

# Unix-like Operating Systems

## Introduction

Jan Trdlička



Czech Technical University in Prague, Faculty of Information Technology  
Department of Computer Systems

## 1 Unix

- History
- Architecture
- Features

## 2 Shell

- Features
- Command line syntax

## 3 Commands

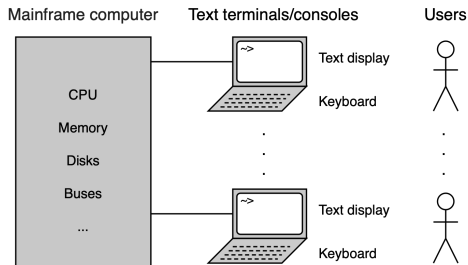
- Filters
- Command execution

## 4 Filesystem

- Directory tree
- Absolute and relative path

## 1969

- First implementation.
- AT&T's Bell Labs (Kenneth Thompson and Dennis Ritchie).



- "Simple hardware"

- Central processing unit (CPU) can only process one instruction stream at a time.
- Only command line interface (CLI).
- No network interface, no remote devices (flash disks,...), ...

## 1972

- New programming language C.
- AT&T's Bell Labs (Dennis Ritchie).

## 1973

- Unix was rewritten in C  $\Rightarrow$  [portability](#).
- Unix was licensed to educational institutions.

## 1977

- Berkeley Software Distribution (BSD)  $\Rightarrow$  new features were added.

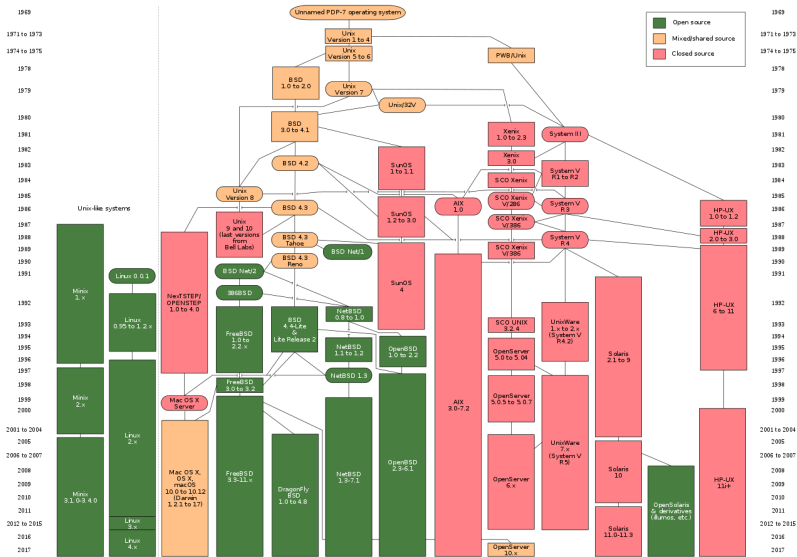
## 1983

- System V Release 4 was commercially the most successful version.
- [GNU project](#)
  - Free UNIX-like operating system (not finished),
  - GNU tools.

## 1991

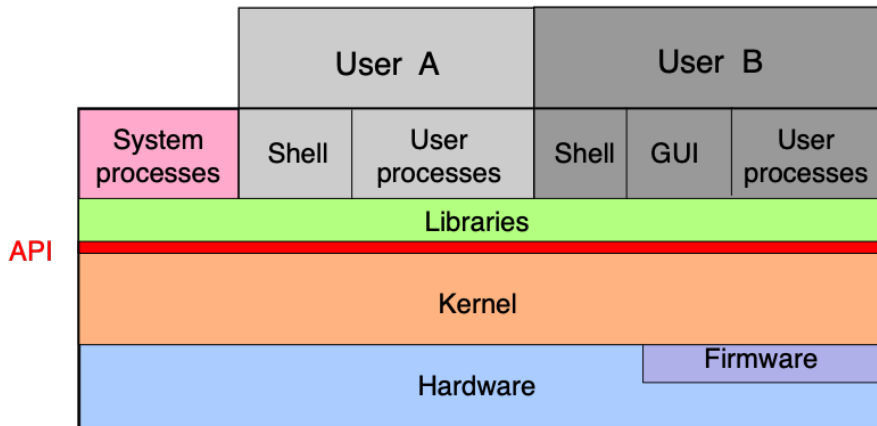
- Linux (Linus Torvalds).

# Unix – history



● Source: Wikipedia

# Unix – architecture



- **Hardware (HW)**

- Physical resources: CPU, RAM, bus, disk, network card, ...
- Firmware (software for hardware testing, kernel loading,...): BIOS, ...

- **Operating system (OS)**

- Kernel and drivers (basic part of OS)
  - Logical resources: users, processes, files, permissions, ...
  - Resource management
  - Application program interface

- **Processes**

- Abstraction of the running program/application
- OS processes
  - Graphical user interface (GUI): GNOME, KDE,...
  - Command-line interface (CLI): shell
  - Command line tools (commands) and other applications
- Other suppliers' processes: web browser, graphic editor, design tools, ...

- **Users/user account**

- Abstraction of physical user for which some attributes are defined
  - User name + password
  - User ID, member of some groups, home directory, login shell, ...

# Unix-like operating systems

- **GNU/Linux**

- Linux kernel
- GNU tools and other tools
- Distributions: [OpenSUSE](#), [Red Hat](#), [Debian](#), ...

```
$> ps -ef
UID          PID    PPID    C  STIME TTY          TIME CMD
gdm           2308     2307    0  17:43 ?           00:00:00 (sd-pam)
honza        7239     6782    0  17:56 pts/0       00:00:00 ps -ef
```

- **Oracle Solaris**

- SunOS kernel + tools (UNIX System V Release 4 + BSD)
- GNU tools can be added (gfind, gsed, gawk, ...)

```
$> ps -ef
      UID      PID  PPID    C   STIME TTY          TIME CMD
      root         0      0     0   Sep 05 ?           0:17  sched
trdlicka 10887   9934     0 17:45:10 pts/27    0:00  ps -ef
```

- **macOS**

- macOS kernel + tools (BSD)
- GNU tools can be added by [Homebrew](#)

```
$> ps -ef
  UID    PID  PPID    C  STIME   TTY          TIME CMD
    0      1     0     0  9:37AM  ??           1:09.65 /sbin/launchd
  501    784    783     0 10:07AM  ttys001      0:00.30 -bash
```



# Unix – features

- **Multiuser OS**

- Multiple users can use Unix at the same time.

- **Processes**

- All non-kernel software is organised into separate, kernel-managed processes.

- **Multitasking OS (time-sharing)**

- Multiple processes can run at the same time.

- **Filesystem**

- Files are stored on disk in a hierarchical file system, with a single top location throughout the system (root, or `" /"`).

- **Command line interface (CLI)**

- Text-based user interface (UI) used to view and manage computer files.
- CLI is implemented by `shell`.

- **Input/output redirection**

- Output from a file or command can be sent (redirected) as input to another file or command.

- **Special files as abstractions of devices and other objects**

- Files in the directory `/dev` represent logical/physical devices.

- **Built-in documentation**

- Manual pages available through command `man`.
- Info about shell builtin commands available through command `help`.
- TextInfo (the GNU Documentation System) available through command `info`.

- **Portability**

- 90% of kernel is written in C.

- **Built-in networking**

- TCP/IP, Network filesystem (NFS), Remote Procedure Call (RPC),...

- **Graphic user interface (GUI)**

- GUI is separated from the Unix kernel and consists of two parts
  - X Window System: interface between kernel and graphic application.
  - Window managers: define the "look and feel" of an X-based GUI (window frames, buttons, ...).

- **Multithreading**

- Multiple threads (instruction streams) can be executed concurrently.

- **Features**

- Interface between user and kernel.
- Command interpreter.

- **Shell implementations** (according to control language syntax)

- **Bourne shell**

- Pascal-like syntax,
- Bourne shell (sh), Korn shell (ksh), Bourne again shell (bash), ...

- **C shell**

- C-like syntax
- C shell (csh), Toronto C shell (tcsh), ...

- **In this course, we will focus on the group of Bourne shells.**

- **Environment settings**

- We can define variables that control system and application behavior.
- The variables are local  $\Rightarrow$  each user has its own environment.
- Can be saved in a configuration file (local to each user)  
 $\Rightarrow$  environment is remembered.

- **Example**

- The system time is absolute (GMT).
- User can define the value of TZ (TimeZone) variable, eg. CET (Central European Time) and all displayed time data will be recalculated.
- When working on a remote server, the user can see the time data (system time, file times, ...) in their local time depending on how the TZ variable is set.

# Example

## Central European Time Zone

```
$> export TZ=CET                                # we define timezone
                                              # (see /usr/share/zoneinfo)

$> date                                          # command date
Tue Sep 24 13:43:58 CEST 2019

$> ls -l /etc/passwd                            # command ls
-rw-r--r-- 1 root root 2058 Sep 23 22:31 /etc/passwd
```

## Japan Time Zone

```
$> export TZ=Japan

$> date
Tue Sep 24 20:43:58 JST 2019

honza@suse100:~> ls -l /etc/passwd
-rw-r--r-- 1 root root 2058 Sep 24 05:31 /etc/passwd
```

- **Interactive mode**

- ❶ The command is defined from the keyboard (default standard input).
- ❷ Command line analysis (find command, meta-character substitution, . . . ).
- ❸ Execution of a command (binary program or script).

- **Batch mode**

- Shell reads commands from a file, called a script.
- It gradually executes (interprets) these commands.
- Script = Unixu commands + control structures (eg. conditional statements, loops, . . . ).

- From the shell point of view, there is no difference between interactive and batch mode.

# Shell – command line syntax

## Variables

- `<prompt> <variable_name>=<value>`
- `<prompt>`
  - Prompt is printed by shell.
  - Value of prompt is defined by the shell variable PS1.
- `<variable_name>`
  - Variable name is identifier.
  - No spaces around symbol `=`.
  - Shell assigns the value to the variable.
- `<value>`
  - By default it is string.
  - If it contains spaces, it must be enclosed in quotation marks.

## Example

```
$> A="abc    123 "  
$> echo "$A "  
abc    123
```

# Shell – command line syntax

## Simple commands

- `<prompt> <command_name> <options> <arguments>`
- `<command_name>`
  - It defines which program will be executed (which).
  - It can be only name or path to the file (relative/absolute).
- `<options>`
  - They can modify the behaviour of command (how).
- `<arguments>`
  - They specify the data to be processed (what).
- **Command name, options and arguments are available**
  - in script by variables `$#, $0, $1, $2, ...`
  - in C program by variables `argc, argv[0], argv[1], ...`

## Example

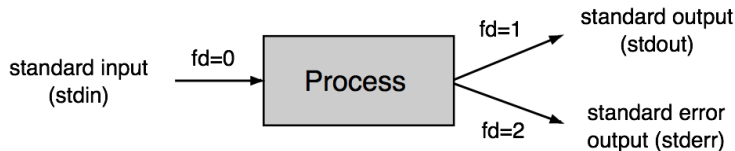
```
$> ls
Desktop  Documents  Downloads  Music  Pictures

$> ls -l /etc/passwd
-rw-r--r-- 1 root root 2058 Sep 23 22:31 /etc/passwd
```



# Filters

- Most of the commands are filters, so that they communicate with the environment through input and output streams.
- Streams are numbered using file descriptors (fd).
- Processes access files using file descriptors.
- Each process has the following descriptors open by default
  - 0 – standard input (keyboard by default)
  - 1 – standard output (terminal by default)
  - 2 – standard error output (terminal by default)



- The default assignment can be redirected.
- The commands can be used in [pipes](#), where the output of one command is redirected like input of next one.

# Command execution

- The commands are executed by the shell.
- The command can be a binary program or a script.
- Command can be executed in several different ways
  - **Foreground execution:** the command is executed and the shell awaits its completion.

```
$> cmd
```

- **Background execution:** the shell does not wait for the command completion and communicates immediately with the user.

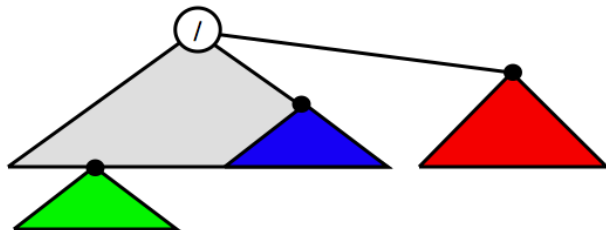
```
$> cmd &
```

- **Sequential execution of multiple commands:** the shell executes the first command, the next one after its termination, etc.

```
$> cmd1 ; cmd2 ; cmd3
```

- **Pipe of multiple commands:** the commands are executed in parallel, stdout of the first command is redirected to stdin of the second command etc.

```
$> cmd1 | cmd2 | cmd3
```



- Data (files) are organised as a **directory tree**.
- Directories can contain files or subdirectories (in Unix, directory is represented as "special" file).
- **Root directory** is represented by symbol **/**.
- Parts of the tree can be mapped to different devices (**disk**, **DVD**, **remote FS**, ...).
- The mapping is transparent to the user.

- It creates a **tree structure** that allows **hierarchical storage of information**.
- **Absolute (complete) path**
  - It always starts in the root directory `/`.
  - Contains a sequence of all directories (separated by `/`) between `/` and the destination file

`/home/stud/smith`

- **Working (current) directory**
  - Can be displayed by command `pwd`.
  - Its value is stored in the shell variable `PWD`.
  - It changes with the command `cd new_working_directory`.
  - It is represented by the absolute path.

## • Relative path

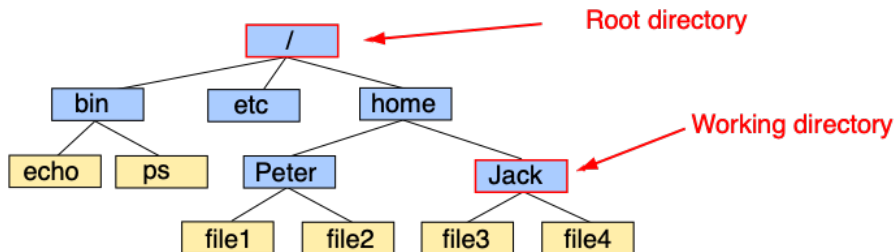
- It always starts in the working (current) \$PWD directory.
- Contains a sequence of subdirectories between \$PWD and the target file.

```
PWD=/home/stud/smith  
../../../../etc
```

## • Home directory

- Defined for each user.
- After login, the working directory is set to home directory.
- Its value is stored in the shell variable `HOME`.
- The user is usually the owner and has the right to write.

# Directories – example



`/home/Peter/file1`

absolute path to the file file1

`../../Peter/file1`

relative path to the file file1

`../Peter/file1`

relative path to the file file1

`/home/Jack/file4`

absolute path to the file file4

`./file4`

relative path to the file file4

`file4`

relative path to the file file4

- **Redirection of input/output**

```
$> date > d.txt           # output of command date  
                          # is redirected to file d.txt
```

```
$> getent passwd | grep "Peter" > p.txt  
                          # info about users with name Peter  
                          # will be save in the file p.txt
```

- **Everything is clear? Is it too simple?**

- Little bit more complicated example?

```
$> echo PID FD EXEC FILENAME; PID=$(pgrep ''); pfiles $PID | awk
'BEGIN{fd=-1;}/^[0-9]/{if(fd>=0){print pid,fd,exec;
fd=-1;};pid=substr($1,0,length($1)-1);exec=$2;}/^[0-9]*:
/{if(fd>=0){print pid,fd,exec;fd=-1;};if
($2=="S_IFREG"){fd=substr($1,0,length($1)-1);}}/^*\//
{fd=-1;}' | while read pid fd exec; do echo $pid $fd $exec
$(echo 0t$pid ::pid2proc \|| ::fd $fd \|| ::print file_t f_vnode
\|| ::vnode2path | mdb -k 2>/dev/null); done
```