GNU/Linux Essentials for HPC Users

Michał Hermanowicz

Interdisciplinary Centre for Mathematical and Computational Modelling (ICM)
University of Warsaw (UW)

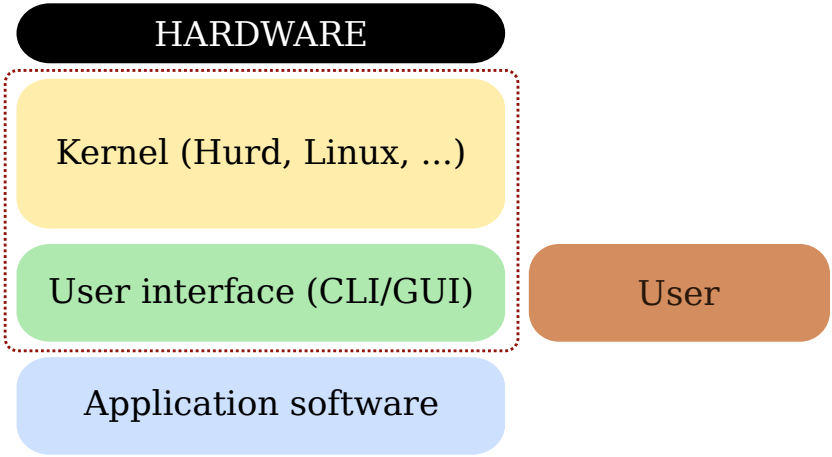
November 19, 2025

Rzeczpospolita Polska

Unia Europejska
Europejski Fundusz Rozwoju Regionalnego

Outline

An operating system: A (very) high-level anatomy



Ken Thompson and Dennis Ritchie (AT&T Bell Labs)

Photo: **Peter Hamer** [CC BY-SA 2.0 (http://creativecommons.org/licenses/by-sa/2.0)], via Wikimedia Commons

UNIX philosophy

Doug McIlroy[1]:

- *write programs that do **one thing** and do it **well**,*
- *write programs to **work together**,*
- *write programs that handle **text** streams as **a universal interface**.*

[1] From: **Peter H. Salus**, *A Quarter-Century of Unix*. Addison-Wesley, 1994. ISBN 0-201-54777-5.

UNIX philosophy

Doug McIlroy[1]:

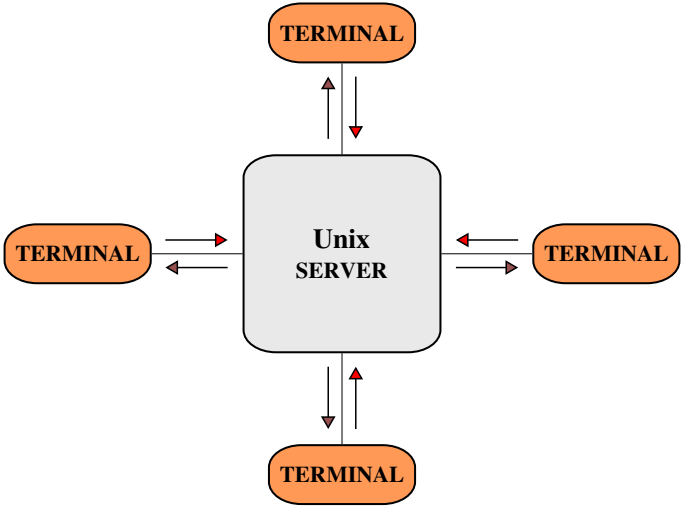
- *write programs that do **one thing** and do it **well***,
- *write programs to **work together***,
- *write programs that handle **text** streams as **a universal interface***.

[1] From: **Peter H. Salus**, *A Quarter-Century of Unix*. Addison-Wesley, 1994. ISBN 0-201-54777-5.

UNIX features:

- multitasking, multiuser (*timesharing* OS),
- hierarchical filesystem,
- *everything is a file* (including devices!),
- networking, written (rewritten) in a high-level language,
- ***pipeline*** mechanism.

UNIX: A multiuser environment



VT100 terminal

Photo: **Jason Scott** [CC BY 2.0 (http://creativecommons.org/licenses/by/2.0)], via Wikimedia Commons

GNU Project and Hurd

GNU Hurd

Recent Changes Preferences *This page:* Edit History Source ?Discussion

What is the GNU Hurd?

The GNU Hurd is the GNU project's replacement for the Unix kernel. It is a collection of servers that run on the Mach microkernel to implement file systems, network protocols, file access control, and other features that are implemented by the Unix kernel or similar kernels (such as Linux). *More detailed*.

What is the mission of the GNU Hurd project?

Our mission is to create a general-purpose kernel suitable for the GNU operating system, which is viable for everyday use, and gives users and programs as much control over their computing environment as possible. *Our mission explained*.

Screenshot: Official website of GNU Hurd (https://www.gnu.org/software/hurd).

Linux (1991)

Date: 25 August 1991
From: torvalds@kruuna.helsinki.fi

Hello everybody out there using minix --

I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing since april, and is starting to get ready. I'd like any feedback on things people like/dislike in minix, as my OS resembles it somewhat (same physical layout of the file-system (due to practical reasons) among other things). (...)

Linus (torvalds@kruuna.helsinki.fi)

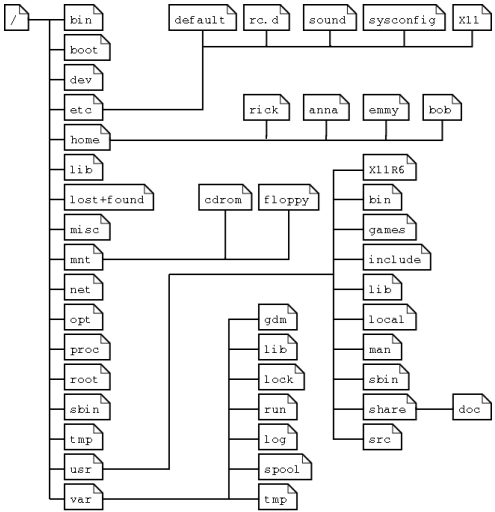
GNU/Linux

Some distributions:

- **Debian** [www.debian.org]
- **Slackware Linux** [www.slackware.com]
- **Arch Linux** [www.archlinux.org]
- **PLD** [www.pld-linux.org]
- **gNewSense** [www.gnewsense.org]
- **Trisquel** [www.trisquel.info]
- **Fedora** [www.getfedora.org]
- **CentOS** [www.centos.org]
- **Scientific Linux** [www.scientificlinux.org]

Graphics: **GNU** (licensed under GFDL 1.3, source: www.gnu.org);
Linux (Tux): Larry Ewing, lewing@isc.tamu.edu, GIMP (www.gimp.org).

Filesystem layout (Diagram: M. Garrels, *Introduction to Linux*, tldp.org)



- `/bin` – *binaries*, programs,
- `/etc` – system configuration files,
- `/dev` – devices,
- `/lib` – libraries (shared/static),
- `/usr` – application software,
- `/mnt` – mount point folders,
- `/home` – user home directories.

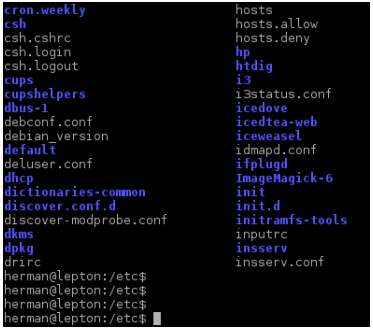
Absolute path:

`/home/student/test/program.c`

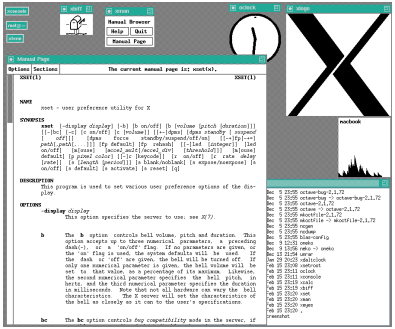
Relative path:

`test/program.c`

GNU/Linux: **User interface**



CLI
(Command Line Interface)



GUI
(Graphical User Interface)

Screenshot (right): Liberal Classic [MIT (http://opensource.org/licenses/mit-license.php)], via Wikimedia Commons

GNU Bash (**B**ourne **A**gain **SH**ell):

interpreter (command language) – **shell**[1].

---

[1] *The GNU Bash Reference Manual*, v. 4.3 (http://www.gnu.org).

GNU/Linux: CLI – **Bash**

GNU Bash (**B**ourne **A**gain **SH**ell):

interpreter (command language) – **shell**[1].

---

[1] *The GNU Bash Reference Manual*, v. 4.3 (http://www.gnu.org).

Moreover:

- default system shell – its **CLI** (Command Line Interface),
- allows interactive and batch execution.

Other shells: `sh`, `csh`, `tcsh`, `ksh`, `zsh`.

GNU/Linux: **CLI**

```
Debian GNU/Linux hpc tty1
hpc login:
```

GNU/Linux: **CLI**

```
Debian GNU/Linux hpc tty1

hpc login: student
```

GNU/Linux: **CLI**

```
Debian GNU/Linux hpc tty1

hpc login: student
Password:
```

GNU/Linux: **CLI**

```
Debian GNU/Linux hpc tty1

hpc login: student
Password:

student@hpc:~$
```

GNU/Linux: **CLI**

```
Debian GNU/Linux hpc tty1

hpc login: student
Password:

student@hpc:~$
student@hpc:~$
```

GNU/Linux: **CLI**

```
Debian GNU/Linux hpc tty1

hpc login: student
Password:

student@hpc:~$
student@hpc:~$
student@hpc:~$
```

GNU/Linux: **CLI**

```
Debian GNU/Linux hpc tty1

hpc login: student
Password:

student@hpc:~$
student@hpc:~$
student@hpc:~$
student@hpc:~$
```

GNU/Linux: **CLI**

```
Debian GNU/Linux hpc tty1

hpc login: student
Password:

student@hpc:~$
student@hpc:~$
student@hpc:~$
student@hpc:~$ pwd
```

GNU/Linux: **CLI**

```
Debian GNU/Linux hpc tty1

hpc login: student
Password:

student@hpc:~$
student@hpc:~$
student@hpc:~$
student@hpc:~$ pwd
/home/student
student@hpc:~$
```

GNU/Linux: **CLI**

```
Debian GNU/Linux hpc tty1

hpc login: student
Password:

student@hpc:~$
student@hpc:~$
student@hpc:~$
student@hpc:~$ pwd
/home/student
student@hpc:~$ ls
```

GNU/Linux: **CLI**

```
Debian GNU/Linux hpc tty1

hpc login: student
Password:

student@hpc:~$
student@hpc:~$
student@hpc:~$
student@hpc:~$ pwd
/home/student
student@hpc:~$ ls
documents test123 task1
student@hpc:~$
```

GNU/Linux: **CLI**

```
Debian GNU/Linux hpc tty1

hpc login: student
Password:

student@hpc:~$
student@hpc:~$
student@hpc:~$
student@hpc:~$ pwd
/home/student
student@hpc:~$ ls
documents test123 task1
student@hpc:~$
```

Running programs – syntax:

```
program [OPTION] [ARGUMENT]
```

Shell variables

Variables:

symbols representing data (numbers, strings).

Shell variables

Variables:

symbols representing data (numbers, strings).

Creating variables:

```
var1=string
variable=1
a=10
```

Shell variables

Variables:

symbols representing data (numbers, strings).

Creating variables:

```
var1=string
variable=1
a=10
```

```
echo $variable
1
```

Shell variables

Variables:

symbols representing data (numbers, strings).

Creating variables:

```
var1=string
variable=1
a=10
```

```
echo $variable
1
```

```
echo $a+1
10+1
```

Shell variables

Environment variables:

are inherited from the parent processes, in contrast to regular (*local*) variables.

Shell variables

Environment variables:

are inherited from the parent processes, in contrast to regular (*local*) variables.

A variable created by:

```
a=1
```

is available within the current terminal session. To make it an environment variable:

```
export a=1
```

or, to *export* previously created local variable:

```
export a
```

Shell variables

Environment variables:

are inherited from the parent processes, in contrast to regular (*local*) variables.

A variable created by:

```
a=1
```

is available within the current terminal session. To make it an environment variable:

```
export a=1
```

or, to *export* previously created local variable:

```
export a
```

Other useful tools: `declare`, `env`.

Shell variables

Examples of the environment variables:

- `$PATH` – holds paths to executables,
- `$PWD` – absolute path to the current working directory,
- `$HOME` – user home directory path,
- `$HOSTNAME` – local host name,
- `$SHELL` – user shell,
- `$USER` – user name.

Shell variables

Examples of the environment variables:

- `$PATH` – holds paths to executables,
- `$PWD` – absolute path to the current working directory,
- `$HOME` – user home directory path,
- `$HOSTNAME` – local host name,
- `$SHELL` – user shell,
- `$USER` – user name.

```
echo $PATH
```

Shell variables

Examples of the environment variables:

- `$PATH` – holds paths to executables,
- `$PWD` – absolute path to the current working directory,
- `$HOME` – user home directory path,
- `$HOSTNAME` – local host name,
- `$SHELL` – user shell,
- `$USER` – user name.

```
echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
```

GNU/Linux: **CLI**

```
student@hpc:~$
```

GNU/Linux: **CLI**

```
student@hpc:~$ ls
```

GNU/Linux: **CLI**

```
student@hpc:~$ ls task1
```

GNU/Linux: **CLI**

```
student@hpc:~$ ls task1
main.c notes.ascii
student@hpc:~$
```

GNU/Linux: **CLI**

```
student@hpc:~$ ls task1
main.c notes.ascii
student@hpc:~$ ls -l task1
```

GNU/Linux: **CLI**

```
student@hpc:~$ ls task1
main.c notes.ascii
student@hpc:~$ ls -l task1
razem 0
-rw-r--r-- 1 student student 0 wrz 1 23:20 main.c
-rw-r--r-- 1 student student 0 wrz 1 23:20 notes.ascii
student@hpc:~$
```

GNU/Linux: **CLI**

```
student@hpc:~$ ls task1
main.c notes.ascii
student@hpc:~$ ls -l task1
razem 0
-rw-r--r-- 1 student student 0 wrz 1 23:20 main.c
-rw-r--r-- 1 student student 0 wrz 1 23:20 notes.ascii
student@hpc:~$ ls -a task1
```

GNU/Linux: **CLI**

```
student@hpc:~$ ls task1
main.c notes.ascii
student@hpc:~$ ls -l task1
razem 0
-rw-r--r-- 1 student student 0 wrz 1 23:20 main.c
-rw-r--r-- 1 student student 0 wrz 1 23:20 notes.ascii
student@hpc:~$ ls -a task1
. .. .x main.c notes.ascii
student@hpc:~$
```

GNU/Linux: **CLI**

```
student@hpc:~$ ls task1
main.c notes.ascii
student@hpc:~$ ls -l task1
razem 0
-rw-r--r-- 1 student student 0 wrz 1 23:20 main.c
-rw-r--r-- 1 student student 0 wrz 1 23:20 notes.ascii
student@hpc:~$ ls -a task1
. .. .x main.c notes.ascii
student@hpc:~$
```

`.x`	hidden file/directory
`..`	parent directory
`.`	current directory
~	home directory

GNU/Linux: **CLI**

```
student@hpc:~$
```

GNU/Linux: **CLI**

```
student@hpc:~$ cd
```

GNU/Linux: **CLI**

```
student@hpc:~$ cd task1
```

GNU/Linux: **CLI**

```
student@hpc:~$ cd task1
student@hpc:~/task1$
```

GNU/Linux: **CLI**

```
student@hpc:~$ cd task1
student@hpc:~/task1$ ls -l main.c
```

GNU/Linux: **CLI**

```
student@hpc:~$ cd task1
student@hpc:~/task1$ ls -l main.c
-rw-r--r-- 1 student student 0 wrz 1 23:20 main.c
student@hpc:~/task1$
```

GNU/Linux: **CLI**

```
student@hpc:~$ cd task1
student@hpc:~/task1$ ls -l main.c
-rw-r--r-- 1 student student 0 wrz 1 23:20 main.c
student@hpc:~/task1$
```

Users belong to groups. Permissions may concern individual users, groups any or other users.

GNU/Linux: **CLI**

```
student@hpc:~$ cd task1
student@hpc:~/task1$ ls -l main.c
-rw-r--r-- 1 student student 0 wrz 1 23:20 main.c
student@hpc:~/task1$
```

Users belong to groups. Permissions may concern individual users, groups any or other users.

Permission syntax – 4 blocks (r – read, w – write, x – execute)

```
-  rwx  rwx  rwx
```

GNU/Linux: **CLI**

```
student@hpc:~$ cd task1
student@hpc:~/task1$ ls -l main.c
-rw-r--r-- 1 student student 0 wrz 1 23:20 main.c
student@hpc:~/task1$
```

Users belong to groups. Permissions may concern individual users, groups any or other users.

Permission syntax – 4 blocks (r – read, w – write, x – execute)

`- rwx rwx rwx`

- 1. block (-): special (file/directory),
- 2. block (`rwx`): file owner,
- 3. block (`rwx`): owner group,
- 4. block (`rwx`): others.

Managing permissions

Specifier:

- `u` (*owner*)
- `g` (*group*)
- `o` (*others*)
- `a` (*all users*)

Managing permissions

Specifier:

- `u` (*owner*)
- `g` (*group*)
- `o` (*others*)
- `a` (*all users*)

Type:

- `r` (4)
- `w` (2)
- `x` (1)

Managing permissions

Specifier:

- `u` (*owner*)
- `g` (*group*)
- `o` (*others*)
- `a` (*all users*)

Type:

- `r` (4)
- `w` (2)
- `x` (1)

Examples:

- `chmod 640 file.txt`
- `chmod 740 file.txt`
- `chown username.group file.txt`

GNU/Linux: **CLI**

```
student@hpc:~/task1$ ls -l main.c
-rw-r--r-- 1 student student 0 wrz 1 23:20 main.c
student@hpc:~/task1$
```

GNU/Linux: **CLI**

```
student@hpc:~/task1$ ls -l main.c
-rw-r--r-- 1 student student 0 wrz 1 23:20 main.c
student@hpc:~/task1$ chmod
```

GNU/Linux: **CLI**

```
student@hpc:~/task1$ ls -l main.c
-rw-r--r-- 1 student student 0 wrz 1 23:20 main.c
student@hpc:~/task1$ chmod -r
```

GNU/Linux: **CLI**

```
student@hpc:~/task1$ ls -l main.c
-rw-r--r-- 1 student student 0 wrz 1 23:20 main.c
student@hpc:~/task1$ chmod -r main.c
```

GNU/Linux: **CLI**

```
student@hpc:~/task1$ ls -l main.c
-rw-r--r-- 1 student student 0 wrz 1 23:20 main.c
student@hpc:~/task1$ chmod -r main.c
student@hpc:~/task1$
```

GNU/Linux: **CLI**

```
student@hpc:~/task1$ ls -l main.c
-rw-r--r-- 1 student student 0 wrz 1 23:20 main.c
student@hpc:~/task1$ chmod -r main.c
student@hpc:~/task1$
```

No message → Done

```
student@hpc:~/task1$
```

GNU/Linux: **CLI**

```
student@hpc:~/task1$ ls -l main.c
-rw-r--r-- 1 student student 0 wrz 1 23:20 main.c
student@hpc:~/task1$ chmod -r main.c
student@hpc:~/task1$
```

No message → Done

```
student@hpc:~/task1$ ls -l main.c
```

GNU/Linux: **CLI**

```
student@hpc:~/task1$ ls -l main.c
-rw-r--r-- 1 student student 0 wrz 1 23:20 main.c
student@hpc:~/task1$ chmod -r main.c
student@hpc:~/task1$
```

No message → Done

```
student@hpc:~/task1$ ls -l main.c
--w------- 1 student student 0 wrz 1 23:20 main.c
student@hpc:~/task1$
```

GNU/Linux: **CLI**

```
student@hpc:~/task1$ ls -l main.c
-rw-r--r-- 1 student student 0 wrz 1 23:20 main.c
student@hpc:~/task1$ chmod -r main.c
student@hpc:~/task1$
```

No message → Done

```
student@hpc:~/task1$ ls -l main.c
--w------- 1 student student 0 wrz 1 23:20 main.c
student@hpc:~/task1$
```

To run scripts:

althought not strictly necessary, we will typically add the execution flag (+x) to the file permissions.

Programming languages: Interpreted vs. Compiled

- **compilation** – the code is translated into machine code by a compiler (C, C++, Fortran),
- **interpretation** – the code is executed by an interpreter (Bash, Python, Perl, PHP).

Programming languages: Interpreted vs. Compiled

- **compilation** – the code is translated into machine code by a compiler (C, C++, Fortran),
- **interpretation** – the code is executed by an interpreter (Bash, Python, Perl, PHP).

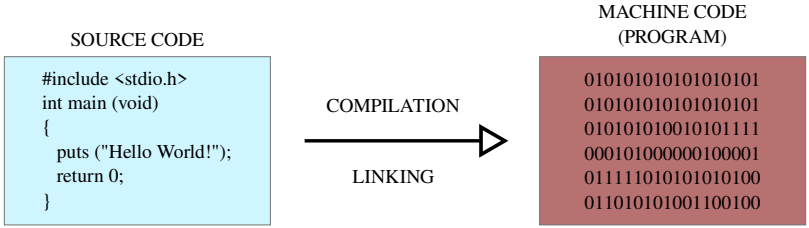
Development tools:

- text editor, compiler/linker/interpreter, debugger,
- optional: IDE (programming environment) typically integrating the abovementioned features (and more).

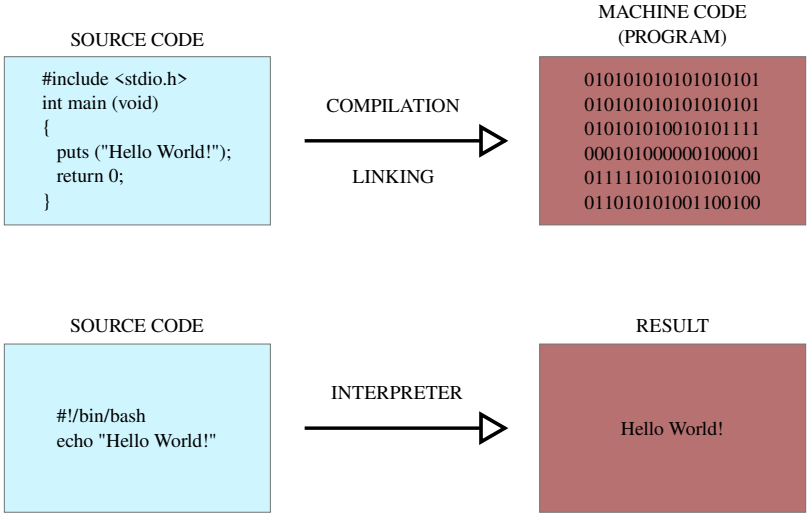
Examples:

- vim, emacs (text editors),
- gcc, gfortran (compilers); gdb (debugger).

Programming languages: Interpreted vs. Compiled



Programming languages: Interpreted vs. Compiled



Source code compilation

```
student@hpc:~$
```

Source code compilation

```
student@hpc:~$ cat hello.c
```

Source code compilation

```
student@hpc:~$ cat hello.c
#include <stdio.h>
int main (void)
{
puts ("Hello World!");
return 0;
}
student@hpc:~$
```

Source code compilation

```
student@hpc:~$ cat hello.c
#include <stdio.h>
int main (void)
{
puts ("Hello World!");
return 0;
}
student@hpc:~$ gcc hello.c
```

Source code compilation

```
student@hpc:~$ cat hello.c
#include <stdio.h>
int main (void)
{
puts ("Hello World!");
return 0;
}
student@hpc:~$ gcc hello.c
student@hpc:~$
```

Source code compilation

```
student@hpc:~$ cat hello.c
#include <stdio.h>
int main (void)
{
puts ("Hello World!");
return 0;
}
student@hpc:~$ gcc hello.c
student@hpc:~$ file a.out
```

Source code compilation

```
student@hpc:~$ cat hello.c
#include <stdio.h>
int main (void)
{
puts ("Hello World!");
return 0;
}
student@hpc:~$ gcc hello.c
student@hpc:~$ file a.out
a.out: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked,
interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32 (...)
student@hpc:~$
```

Source code compilation

```
student@hpc:~$ cat hello.c
#include <stdio.h>
int main (void)
{
puts ("Hello World!");
return 0;
}
student@hpc:~$ gcc hello.c
student@hpc:~$ file a.out
a.out: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked,
interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32 (...)
student@hpc:~$ ./a.out
```

Source code compilation

```
student@hpc:~$ cat hello.c
#include <stdio.h>
int main (void)
{
puts ("Hello World!");
return 0;
}
student@hpc:~$ gcc hello.c
student@hpc:~$ file a.out
a.out: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked,
interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32 (...)
student@hpc:~$ ./a.out
Hello World!
student@hpc:~$
```

Running shell scripts

```
student@hpc:~$ man gcc
```

```
student@hpc:~$
```

Running shell scripts

```
student@hpc:~$ man gcc
```

```
student@hpc:~$ cat hello.sh
```

Running shell scripts

```
student@hpc:~$ man gcc
```

```
student@hpc:~$ cat hello.sh
#!/bin/bash
echo "Hello World!"
student@hpc:~$
```

Running shell scripts

```
student@hpc:~$ man gcc
```

```
student@hpc:~$ cat hello.sh
#!/bin/bash
echo "Hello World!"
student@hpc:~$ file hello.sh
```

Running shell scripts

```
student@hpc:~$ man gcc
```

```
student@hpc:~$ cat hello.sh
#!/bin/bash
echo "Hello World!"
student@hpc:~$ file hello.sh
hello.sh: Bourne-shell script, ASCII text executable
student@hpc:~$
```

Running shell scripts

```
student@hpc:~$ man gcc
```

```
student@hpc:~$ cat hello.sh
#!/bin/bash
echo "Hello World!"
student@hpc:~$ file hello.sh
hello.sh: Bourne-shell script, ASCII text executable
student@hpc:~$ ./hello.sh
```

Running shell scripts

```
student@hpc:~$ man gcc
```

```
student@hpc:~$ cat hello.sh
#!/bin/bash
echo "Hello World!"
student@hpc:~$ file hello.sh
hello.sh: Bourne-shell script, ASCII text executable
student@hpc:~$ ./hello.sh
Hello World!
student@hpc:~$
```

Running shell scripts

```
student@hpc:~$ man gcc
```

```
student@hpc:~$ cat hello.sh
#!/bin/bash
echo "Hello World!"
student@hpc:~$ file hello.sh
hello.sh: Bourne-shell script, ASCII text executable
student@hpc:~$ ./hello.sh
Hello World!
student@hpc:~$ ls -l hello.sh
```

Running shell scripts

```
student@hpc:~$ man gcc
```

```
student@hpc:~$ cat hello.sh
#!/bin/bash
echo "Hello World!"
student@hpc:~$ file hello.sh
hello.sh: Bourne-shell script, ASCII text executable
student@hpc:~$ ./hello.sh
Hello World!
student@hpc:~$ ls -l hello.sh
-rwxr-xr-x 1 student student 33 paź 22 12:13 hello.sh
student@hpc:~$
```

Hashbang – a special line:

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

Running shell scripts

```
$ ./hello.sh
```

(`chmod +x hello.sh`)

```
$ bash ./hello.sh
```

(`-r--------` is enough)

Running shell scripts

```
$ ./hello.sh
```

(`chmod +x hello.sh`)

```
$ bash ./hello.sh
```

(`-r--------` is enough)

Standard output (*stdout*, 1)

```
$ ./hello.sh
Hello World!
```

← standard OUTput

Running shell scripts

```
$ ./hello.sh
```

(`chmod +x hello.sh`)

```
$ bash ./hello.sh
```

(`-r--------` is enough)

Standard output (*stdout*, 1)

```
$ ./hello.sh
Hello World!
```

← standard OUTput

Redirecting *stdout* to file:

```
$ ./hello.sh > file
```

Running shell scripts

```
$ ./hello.sh
```

(`chmod +x hello.sh`)

```
$ bash ./hello.sh
```

(`-r--------` is enough)

Standard output (*stdout*, **1**)

```
$ ./hello.sh
Hello World!
```
← standard OUTput

Redirecting *stdout* to file:

```
$ ./hello.sh > file
```

Alternatively:

```
$ ./hello.sh 1>file
```

Running shell scripts

By analogy, (*stderr* = 2):

```
$ ./hello.sh 2>file_err
```

Running shell scripts

By analogy, (*stderr* = 2):

```
$ ./hello.sh 2>file_err
```

stderr → *stdout*:

```
$ ./hello.sh 2>&1
```

Running shell scripts

By analogy, (*stderr* = 2):

```
$ ./hello.sh 2>file_err
```

stderr → *stdout*:

```
$ ./hello.sh 2>&1
```

stdout and *stderr* to **one** file:

```
$ ./hello.sh &>file
```

Running shell scripts

By analogy, (*stderr* = 2):

```
$ ./hello.sh 2>file_err
```

stderr → *stdout*:

```
$ ./hello.sh 2>&1
```

stdout and *stderr* to **one** file:

```
$ ./hello.sh &>file
```

We can also:

```
$ ./hello.sh &>/dev/null
```

Running shell scripts

By analogy, (*stderr* = 2):

```
$ ./hello.sh 2>file_err
```

stderr → *stdout*:

```
$ ./hello.sh 2>&1
```

stdout and *stderr* to **one** file:

```
$ ./hello.sh &>file
```

We can also:

```
$ ./hello.sh &>/dev/null
```

And:

```
$ ./hello.sh 1>file 2>errors
```

Running shell scripts

/dev/null

is a special file – zero device. Redirected information is irreversibly lost.

Running shell scripts

/dev/null

is a special file – zero device. Redirected information is irreversibly lost.

Everything is a file, including devices, so we can do:

```
$ ./hello.sh > /dev/lpr
```

`/dev/lpr` is a printer. If connected and properly configured, it will print the redirected file.

Running shell scripts

/dev/null

is a special file – zero device. Redirected information is irreversibly lost.

Everything is a file, including devices, so we can do:

```
$ ./hello.sh > /dev/lpr
```

`/dev/lpr` is a printer. If connected and properly configured, it will print the redirected file.

Other devices: scanners, plotters, hard drives, streamers, etc.

Combining tools for text and data processing

INPUT:

- file/stream.

PROCESSING:

- program/skrypt wykonujący operacje na danych wejściowych.

OUTPUT:

- processed data (written to file or stdout).

Pipeline

A *pipeline* (|)

redirects standard output of a program onto the standard input of another program.

(Not particularily good) example with `grep`:

```
$ cat numbers.txt | grep 23
```

Pipeline

A *pipeline* (|)

redirects standard output of a program onto the standard input of another program.

(Not particularily good) example with `grep`:

```
$ cat numbers.txt | grep 23
23
245723
```

Pipeline

A *pipeline* (|)

redirects standard output of a program onto the standard input of another program.

(**Not particularily good**) example with `grep`:

```
$ cat numbers.txt | grep 23
23
245723
```

The `wc` tool (*word count*) – counts words, lines, bytes:

```
$ wc -l numbers.txt
```

(← with `-l` option to count lines)

Pipeline

A *pipeline* (|)

redirects standard output of a program onto the standard input of another program.

(**Not particularily good**) example with `grep`:

```
$ cat numbers.txt | grep 23
23
245723
```

The `wc` tool (*word count*) – counts words, lines, bytes:

```
$ wc -l numbers.txt    (← with -l option to count lines)
5
```

Pipeline

A *pipeline* (|)

redirects standard output of a program onto the standard input of another program.

(**Not particularily good**) example with `grep`:

```
$ cat numbers.txt | grep 23
23
245723
```

The `wc` tool (*word count*) – counts words, lines, bytes:

```
$ wc -l numbers.txt    (← with -l option to count lines)
5
$ cat numbers.txt | grep 23 | wc -l
```

Pipeline

A *pipeline* (|)

redirects standard output of a program onto the standard input of another program.

(Not particularily good) example with `grep`:

```
$ cat numbers.txt | grep 23
23
245723
```

The `wc` tool (*word count*) – counts words, lines, bytes:

```
$ wc -l numbers.txt    (← with -l option to count lines)
5
$ cat numbers.txt | grep 23 | wc -l
2
```

Processing ASCII data

A text file `numbers.txt`:

```
1234  7564  3761  4176  8786
2456  5465  8361  1112  7711
8462  2324  5545  3332  3471
7568  3456  1142  6161  4221
5547  5091  8181  4444  8123
```

Processing ASCII data

A text file `numbers.txt`:

```
1234  7564  3761  4176  8786
2456  5465  8361  1112  7711
8462  2324  5545  3332  3471
7568  3456  1142  6161  4221
5547  5091  8181  4444  8123
```

```
student@hpc:~$
```

Processing ASCII data

A text file `numbers.txt`:

```
1234  7564  3761  4176  8786
2456  5465  8361  1112  7711
8462  2324  5545  3332  3471
7568  3456  1142  6161  4221
5547  5091  8181  4444  8123
```

```
student@hpc:~$ grep 9 numbers.txt
```

Processing ASCII data

A text file `numbers.txt`:

```
1234  7564  3761  4176  8786
2456  5465  8361  1112  7711
8462  2324  5545  3332  3471
7568  3456  1142  6161  4221
5547  5091  8181  4444  8123
```

```
student@hpc:~$ grep 9 numbers.txt
5547  5091  8181  4444  8123
student@hpc:~$
```

Processing ASCII data

A text file `numbers.txt`:

```
1234  7564  3761  4176  8786
2456  5465  8361  1112  7711
8462  2324  5545  3332  3471
7568  3456  1142  6161  4221
5547  5091  8181  4444  8123
```

```
student@hpc:~$ grep 9 numbers.txt
5547  5091  8181  4444  8123
student@hpc:~$ cat numbers.txt | grep 9
```

Processing ASCII data

A text file `numbers.txt`:

```
1234  7564  3761  4176  8786
2456  5465  8361  1112  7711
8462  2324  5545  3332  3471
7568  3456  1142  6161  4221
5547  5091  8181  4444  8123
```

```
student@hpc:~$ grep 9 numbers.txt
5547  5091  8181  4444  8123
student@hpc:~$ cat numbers.txt | grep 9
5547  5091  8181  4444  8123
student@hpc:~$ cat numbers.txt | grep 9 | wc -l
```

Processing ASCII data

A text file `numbers.txt`:

```
1234  7564  3761  4176  8786
2456  5465  8361  1112  7711
8462  2324  5545  3332  3471
7568  3456  1142  6161  4221
5547  5091  8181  4444  8123
```

```
student@hpc:~$ grep 9 numbers.txt
5547  5091  8181  4444  8123
student@hpc:~$ cat numbers.txt | grep 9
5547  5091  8181  4444  8123
student@hpc:~$ cat numbers.txt | grep 9 | wc -l
1
student@hpc:~$
```

HPC architecture

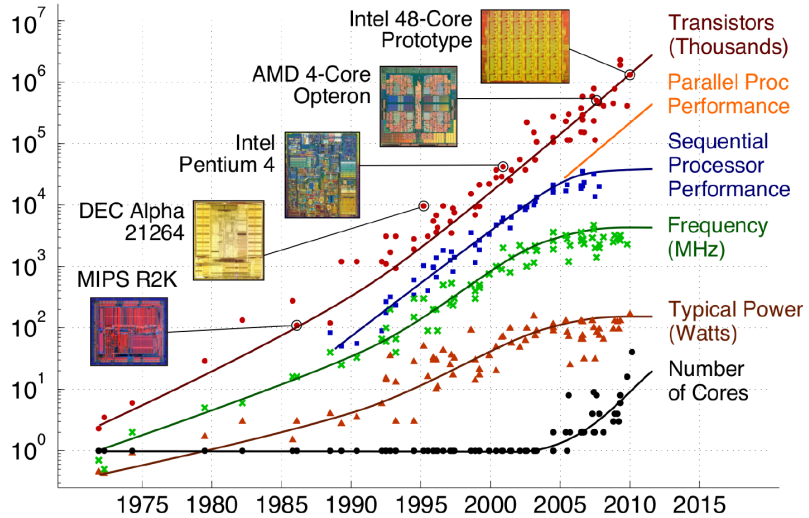
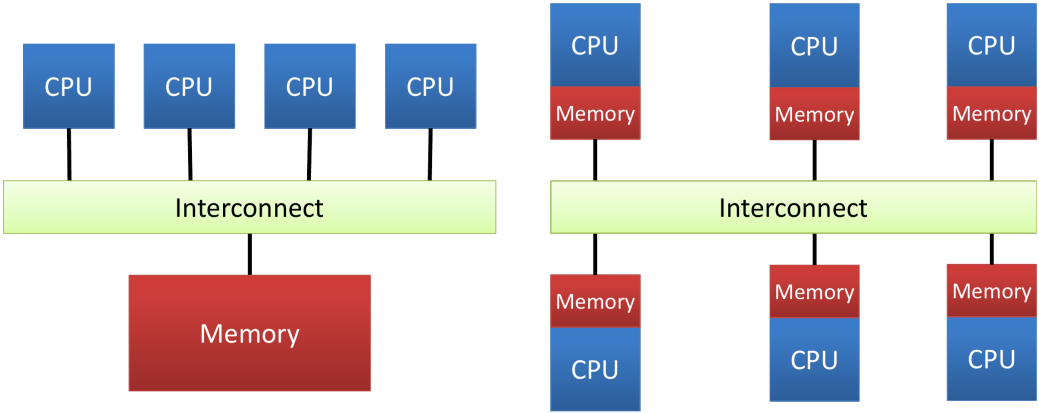


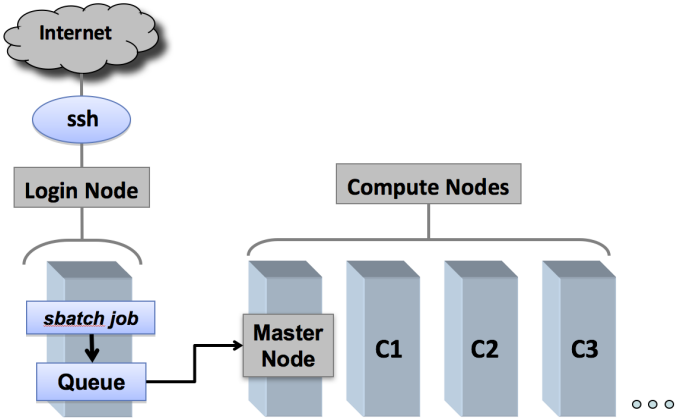
Figure: Christopher Batten, ECE 5950 Complex Digital ASIC Design Course Overview

Shared and distributed memory systems



Źródło: **R. Busseuil** et al. DOI: 10.1007/978-3-642-28566-0_10

HPC infrastructure



Cornell Virtual Workshop (`https://cvw.cac.cornell.edu/environment/slurm_intro`)

Environment Modules

Environment Modules (https://modules.readthedocs.io/en/v4.1.4)

The Modules package is a tool that simplify shell initialization and lets users easily modify their environment during the session with modulefiles.

Environment Modules

Environment Modules (https://modules.readthedocs.io/en/v4.1.4)

The Modules package is a tool that simplify shell initialization and lets users easily modify their environment during the session with modulefiles.

`module avail`

```
apps/abinit/10.0.9
apps/mathematica/14.1
common/hdf5/1.14.3
apps/amber/24
apps/matlab/R2022a
(...)
```

Environment Modules

Environment Modules (https://modules.readthedocs.io/en/v4.1.4)

The Modules package is a tool that simplify shell initialization and lets users easily modify their environment during the session with modulefiles.

```
module avail
```

```
apps/abinit/10.0.9
apps/mathematica/14.1
common/hdf5/1.14.3
apps/amber/24
apps/matlab/R2022a
(...)
```

```
module load ...
module unload ...
module list ...
```

Live Demonstration

→ **SSH Session**