



703013 PS Operating Systems (Betriebssysteme) A Short Introduction to Linux/Unix

Philip Salzmann, Alexander Hirsch, Stefan Pedratscher,
Michael Plattner, Markus Reiter, Dennis Sommer

* based on material by Stefan Podlipnig and many others

General Information on UNIX

- ▶ Multi-user OS developed in the 1970s
- ▶ Many derivatives and UNIX-like operating systems
 - ▶ Linux!
- ▶ Many significant properties
 - ▶ hierarchical file systems
 - ▶ single interface for data, device and inter-process communication
 - ▶ background processes
 - ▶ synchronous and asynchronous operation
 - ▶ filter programs (cut, grep, sed, ...)
 - ▶ highly portable

Once Upon a Time...

- ▶ First UNIX version developed 1969 by AT&T (Bell Labs)
 - ▶ Ken Thompson, Dennis Ritchie, Brian Kernighan, Douglas McIlroy, Joe Ossanna
 - ▶ written in assembler code
- ▶ Over time, many different variants formed
 - ▶ BSD-Systems (Univ. California), HP-UX (Hewlett Packard), DG/UX (Data General), AIX (IBM), IRIX (Silicon Graphics), Solaris (Oracle), Mac OS X (Apple), ...
- ▶ Standardization was inevitable

Standardization in and around UNIX

▶ ISO C

- ▶ American National Standards Institute (ANSI)
- ▶ Standard for C Programming Language

▶ Portable Operating System Interface (POSIX)

- ▶ Institute of Electrical and Electronics Engineers (IEEE)
- ▶ Family of standards (system interfaces, threads, shells, ...)
- ▶ POSIX is based on UNIX but not limited to it
 - ▶ POSIX also supported by other operating systems

▶ Single UNIX Specification

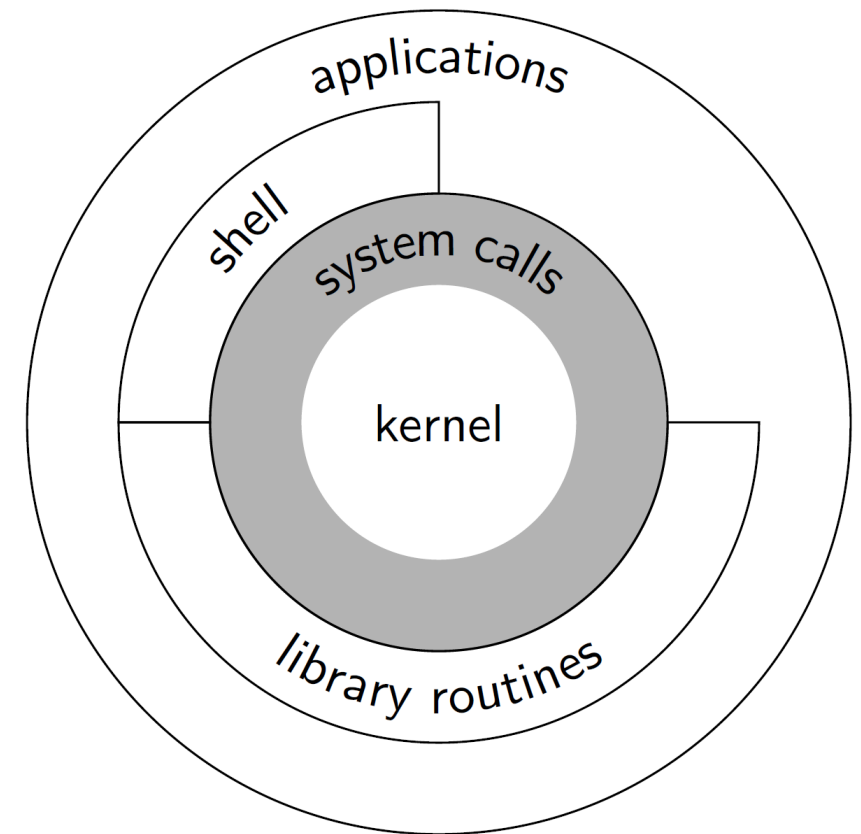
- ▶ required to use “UNIX” trademark, built around POSIX
- ▶ but very few BSD or LINUX systems submitted for compliance

UNIX Implementations

- ▶ Mostly done by companies
- ▶ Three major players
 - ▶ System V Release 4 (SVR 4)
 - ▶ Unix System Laboratories (USL), AT&T
 - ▶ SVR 4 meets POSIX standard requirements
 - ▶ Covers large class of UNIX derivatives (e.g. Solaris)
 - ▶ Berkley Software Distribution (BSD)
 - ▶ University of California at Berkeley (UCB)
 - ▶ Covers large class of UNIX derivatives (e.g. FreeBSD, Mac OS X)
 - ▶ Linux
 - ▶ public domain (GNU license)
 - ▶ Holds features of SVR 4, POSIX and BSD family

UNIX Architecture

- ▶ **System core**
 - ▶ interface for direct hardware access (e.g. peripheral devices, memory, CPU)
- ▶ **System calls**
 - ▶ interface to the core
- ▶ **Library calls**
 - ▶ frequently-used features (e.g. printf in stdio.h of libc)
- ▶ **Shell**
 - ▶ interface for executing applications
- ▶ **Applications**



UNIX Shell

- ▶ Main interface to the system
- ▶ Is an interpreter
 - ▶ takes a command
 - ▶ interprets it
 - ▶ executes it
 - ▶ waits for the next command
- ▶ Standard composition of commands
 - ▶ `command [flags...] [files...]`
 - ▶ e.g. `ls -l start1.txt start2.txt`
 - ▶ e.g. `cd /tmp`
 - ▶ e.g. `pwd`

The Single Most Important Command: man

- ▶ Offers access to the manpages (documentation for each command and its options)
- ▶ `man <command>`
 - ▶ Examples
 - ▶ `man ls` – shows the documentation for the command `ls` (in section 1)
 - ▶ `man 5 config-file` – shows the documentation for `config-file` in section 5
 - ▶ `man -k query` – lists the manpages that contain `query`
 - ▶ `man man` – shows the manpage for `man`
 - ▶ `man intro` – shows an introduction to user commands
- ▶ A manpage usually exceeds the available space on the screen
 - ▶ navigation keys plus shortcuts: **d** (half page down), **u** (half page up), ...
 - ▶ hit **/** to search within the man page, use **n** and **N** to navigate between results
 - ▶ hit **q** to quit the man page

My \$HOME is my Castle

- ▶ When starting a shell, the current directory is the home directory
- ▶ `pwd` – prints the current working directory
 - ▶ e.g. `/home/foo`
- ▶ Differences with regard to Windows
 - ▶ directories are separated by a slash (/), not a backslash (\)
 - ▶ there are no drive letters, everything is located in the root directory /
 - ▶ Linux/UNIX is case sensitive
 - ▶ file endings (e.g. `.txt`, `.pdf`) are often omitted
 - ▶ e.g. `.gitignore`, `filewithnoending`
 - ▶ use `file <filename>` command to determine its type based on its contents
 - ▶ executable files often have no ending

UNIX File System (1/2)

- ▶ Hierarchical tree structure
- ▶ The root is simply /
- ▶ Individual file path components are separated by /
- ▶ Each component is a directory (or at the end a file)
 - ▶ e.g. /usr/sbin/bzip2
 - ▶ where usr and/sbin are directories and bzip2 is a file
- ▶ Access is controlled via permissions
 - ▶ read
 - ▶ write
 - ▶ execute

UNIX File System (2/2)

▶ Important commands

- ▶ `pwd` – show the current working directory
- ▶ `cd` – change directory (if no argument given → changes to `$HOME`)
- ▶ `mkdir, rmdir` – make or remove directories

▶ Absolute and relative paths are possible

- ▶ absolute paths start with `/` and start from the root
 - ▶ `cd /etc/init.d`
- ▶ relative paths do not start with `/` and start from the current directory
 - ▶ `cd foo/bar`

▶ Special placeholders

- ▶ `.` is the current directory
- ▶ `..` is the parent directory
- ▶ `$HOME` or `~` is the home directory of the current user

Working With Files (1/3)

- ▶ `ls` – list the contents of a directory
- ▶ `ls -l` – list the contents using the “long” format (including permissions)
- ▶ `file` – show type information regarding a file

- ▶ Wildcards
 - ▶ `?` represents a single character
 - ▶ `*` represents 0 or multiple characters
 - ▶ `[b-d]` represents b, c, or d
 - ▶ `{conf,loc}` represents either `conf` or `loc`
 - ▶ note: wildcards are interpreted by the shell, not the command (e.g. executable) itself

Working With Files (2/3): Wildcard Examples

► Examples

- files in the current directory:

date, out1, out2, out3, outer, prog.f, prog.o

Pattern	Match
out?	out1, out2, out3
prog.[fo]	prog.f, prog.o
*	date, out1, out2, out3, outer, prog.f, prog.o
*.f	prog.f
out*	out1, out2, out3, outer

Working With Files (3/3)

- ▶ `cat` – print the contents of a file on the screen
- ▶ `less` – does the same, but page by page
- ▶ Editors
 - ▶ `nano`
 - ▶ `vi, vim`
 - ▶ `emacs`
 - ▶ `kate`
 - ▶ `gedit`
 - ▶ ...
- ▶ Display strings directly via `echo`
 - ▶ e.g. `echo "Hello World"`

Input, Output, and Redirections

- ▶ `stdin` – standard input (your keyboard, by default)
- ▶ `stdout` – standard output (your screen, by default)
- ▶ `stderr` – standard error (your screen, by default)
- ▶ Redirection
 - ▶ `>, <, 2>` redirects `stdout`, `stdin`, and `stderr` respectively (to a file)
 - ▶ `2>&1` redirects `stderr` to `stdout`
 - ▶ `A | B` redirects `stdout` of A to `stdin` of B (unnamed pipe)
- ▶ Examples
 - ▶ `ls -l > ls.txt` (redirects `ls -l` to `ls.txt` – will overwrite if present)
 - ▶ `ls -l >> ls.txt` (redirects `ls -l` to `ls.txt` – will append if present)
 - ▶ `ls ~/nope 2> err.txt` (redirects error messages)
 - ▶ `ls test1.txt nope > out.txt 2>&1`
(`test1` exists, `nope` does not exist, but output is in `out.txt`)
 - ▶ `ls -l | grep txt` (output of `ls` handed to `grep`, only shows entries containing `txt`)

Copy, Move, Rename, and Delete Files

- ▶ `cp` – copies files
 - ▶ `cp -r` also copies directories
- ▶ `rm` – removes files
 - ▶ Note: `rm -r` removes files recursively (e.g. a directory and its contents)
 - ▶ Note: `rm -rf` removes files recursively **without asking** – pay attention!
- ▶ `touch filename`
 - ▶ if `filename` does not exist, create it (will be empty)
 - ▶ if `filename` does exist, update its timestamp
- ▶ `mv` – moves or renames files

Connecting and Reusing Commands (1/2)

- ▶ Commands can be successful, or they can fail
 - ▶ Successful commands return **exit code** = 0
 - ▶ Failed commands return exit code $\neq 0$
- ▶ Multiple commands can be issued at once
 - ▶ `A ; B` will first execute A and then B (always)
 - ▶ `A && B` will first execute A and then B if A was executed successfully
 - ▶ `A || B` will first execute A and then B if A was **not** executed successfully
 - ▶ `{ ... }` groups multiple commands, returns exit status of last command
 - ▶ `{ A || B; } && C` vs
 - ▶ `A || { B && C; }` (note the required trailing semicolon)

Connecting and Reusing Commands (2/2)

- ▶ Shell allows access to history of commands
 - ▶ `history` – shows a log of all commands, numbered
 - ▶ `!!` – re-executes the last command
 - ▶ `!-2` – re-executes the second last command
 - ▶ `!<number>` – re-executes the x-th last command
 - ▶ `!xt` – re-executes the last command that starts with xt

Additional Commands

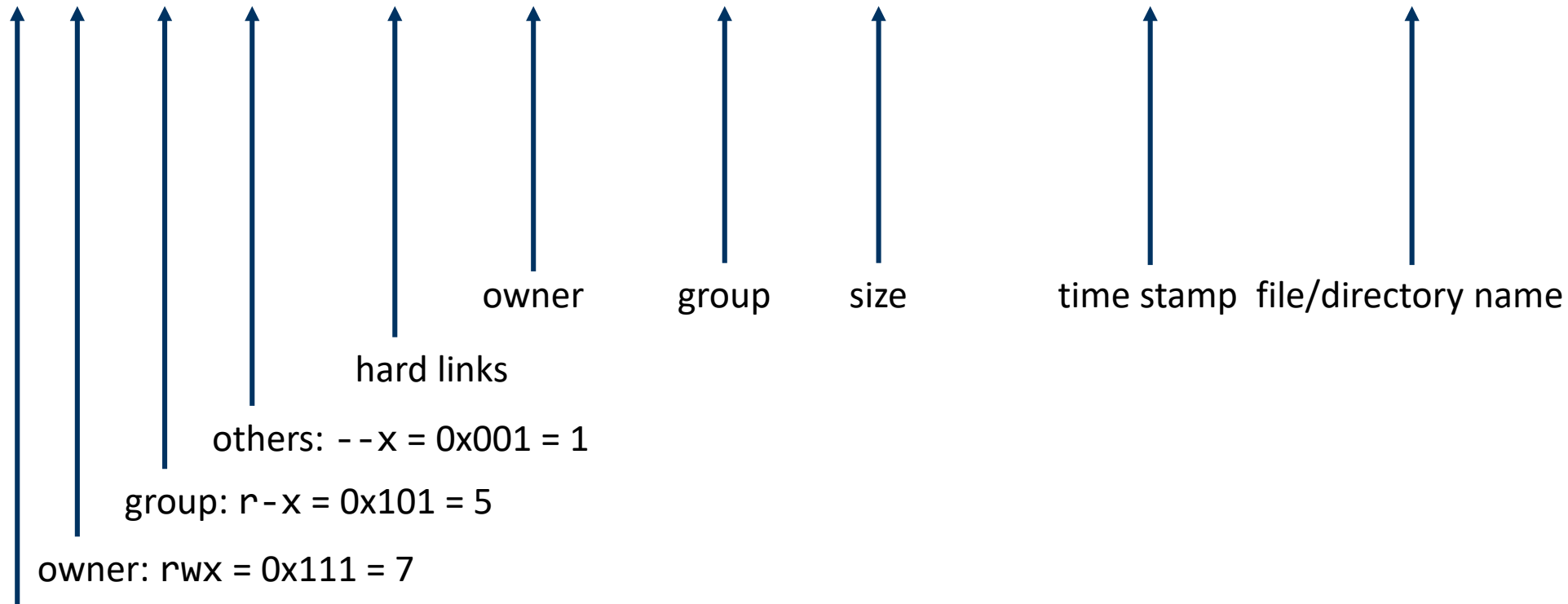
- ▶ `info` – similar to `man` but with cross-references
- ▶ `apropos` – search manpage names and descriptions
- ▶ `find` – search for files
- ▶ `head, tail` – show the first or last few lines of a file
- ▶ `sort` – sort lines
- ▶ `grep` – show or hide lines matching a pattern
- ▶ `date` – set/show the system date and time
- ▶ `hostname` – set/show the name of the computer
- ▶ `wc` – count lines, words and bytes
- ▶ `xargs` – build and execute commands from standard input

Access Permissions

- ▶ Bitmask with 9 bits
- ▶ 3 bits each for access permissions (read, write, execute) for each of the three user classes (owner, group, others)
- ▶ 3 types of permissions, applicable to both files and directories
 - ▶ **read (r)**: file: may be read; directory: content may be listed but not the access permissions
 - ▶ **write (w)**: file: may be written; directory: may create, move, and remove files and directories within
 - ▶ **execute (x)**: file: may be executed; directory: may change into this directory
- ▶ Additionally: special permissions
- ▶ Important commands
 - ▶ `chmod` – change access permissions
 - ▶ `chown` – change ownership

Output of `ls -l`

```
-rw-r--r--. 1 philipp dps      0 Mar 18 20:58 bar
drwxr-xr-x. 2 philipp dps 4096 Mar 18 20:58 foo
```



file type (`-` for files, `d` for directories, `l` for symbolic links, `c` for device file)

Processes

- ▶ Programs consist of one or more processes (usually one)
- ▶ Each process gets a unique ID upon start (called a PID)
- ▶ `ps` – shows a selection of currently present processes
 - ▶ `ps -e` – shows all processes
 - ▶ `ps aux` – shows all processes with additional information (who executes it, CPU time thus far, ...)
- ▶ `ps tree` – shows process hierarchy (parents and children – tree structure)
 - ▶ root is called `init`, started upon boot up
- ▶ `top` – same information as `ps`, but interactive
- ▶ `htop` – modern version of `top`

Processes in the Shell

- ▶ `&` – starts process in the background
- ▶ `Ctrl-Z` – suspend the current foreground process
- ▶ `fg` – resume job in the foreground
- ▶ `bg` – resume suspended job in the background
- ▶ `jobs` – show all suspended and background processes
- ▶ `Ctrl-C` – stop the current foreground process (SIGINT signal)
- ▶ `nice` – set process priorities

Stopping and Killing Processes

- ▶ **kill** – sends a signal to a process
 - ▶ `kill -TERM 3333` – stops process with PID 3333 with SIGTERM
 - ▶ `kill -9 3333` – kills process with SIGKILL (cannot be caught by process)
 - ▶ `kill -9 -1` – kill all processes
 - ▶ `kill -l` – shows all signals

System Status

- ▶ `df` – shows disk space usage
- ▶ `du` – shows file/directory space usage
- ▶ `vmstat` – shows system statistics
 - ▶ `vmstat 1` – continuously shows current state every second
- ▶ `/proc` – holds files that report system state and information
 - ▶ `e.g. cat /proc/cpuinfo` – holds CPU model information
 - ▶ `e.g. cat /proc/[PID]` – holds information about process [PID]

Shell Variables

- ▶ Shell supports variables
- ▶ Values can be assigned and tested for arbitrarily
- ▶ **No type system!**
- ▶ Example
 - ▶ `HELLO_VAR="Hello World"` – set `HELLO_VAR` to value “Hello World”
 - ▶ `export HELLO_VAR` – make `HELLO_VAR` available to subsequently executed processes
 - ▶ `echo $HELLO_VAR` – print the content of `HELLO_VAR`
- ▶ Variable names can contain letters (case sensitive!), numbers and underscore, but must start with a letter
- ▶ `unset` – deletes variables (undo `export`)

Special Shell Variables

- ▶ `$HOME` – home directory
- ▶ `$PATH` – executable search path
(e.g., call `bash` directly instead of `/usr/bin/bash`)
- ▶ `$PWD` – current directory
- ▶ `$USER` – current user name
- ▶ `$SHELL` – current shell
- ▶ `$$` – process number of the current shell
- ▶ `$?` – exit status of the last command

Quoting

- ▶ Shell can interpret many special characters
 - ▶ e.g. *, ?, !, [,], &, ...
- ▶ They can also be used as normal characters (= will not be interpreted) if quoted or escaped properly
 - ▶ ‘ – single quote, will escape everything until next ‘
 - ▶ “ – double quote, will escape everything until next “ except for \, \$ and ` (back ticks)
 - ▶ \ - will escape the subsequent character

Special Characters Summary

- ▶ `;` – separator for multiple commands
- ▶ `&` – start process in the background
- ▶ `()` – group a command and create a sub-shell
- ▶ `|` – pipe (for streaming/redirection)
- ▶ `< > &` – redirection symbols
- ▶ `* ? [] ~ + - @ !` – meta characters for file names
- ▶ ``pwd`` – back ticks (pwd will be executed and substituted for result)
- ▶ `$` – variable substitution
- ▶ `[newline] [space] [tab]` – word separators

Command Substitution

- ▶ ``...`` – back tick/accent grave/grave accent – NOT a single quote
 - ▶ e.g. ``pwd``
- ▶ `$(...)` – equivalent (and **preferred!**) way of doing it
- ▶ command in between will be executed
- ▶ stdout of command will be returned
- ▶ Examples
 - ▶ `VAR=pwd` – assigns the string (!) value “pwd” to variable VAR
 - ▶ `echo $VAR` → “pwd”
 - ▶ `VAR=$(pwd)` – assigns the string value of the current directory to variable VAR
 - ▶ `echo $VAR` → /home/...

Concluding Advice

- ▶ Check man pages and tutorials/explanations on the Internet – there are tons of really good ones (e.g. StackOverflow)
- ▶ Ensure proper quoting / escaping when using special characters
- ▶ Be exact and consider special cases
 - ▶ e.g. `rm -rf /$VARIABLE` – if `$VARIABLE` is empty, this is equal to `rm -rf /`
 - ▶ e.g. `rm -rf /usr /lib/nvidia-current/xorg/xorg`



accidental whitespace, will remove all of /usr

- ▶ Further reading:
<https://github.com/ketancmaheshwari/lisa19>