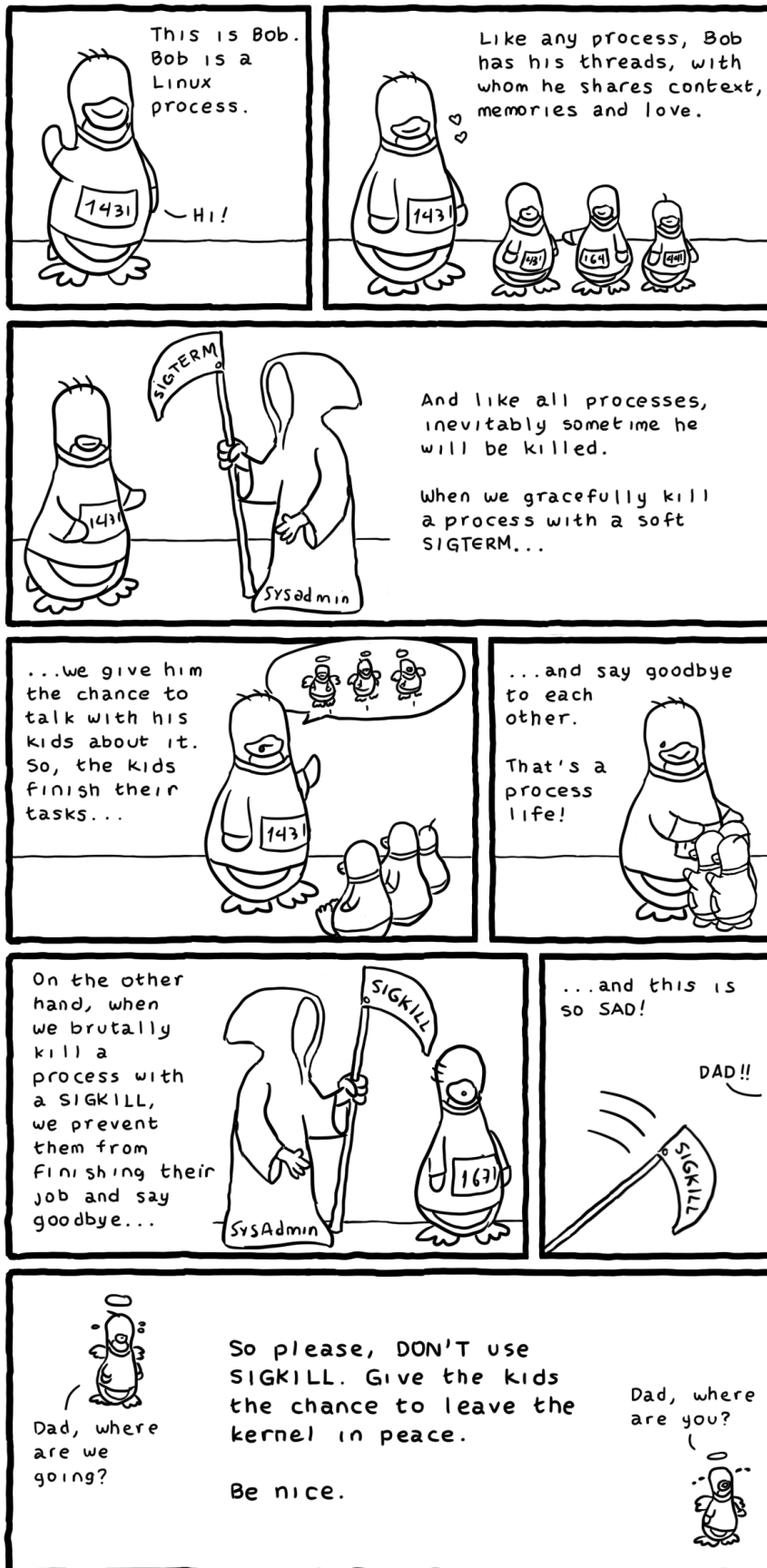
Menjalankan program dengan prioritas (*niceness*)<sup>1</sup> tertentu.

`nice` [OPTION] COMMAND

---

<sup>1</sup> nilai *niceness* antara -20 (prioritas tinggi) sampai 19 (prioritas rendah)





Daniel Stori {turnoff.us}

Gambar 16: Don't SIGKILL

## PROCESS AND JOB CONTROL

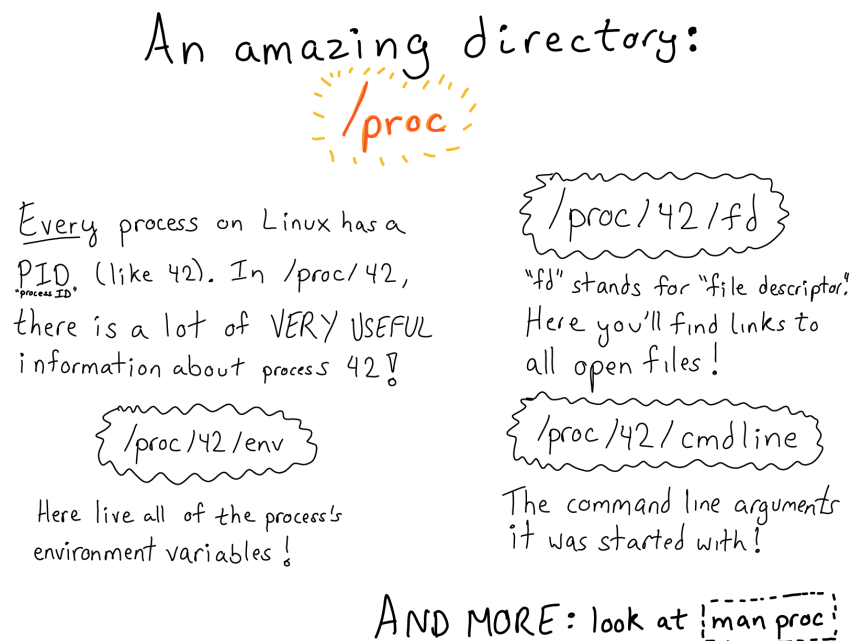
- `-n NICE`: mengeset nilai NICE

*renice*

Mengubah prioritas proses yang sudah berjalan.

`renice [OPTION] PID`

- `-n NICE`: mengubah nilai NICE



Gambar 17: *Process information filesystem*

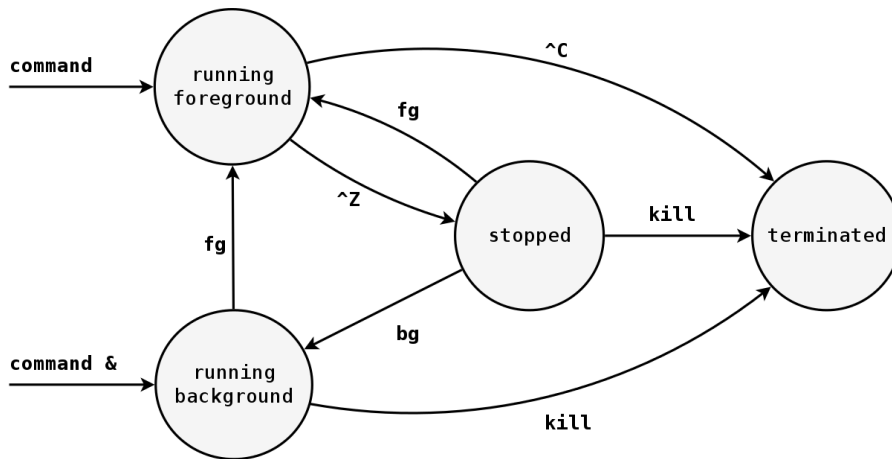
## JOB CONTROL

*Process State*

*Background Process*

Untuk menjalankan proses di *background*, tambahkan tanda `&` pada akhir perintah.

COMMAND `&`

Gambar 18: *Process state**jobs*Menampilkan daftar *job* yang sedang aktif.*jobs**fg*Memindahkan *job* ke *foreground*.*fg* [JOBSPEC]*bg*Memindahkan *job* ke *background*.*bg* [JOBSPEC]



## SHELL SCRIPTING

---

### Shell Scripting

- menyimpan perintah *shell* ke dalam suatu *file*
- fitur pemrograman: variabel, kontrol aliran, fungsi
- berguna untuk:
  - pemrosesan teks
  - otomatisasi administrasi sistem

### *Contoh*

```
$ cat > hello
#!/bin/sh
echo "Hello world"
exit 0
^D
```

```
$ chmod +x hello
```

```
$ ./hello
Hello world
```

### *Referensi*

- [man sh](#)
- <http://wiki.bash-hackers.org>
- <http://www.commandlinefu.com>

## SHELL SCRIPTING

### EKSPANSI

#### Pathname Expansion

- \*: nol atau lebih karakter apapun
- ?: tepat satu karakter apapun
- [...]: tepat satu karakter di dalam *range*
- [!...]: tepat satu karakter selain dalam *range*

#### Pathname Expansion

```
ls /bin/e*
# /bin/echo /bin/ed /bin/egrep

ls /bin/e?
# /bin/ed

ls /bin/[cde]?
# /bin/cp /bin/dd /bin/df /bin/ed

ls /bin/[!a-n]?
# /bin/ps /bin/rm /bin/sh /bin/ss /bin/su
```

#### Parameter Expansion

```
web="cs.ipb.ac.id"

echo $web
# cs.ipb.ac.id

echo ${#web}
# 12

echo ${web%.ac.id}
# cs.ipb

echo ${web#cs.}
# ipb.ac.id
```

#### Command Substitution

```
$(...)
```

Menjalankan perintah dan mengembalikan keluarannya.

```
echo "I am $(whoami)."
# I am root.

echo "Today is $(date +%A)."
# Today is Tuesday.
```

Arithmetic Expansion

`$((...))`

Mengevaluasi ekspresi aritmatika dan mengembalikan keluarannya.

```
x=5
y=3

echo $((x + y)) $((x * y)) $((x / y)) $((x % y))
# 8 15 1 2

echo $((x > y)) $((x == y)) $((x > y && y > 0))
# 1 0 1
```

*Parameter Khusus*

- `$@`: semua parameter
  - `$1`: parameter pertama
  - `$2`: parameter kedua
  - ...
- `$#`: jumlah semua parameter
- `$?`: status keluaran perintah terakhir
- `$$`: PID proses *shell*

EKSPRESI

[

Mengecek *file* dan membandingkan nilai. Status keluaran berupa 0 (*true*) atau 1 (*false*).

[ `EXPR` ]

- `-f FILE`: *file* biasa?

- `-d FILE`: *file* direktori?
- `-r FILE`: *file* bisa dibaca?
- `-w FILE`: *file* bisa ditulis?
- `STR1 = STR2`: kedua *string* sama?
- `INT1 -eq INT2`: kedua angka sama?
- `INT1 -lt INT2`: lebih kecil?
- `INT1 -gt INT2`: lebih besar?

[

```
[ "hello" ]; echo $?
# 0 (true)
```

```
[ -r /etc/passwd ]; echo $?
# 0 (true)
```

```
[ -r /etc/passwd ] && [ ! -w /etc/passwd ]; echo $?
# 0 (true)
```

```
[ "hello" = "world" ]; echo $?
# 1 (false)
```

```
[ 3 -gt 2 ]; echo $?
# 0 (true)
```

*seq*

Mencetak sekuens angka, berguna untuk *looping*.

`seq` [FIRST [INCREMENT]] LAST

```
seq 10
# 1 2 3 4 5 6 7 8 9 10
```

```
seq 1 2 10
# 1 3 5 7 9
```

*read*

Membaca satu baris masukan.

`read` NAME...



```
read input
# <ketikkan: hello>

echo $input
# hello
```

## KONTROL ALIRAN

### *Percabangan*

```
if ...
    then ...
elif ...
    then ...
else
    ...
fi
```

### *Kasus*

```
case WORD in
    PATTERN)
        ... ;;
esac
```

### *Perulangan*

```
for NAME in WORDS
do ...
done
```

### *Perulangan*

```
while ...
do ...
done

until ...
do ...
done
```

*Fungsi*

```
NAME () {
    ...
}
```

## CONTOH

*Percabangan*

```
#!/bin/sh
## create public_html directory if not exist

webdir=~/public_html

if [ ! -d $webdir ]; then
    mkdir $webdir
fi

exit 0
```

*Kasus*

```
#!/bin/sh
## is today weekend?

case $(date +%a) in
    Sat|Sun)
        echo "weekend";;
    *)
        echo "weekday";;
esac
```

*Perulangan*

```
#!/bin/sh
## cube from 1 to 10

for i in $(seq 10); do
    echo $((i*i*i))
done
```

```
exit 0
```

### *Fungsi*

```
#!/bin/sh
## Caesar cipher (ROT13)
```

```
rot13 () {
    tr A-Z N-ZA-M
}
```

```
uppercase () {
    tr a-z A-Z
}
```

```
uppercase | rot13
```

```
exit 0
```

### *Perulangan dan Percabangan*

```
#!/bin/sh
## integer divisible by 3 or 5 between 1-100

count=0
for i in $(seq 100); do
    if [ $((i%3 == 0 || i%5 == 0)) -eq 1 ]; then
        count=$((count + 1))
    fi
done
echo $count
```

## LATIHAN

### *Frekuensi Kata Terbanyak*

Buatlah program “topwords” untuk mencetak 5 kata dengan frekuensi terbanyak dari masukan stdin!

```
$ man ls | ./topwords
21 of
```

```

20 sort
19 by
18 the
16 with

```

### *Contoh pipeline*

1. ubah ke *lowercase*
2. jadikan satu kata satu baris
3. urutkan
4. hitung kemunculan kata yang sama
5. urutkan berdasarkan angka secara menurun
6. ambil 5 baris teratas

### *Identifikasi Penyerang*

Server NCC diserang dari luar melalui *port* SSH. Cek isi *log file* `auth.log.gz`<sup>1</sup> dan identifikasi *n* alamat IP penyerang terbanyak!

```

$ ./top-ip-attack auth.log.gz 3
116.31.116.52    3859
221.194.47.229  1882
221.194.47.208  1819

```

### *Contoh pipeline*

1. tampilkan *file* log dengan `zcat`
2. ambil baris yang mengandung kata `'Failed'`
3. ekstrak alamat IP tiap baris
4. urutkan
5. hitung kemunculan alamat IP yang sama
6. urutkan berdasarkan angka secara menurun
7. ambil *n* baris teratas

### *Cek Tugas Email*

Bantulah asprak SO untuk mengidentifikasi praktikan yang belum mengumpulkan tugas email. Diberikan *file* `auriza.mbox`<sup>2</sup>, ambil nama *user*-nya, kemudian bandingkan dengan daftar *user* pada *file* `passwd`<sup>3</sup>!

<sup>1</sup> <https://lms.ipb.ac.id/mod/folder/view.php?id=28806>

<sup>2</sup> <https://lms.ipb.ac.id/mod/folder/view.php?id=28806>

<sup>3</sup> <https://lms.ipb.ac.id/mod/folder/view.php?id=28806>

```
$ ./check-no-mail
anni
michaeln
...
```

### Contoh pipeline

1. Daftar pengirim email
  1. tampilkan *file* kotak surat
  2. ambil baris yang mengandung kata 'From '
  3. ekstrak nama *user*
  4. urutkan
  5. hilangkan nama yang berulang
2. Daftar semua *user*
  1. tampilkan *file* /etc/passwd
  2. ambil baris yang mengandung kata 'G64'
  3. ekstrak nama *user*
  4. urutkan
3. Cari bedanya
  1. bandingkan keluaran dua *pipeline* di atas dengan `diff`
  2. sesuaikan format sesuai keluaran yang diminta

### Tugas: Spelling Checker<sup>4</sup>

Buatlah program untuk mengecek kesalahan ejaan pada dokumen bahasa Inggris. Gunakan daftar kata pada *file* `words`<sup>5</sup> untuk membandingkan ejaan. Lihat video berikut<sup>6</sup> untuk bantuan *pipeline*-nya. Kumpulkan jawaban di LMS.

```
$ ./myspell sentence
laboratories
prive
timesharing
unix
```

<sup>4</sup> tugas bersifat **opsional**, plagiasi akan mendapat sanksi berat.

<sup>5</sup> /usr/share/dict/words

<sup>6</sup> <https://youtu.be/tc4ROCJYbm0?t=5m58s>



## PERINTAH TAMBAHAN

---

### KOMPRESI

#### *tar*

Menyimpan dan mengekstrak *file* dari arsip *tape/disk*.

`tar` [OPTION] [PATHNAME...]

- `-c`: buat arsip baru
- `-x`: ekstrak *file* dari arsip
- `-f FILE`: gunakan FILE ini
- `-C DIR`: ganti ke direktori DIR
- `-z`: filter `gzip`

#### *gzip*

Mengkompresi *file*.

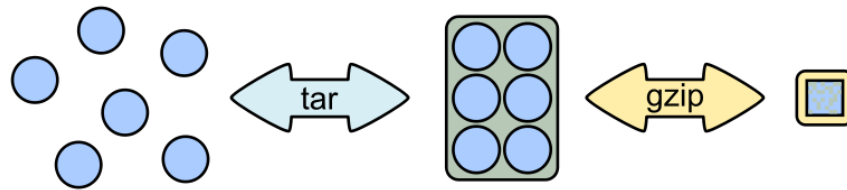
`gzip` [OPTION] [FILE]

#### *gunzip*

Mengekstrak *file*.

`gunzip` [OPTION] [FILE]

*tar.gz*

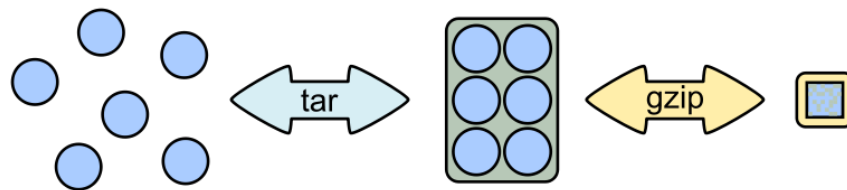


(sumber: <https://commons.wikimedia.org/wiki/File%3ATargzip.svg>)

```
## compress
tar -c "DIR" | gzip > "DIR.tar.gz"

## extract
gunzip < "DIR.tar.gz" | tar -x
```

*tar.gz*



(sumber: <https://commons.wikimedia.org/wiki/File%3ATargzip.svg>)

```
## compress
tar -cz "DIR" -f "DIR.tar.gz"

## extract
tar -xzf "DIR.tar.gz"
```

*zip*

Membungkus dan mengkompresi *file*.

**zip** [OPTION] ZIPFILE FILE...

- -e: enkripsi
- -r: rekursif

*unzip*

Mengekstrak *file* arsip ZIP.



`unzip` [OPTION] ZIPFILE

- `-d DIR`: ekstrak ke direktori DIR

*gpg*

Enkripsi dan tanda tangan digital.

`gpg` [OPTION] [FILE]

- `-e`: enkripsi kunci publik
- `-c`: enkripsi kunci simetris
- `-d`: dekripsi

KONVERSI

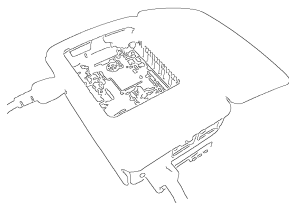
*convert*

Konversi format citra, ukuran, *blur*, *crop*, dan sebagainya.

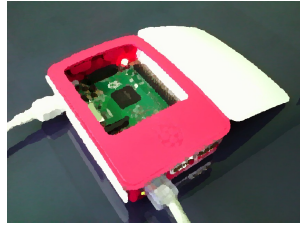
`convert` INFILE [OPTION] OUTFILE

- `-blur GEOM`: mengurangi detail
- `-canny GEOM`: deteksi tepi Canny
- `-equalize`: ekualisasi histogram
- `-negate`: balikan warna
- `-normalize`: normalisasi jangkauan warna
- `-paint RADIUS`: efek lukisan minyak
- `-resize GEOM`: mengubah ukuran

`convert "rpi.jpg" -canny 2x2 -negate "rpip.png"`



`convert "rpi.jpg" -paint 5 "rpip.jpg"`



### *avconv*

Konversi audio dan video.

```
avconv [IN-OPTION] -i INFILE [OUT-OPTION] OUTFILE
```

- `-b:` *bitrate*
- `-f:` *frame rate*
- `-s:` *frame size*
- `-ss:` waktu awal
- `-t:` waktu durasi
- `-vcodec:` *video codec*
- `-qscale:v:` kualitas video (1 = *best*, 31 = *worst*)

### *pandoc*

Konversi Markdown ke format lainnya (HTML, LaTeX, PDF).

```
pandoc [OPTION] [FILE...]
```

- `-s:` *standalone*, dokumen utuh
- `-t FMT:` format keluaran: `hmtl5`, `beamer`, `revealjs`, ...
- `-o FILE:` tulis keluaran ke `FILE`
- `--mathjax:` render persamaan matematis

```
echo "# Heading" | pandoc
## <h1 id="heading">Heading</h1>
```

```
echo "# Heading" | pandoc -t latex
## \section{Heading}\label{heading}
```

```
pandoc -s "file.md" -o "file.html"
```

```
pandoc "file.md" -o "file.pdf"
```

```
pandoc -t beamer "slide.md" -o "slide.pdf"
```

*tesseract*

Konversi citra ke teks (*optical character recognition*).

`tesseract IMAGEFILE OUTFILE`

- `-l LANG`: bahasa yang digunakan (`eng`, `ind`, `ara`, ...)



```
tesseract -l ind "shalat.jpg" stdout
## Jagalah shalat waij dan (terutama) shalat 'Ashr
## Quran . Com/21238
```

```
tesseract -l ara "shalat.jpg" stdout
```

*espeak*

Konversi teks ke suara (*speech synthesizer*).

`espeak [OPTION] [WORDS]`

- `-p INT`: ketebalan suara (0–99)
- `-s INT`: kecepatan kata per menit
- `-v VOICE`: jenis suara (`en`, `id`, `fr`, ...)

*dot*

Konversi teks ke graf.

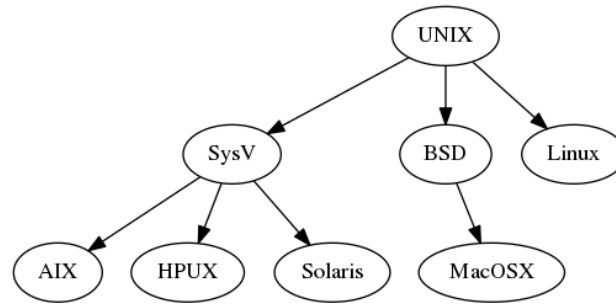
`dot [OPTION] [FILE]`

- `-Tpng`: format keluaran PNG
- `-Tsvg`: format keluaran SVG

```
echo "
digraph unix {
    UNIX -> {SysV BSD Linux};
    SysV -> {AIX HPUX Solaris};
    BSD -> MacOSX;
```

## PERINTAH TAMBAHAN

```
}  
" | dot -Tpng > "unix.png"
```



### *figlet*

Konversi teks ke karakter besar.

```
figlet [OPTION] [MESSAGE]
```

- -c: rata tengah
- -f FONT: pilih jenis *font* (format .flf)
- -w INT: atur lebar keluaran teks

```
figlet "Hello"
```

```
  _  _  _  _  _  
| | | | _ _ | | |  
| | | | / _ \ | | / _ \  
| _ | | _ / | | ( ) |  
| | | | \ _ _ | | \ _ _ /
```

```
figlet -f block "Hello"
```

```
_ | _ | _ | _ |  
_ | _ | _ | _ |  
_ | _ | _ | _ |  
_ | _ | _ | _ |  
_ | _ | _ | _ |
```

## INFO SISTEM

### *lscpu*

Menampilkan informasi tentang arsitektur CPU.

```
lscpu
```

*lshw*

Menampilkan informasi konfigurasi perangkat keras.

`lshw [OPTION]`

- `-short`: format pendek
- `-html`: format HTML

*lspci*

Menampilkan informasi semua perangkat PCI.

`lspci [OPTION]`

- `-k`: tampilkan *driver* kernel yang menangani perangkat
- `-v`: tampilkan informasi detail
- `-vv`: tampilkan informasi lebih detail

*lsusb*

Menampilkan informasi perangkat USB.

`lsusb [OPTION]`

- `-t`: tampilkan pohon hierarki perangkat
- `-v`: tampilkan informasi detail

*lsblk*

Menampilkan informasi perangkat blok.

`lsblk [OPTION]`

- `-m`: tampilkan pemilik dan mode
- `-t`: tampilkan topologi perangkat blok

*lslocks*

Menampilkan daftar kunci yang sedang dipegang.

`lslocks [OPTION]`

- `-p PID`: tampilkan proses ini saja

### *lsof*

Menampilkan daftar *file* yang sedang dibuka.

**lsof** [OPTION] [FILENAME]

- **-p** PID: tampilkan proses ini saja
- **-u** USER: tampilkan proses dari *user* ini saja
- **-i**: tampilkan soket Internet
- **-U**: tampilkan soket UNIX

### *df*

Menampilkan sisa ruang disk.

**df** [OPTION]

- **-a**: tampilkan semua *filesystem*
- **-h**: tampilkan dalam format *human-readable*
- **-i**: tampilkan informasi inode
- **-T**: tampilkan tipe *filesystem*

### *du*

Menampilkan perkiraan penggunaan disk.

**du** [OPTION] [FILE]

- **-d** N: tingkat kedalaman direktori
- **-h**: tampilkan dalam format *human-readable*
- **-s**: tampilkan totalnya saja
- **--inodes**: tampilkan informasi inode

### *free*

Menampilkan besar memori yang kosong dan terpakai.

**free** [OPTION]

- **-h**: tampilkan dalam format *human-readable*

### *vmstat*

Menampilkan statistik *virtual memory*.

`vmstat` [OPTION] [DELAY]

- `-d`: tampilkan statistik disk
- `-s`: tampilkan total statistik memori
- `-w`: mode tampilan lebar
- `-S M`: unit keluaran dalam mega

*bmon*

Monitor *bandwidth* jaringan.

`bmon` [OPTION]

- `-p IF`: tampilkan *network interface* ini saja