Faculty of Mathematics and Physics
CHARLES UNIVERSITY

NOFY077

Introduction to the Linux OS

Peter Huszár
KFA: Department of Atmospheric Physics

Pavel Řezníček
ÚČJF: Institute of particle and nuclear physics

September 30, 2025

Overview and Organization

Introduction to the Operation system Linux, focus on the command line, scripting, basic services and tools used in (not only) physics: tasks automation in data processing and modeling

Organization

- Graded Assessment (KZ): attendance to the lectures, worked out homeworks

Literature

- C. Herborth: Unix a Linux - Názorný průvodce, Computer Press, Praha, 2006
- D. J. Barrett: Linux - Kapesní přehled, Computer Press, Praha, 2006
- M. Sobell: Mistrovství v RedHat a Fedora Linux, Computer Press, Praha, 2006
- M. Sobell: Linux - praktický průvodce, Computer Press, Praha, 2002
- E. Siever: Linux v kostce, Computer Press, Praha, 1999
- Number of online sources...

Study materials and homeworks

- http://kfa.mff.cuni.cz/linux

KFA MFF UK

ÚČJF MFF UK

Syllabus

1. UNIX systems, history, installation, basic applications
2. Structure of the Linux OS, file systems, hierarchy of the file system
3. Command line, shells, remote access (ssh, ftp)
4. Processes and their administration, basic system commands, packages, printing
5. Users, file and directory permissions
6. Work with files and directories, file compression, links, partition
7. Text-file processing commands, redirection, pipeline
8. Regular expressions
9. Command line based text editors
10. User and system variables, output processing
11. Scripts: basic construction, conditionals, loops, functions, automation
12. Networking, server-client services: http, (s)ftp, scp, ssh, sshfs, nfs
13. Programming in Linux (examples of Fortran, C/C++, Python), version control systems, documents in Latex

Motivation for Linux

Operating systems, two basic types:

1. **Windows**
2. **UNIX** (**Mac** and **Android** are also UNIX-based)

- Windows historically more focused on user-PCs
- UNIX systems used on servers (mail, web, computing, networking)
- Nowadays the abilities of both systems are close each to other

Why Linux ?

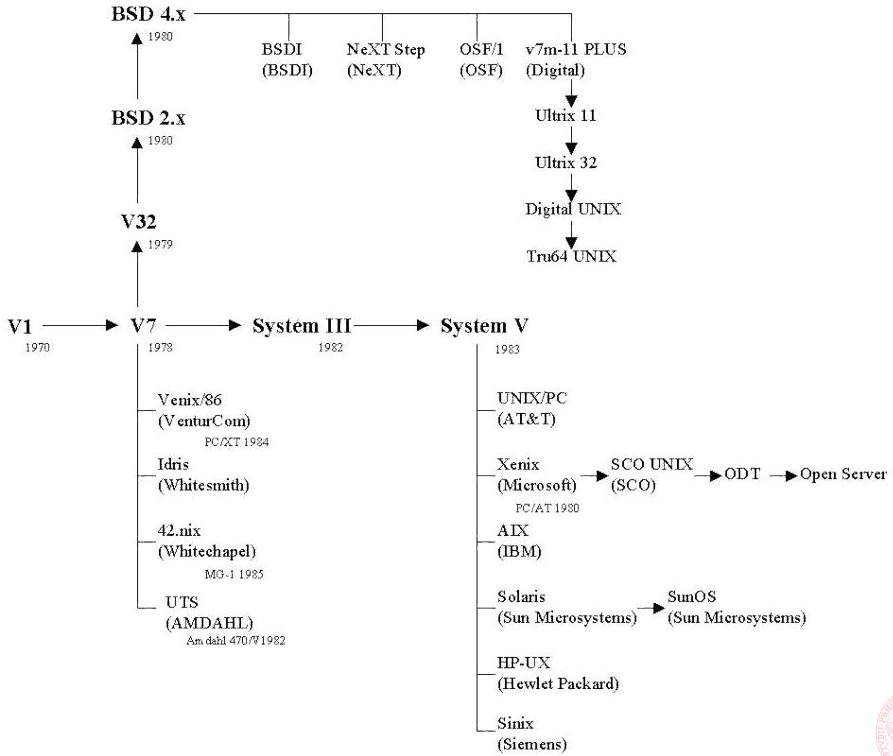
- Powerful command-line (*cmd*)
  - Applications started from *cmd*
  - System control via *cmd*
  - Programs control, compilation, modifications and debugging via *cmd*
  - Scripting / programming using shell
- Natural remote access (including graphic windows), remote administration
- Open source & free applications, support through wide-community

UNIX - trademark of operating system created in Bell Laboratories of AT&T company in 1965

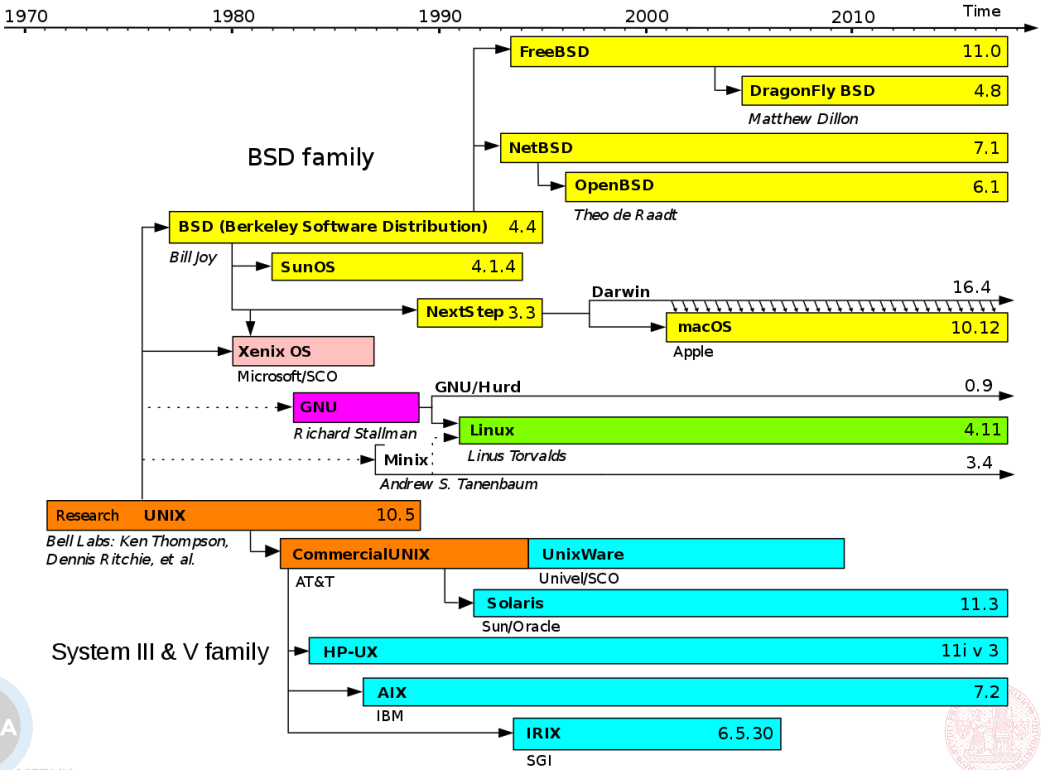
Path to UNIX

- **1964**: Bell Telephone Laboratories, General Electric and MIT develops OS Multics (MULTIplexed Information and Computing System)
  - Unfinished
  - *Kenn Thompson, Dennis Ritchie, Brian Kernighan*
- **1969**: Bell Labs. withdraws from Multics project, *Thompson* writes basic OS (kernel, shell), editor and assembler for the PDP-7 computer
- **1970**: *Kernighan* suggests the name of the OS UNIX (firstly UNICS)
- **1971**: *Thompson* asks for new PDP-11 computer for further development (rejected); *Thompson* pretends development of automatized office $\rightarrow$ computer assigned for text processing
- **1973**: *Thompson* rewrites Fortran language - languages B, *Richie* rewrites UNIX into C language for better portability
- **1978**: UNIX v7 released for Universities (Berkeley) - UNIX divided into two parts:
  - AT&T (System III, System V)
  - Berkeley (BSD 3.0)

UNIX Systems



UNIX Systems



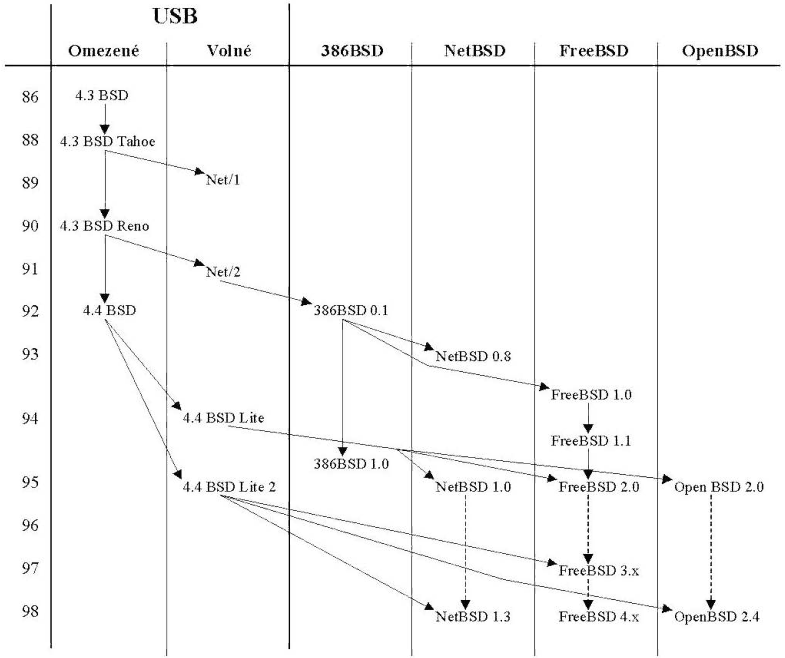
Present UNIX Systems

- SUN: Sun OS, Solaris
- Silicon Graphics: Irix
- DEC: Ultrix, Digital Unix
- IBM: AIX
- HP: HP-UX
- Siemens Nixdorf: SINIX
- Novell: UNIXware
- SCO: SCO Unix
- FreeBSD, NetBSD, OpenBSD, ...
- **Linux**, Mac OS X
- Android OS

Licensing

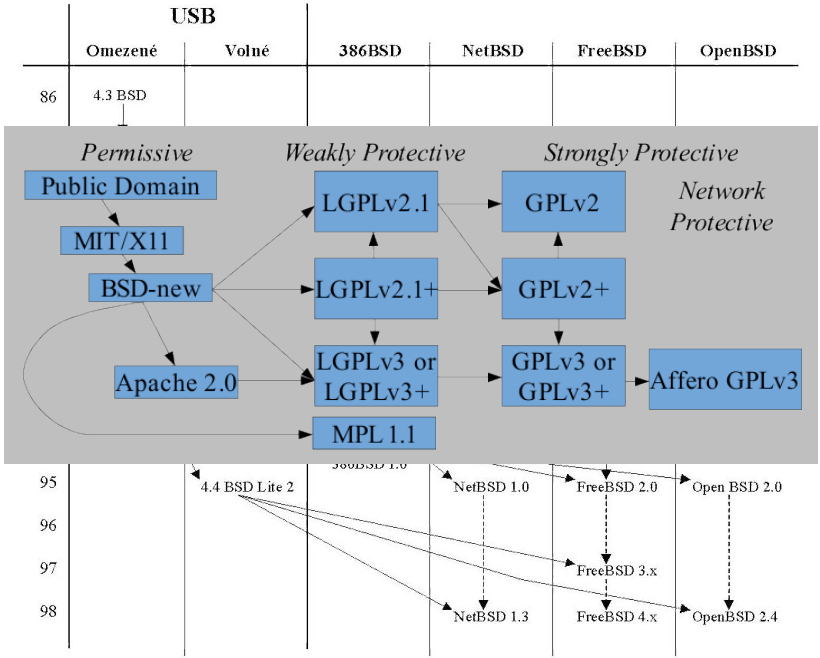
Non-free source codes → release of open and free versions

- Development through free versions



Non-free source codes → release of open and free versions

- Development through free versions



Linux foundation

- **1984**: *Richard Stallman* creating GNU (GNU's not UNIX)
  - Attempt to create free (license) system
  - Creating General Public License (GPL)
  - *Freedom to run, study, share and modify the software*
- **1991**: *Linus Torvalds* trying to create freely available system
  - Based on Minix
  - Writing kernel of the system

GNU Linux - Linux kernel, tools and GNU libraries

Two SW development models:

Bazaar

- Source code developed over the Internet in view of the public - *Linus Torvalds*



Tux

Cathedral

- Source code available with each software release, but code developed between releases is restricted to an exclusive group of software developers

Heckert



User View on Linux

- **Kernel** of the system: drivers, processes, memory, filesystem management
- **Filesystem**: different from Windows
- **Console**: text-based interface
- **Graphical interface**:
  - Management allowing remote transfer or graphics windows or even whole screen
  - Terminal (*cmd*)
  - Common applications: office, web-browsers, multimedia, file-browsers, coding editors, ...
- **Software repositories**
- Natural multi-user and multitasking

Based on free SW

→ Practically all parts of the system have number of variants
→ Number of Linux distributions:

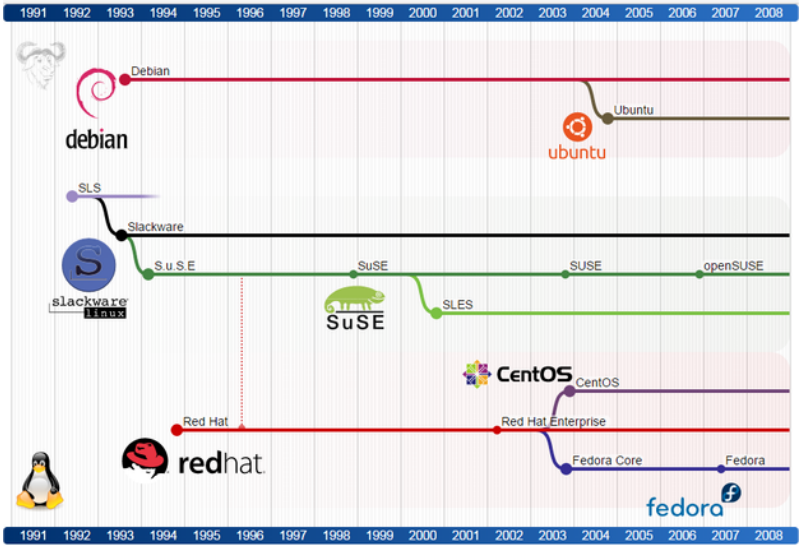
- Various graphical interfaces
- Various choices of default applications
- Various program and library versions (stable vs. bleeding edge)
- Various SW repository types

KFA MFF UK

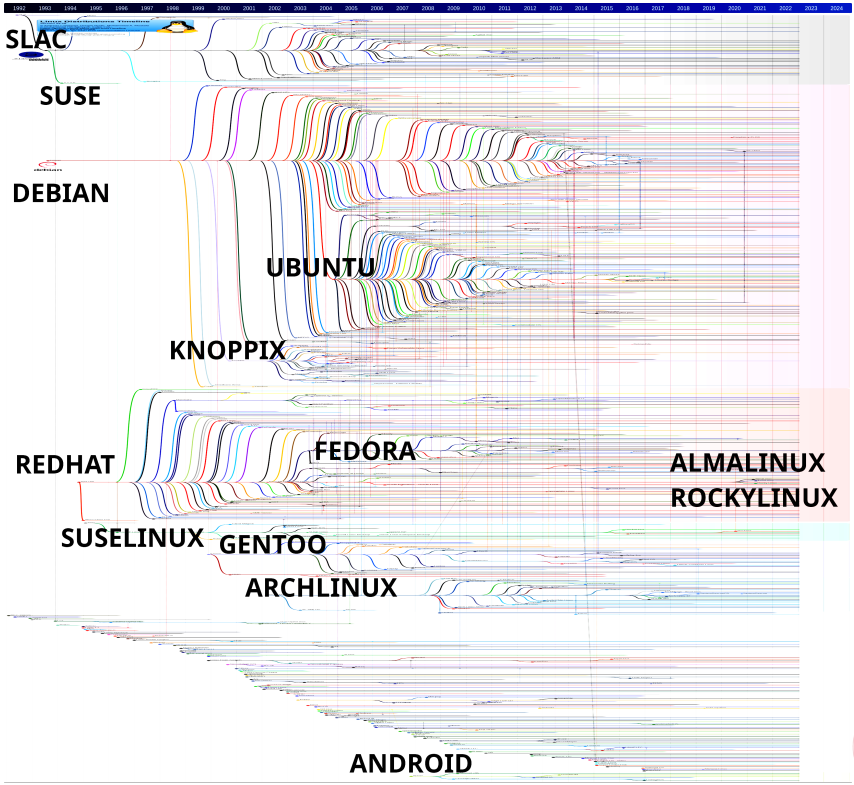
ÚČJF MFF UK

Linux Distributions

- List of components: kernel of the operating system GNU/Linux, libraries and other GNU tools
- Have its own installation image, packages and their repositories
- Historically 3 distributions: *Debian, Red Hat* and *Slackware* expanded into several hundreds, including *Ubuntu, openSUSE, Fedora, Mandriva, Gentoo,* ...
- Most distributions have their own Live versions: run from USB stick or CD/DVD



Linux Distributions Chart (up to 2023)



Debian



- *Debian* is one of the **most extensive** distributions
- Fully **developed by the community**
- Supports 11 platforms:
  - amd64, i386, Arm, PowerPC, mips, s390, ...
- More than 50000 packages
- Includes even non-Linux kernels (FreeBSD, NetBSD)
- Very **specific development cycle**:
  - Deployment of **stable** release (getting only minor security updates) once per $\sim$ two years
    - Very stable and secure Linux, often used for servers
  - In the mean time new packages and package versions are tested in **experimental**, **unstable** and **testing** branches
  - After certain time the *testing* release becomes *stable*
  - For impatient users, there are also *backports* for the *stable* release
  - Using *testing* distribution can serve as relatively stable "rolling distribution" (continuously getting updates without and "release" dates)
    - Using *unstable* release as "rolling" is not that comfortable as it can contain serious incompatibilities
- Debian (and most of its derivaties) uses package system based on `*.deb` packages and system `apt`
  - Automatic resolution of package dependencies, conflicts, diversions, alternatives
  - Pre/post-installation scripts
  - Complex system to create or build own packages
  - Widely considered as best packaging system

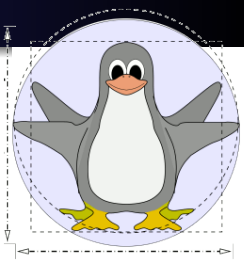
KFA MFF UK

ÚČJF MFF UK

Ubuntu



- *Ubuntu* is a system based on *Debian*
- Free, with both community and professional support
- Ubuntu community is based on principles described in the Ubuntu Manifesto:
  - Should have the freedom to download, run, copy, distribute, study, share, change and improve their software for any purpose, without paying licensing fees.
  - Should be able to use their software in the language of their choice.
  - Should be able to use all software regardless of disability.
  - *Southern African philosophy of ubuntu (literally, "human-ness")*
- Suitable both as user-PC as well as for servers
- Supports **most common architectures**: PC 32bit (i386), PC 64bit (amd64) and PowerPC (older Apple iBook, Powerbook, G4, G5)
- Half-year release, long-term supported release every 2 years
- **Unity** is the default GUI (Graphic User Interface)
  - MAC-like, touchscreen friendly
  - However, number of variants with different GUI exists (*Kubuntu, Xubuntu, Lubuntu*, ...),
  - As well as specific-focus derivatives (*Edubuntu*, ...),
  - or enhanced distributions (*Linux Mint*, ...)



- *KNOPPIX* is a primarily **Live OS**
- Already after installation contains wide range of SW, automatic HW detection and HW support (sound, graphics, peripherals)
  - The CD contains up to 2 GB of compressed SW
  - The DVD version up to 8 GB of compressed SW
- Can serve as "rescue" Live system
- Can be installed as ordinary user-PC as well
- Derivatives: localized and enhanced system *Danix*

Red Hat



- *Red Hat* is a commercial Linux
- **Enterprise-level support**
- Packages system uses `*.rpm` files (2^{nd} widely-used package system)
- Number of free derivates with community-support only

KFA
MFF UK

ÚČJF
MFF UK

Fedora



- *Fedora* is a **non-commercial Red Hat derivative**
- Development (community) **supported by the Red Hat** company
- Serves as testing platform for the commercial *Red Hat* releases
- **Progressive**, implementing new features very soon
- Focused on user-PC, suitable for beginners
- Wide range of SW in the distribution
- Supports **most common architectures**: i386, amd64 and PowerPC
- Very **strict in licensing**, e.g. yet recently missing `*.mp3` support



- *CentOS* is another *Red Hat* derivative
- Originally started as independent distribution, but transferred under *Red Hat* (developers from the *Red Hat* open-source team)
- Free and community supported
- Often **used on servers**
  - Used in CERN (European Organization for Nuclear Research)



- *Mandriva* is originally based on *Red Hat*
  - Started in 1998 started by *Geld Welb*, attempting to make more comfortable GUI experience
  - The modified distribution, called *Mandrake*, was put on servers and *Geld Welb* left for holidays
  - Strong response from users after return, offers to contribute from developers and testers
  - Nowadays, *Mandriva* has number of developers in France and USA
- Focusing on **office-PC and multimedia-PC**
- Wide range of SW in the distribution
- Easy maintenance
- Strong **involvement of users** on the final shape of the distribution
  - Beta-testers
  - Suggestions for improvements and changes

Slackware



- *Slackware* was **the first widely used Linux distribution**, created in 1992 by *Patrick Volkerding*, who remains its only official developer
- Although many contributors bring new packages, the existence of the only one developer makes the concept of the distribution unified and development trouble-free
- *Slackware* is known for **simplicity and clarity**:
  - Using easy to understand text configuration files
  - Transparency in every process
- **Simple packaging system**, able to only install and remove packages:
  - No dependency resolutions
  - No automatic updates
  - → Very flexible in what is being installed on the system
  - Still there are enhancements that overcomes the simplifications above

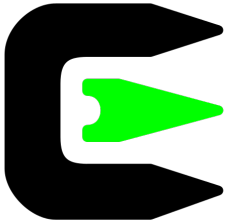


- *SUSE* was created in 1992 in Germany
  - First release in 1994
  - 2003/2004 bought by **Novell**
- Release of boxed version with manual first (together with Live DVD for preview only), online after serveral months
- Possibility to buy professional support
- Later project *openSUSE*:
  - Community based program sponsored by Novell
  - Easier and free access to *SUSE* Linux



- *Gentoo* uses system similar to the *BSD* ports, called `Portage`
  - `Portage` is a very flexible packaging system
    (flexible in the installation and maintenance of the SW)
  - SW is being built from source at the time of installation
  - The `Portage` system allows to set various installation configurations via use-flags
  - Includes build-dependencies
  - Safe installation and deinstallation of packages (via so called sandbox)
  - Protection of config files etc.
- **Very flexible installation**, e.g. one can built system from source-code with selected optimization → *Gentoo* is often considered as **metadistribution**
- Active development, rapid fixes and updates
- Half-year releases

Cygwin



- *Cygwin* is a Linux system that is compiled and runs natively on Windows
  - No emulation as virtual machine
- Settings partly bound to windows (users)
- Possible to lanuch windows programs
- Nowadays owned by *Red Hat*

Windows Subsystem for Linux



- *Windows Subsystem for Linux* (WSL) introduced in 2016 Windows update
  - Shell on Windows / *Ubuntu* on Windows
  - But not really limited to one distribution
  - Enable in "Windows Features" seetings
- Allows to run linux binaries (64bit) natively on Windows 10
- Can also run graphical applications (after some tunning...)
- Superuser (root) privileges limited (can't change Windows config - network etc.)
- Underlying principle: Linux system calls translated to Windows system calls
  - Performance worse than on native Linux
- Special filesystem

VirtualBox, VMware



- *VirtualBox* and *VMware* create virtual machine, in which another OS can run (Windows, Linux, MAC)
- Commercial SW, *VirtualBox* is open-source and free for personal or educational use
- Makes use of Intel/AMD hardware-assisted virtualization
- Can run several Linux installations on Windows in parallel (if enough resources on PC)

Linux Installation Notes

Install Linux Inside Windows

- Not genuine Linux experience: administration and hardware management may be limited by what Windows allows...
- No need for disk repartitioning
- Easy to uninstall

Cygwin

- Simply follow installation instructions, installs as an ordinary windows application

Windows Subsystem for Linux

- Enable in "Windows Features" settings
- Install *Ubuntu* via Microsoft Store: search for "Run Linux on Windows" and choose distribution

VirtualBox

1. Install *VirtualBox*
2. Create new virtual system: allocate part of RAM and disk space (VDI type)
3. Download *Ubuntu* ISO image and install the system in *VirtualBox* via "Start" button

Install Linux as Dual-Boot Machine

1. Free disk space in Windows if needed; tips:
   - May need to switch off (temporarily) disk encrypting, otherwise disk repartitioning won't work
   - Disk cleanup, including cleanup of system files (downloaded updates, old win versions, log files from upgrades, ...)
   - Compress system files
   - Disable hibernation (removes large `hiberfile.sys` file)
   - Make small memory swap file (shrinks large `pagefile.sys` file)
   - Make small space for restore points
   - Make small shadow storage space
   - Cleanup `C:\SWSETUP` which contains driver updates installation files
   - Backup notebook recovery partition to USB stick and then delete the partition; not the $\sim (1-2)$ GB at disk beginning, but the $(\sim (10-20)$ GB at the end !
   - $\rightarrow$ **Can shrink Windows partition down to $\sim$ 30 GB (but will need USB for larger Windows updates)**
2. Most distributions have (Live) CD/DVD/USB to start the installation
   - Install the `*.iso` image on USB using *Rufus* program on Windows
3. No complications when following instructions...
4. However, someties it is good to have partitioning of disk under control, instead of relying on the partitioning the Live Linux performs



KFA MFF UK

ÚČJF MFF UK

Disk Repartitioning

1. Use Live Linux, e.g. specialized *System Rescue CD* using *GParted* program
2. Follow instruction to install on USB:
   - With *Rufus* on Windows
   - With `dd if=name.iso of=/dev/sdX ; sync` on Linux
3. May need to play with Boot/Startup sequence and with UEFI/SecureBoot in BIOS to allow boot from the USB stick
4. Most likely will have to create logical partition, since the default partition table does allow more than 4 primary partitions
   - Windows, Windows recovery, Linux root, Linux home, swap

Partitions to create

- Root partition '`/`' of type *ext4* and size $\sim (20 - 50)$ GB
- Home partition '`/home`' of type *ext4*
- Swap partition of type *swap*
  - $> 2\times$ RAM for small RAM systems (< 4 GB)
  - $> 1\times$ RAM for middle-size RAM systems (< 8 GB)
  - $0.5\times$ RAM for large RAM systems (> 8 GB) and when hibernation-to-disk is not needed
  - Ubuntu recommendations: $\mathrm{SWAP} = \sqrt{\mathrm{RAM}}$ when hibernation-to-disk is not needed, add $1 \times \mathrm{RAM}$ when it is needed

When touching (resizing) Windows partition, always make only one step at a time and reboot to Windows for automatic check of the newly-repartitioned disk. Otherwise Windows may become unbootable ! (although there is a fix...)

KFA MFF UK ÚČJF MFF UK

Dual-boot features

- May need to re-point WinRE (Windows Recovery Environment) to the correct partition on disk (usually is at the end of the disk)
  - `reagentc /info`
  - `reagentc /setreimage /path \?\GLOBALROOT\device\harddisk0\partition2\Recovery\WindowsRE`
  - `reagentc /enable`
- Windows recovery to notebook factory settings may not work anymore
- Putting PC to hibernate on disk in both Windows and Linux at the same time may lead to loss of data if Windows disks are mounted in Linux

Linux boot recovery (Grub)

- May happen if / after reinstalling Windows
- Follow instructions at *System Rescue CD*

Linux Uninstall

1. Do not forget any data on your Linux disks :)
2. Set bootable flag (e.g. in *GParted*) program back to the Windows system partition
3. Install back the Windows *Master Boot Record* from within Linux (or Live Linux); several ways are possible, depending on what SW is available on your Linux (commands below to be run as root):
   A. `dd bs=440 count=1 conv=notrunc if=/usr/lib/syslinux/mbr.bin of=/dev/sdX` (using `syslinux` package, `mbr.bin` may be located at different path)
   B. `lilo -M /dev/sdX mbr` (using `lilo` package)
   C. `install-mbr -i n -p D -t 0 /dev/sdX` (using `mbr` package)
   D. `apt install boot-repair` from `ppa:yannubuntu/boot-repair` repository
4. Boot to Live Linux (*System Rescue CD*)
5. Wipe data (rewrite with random contents) using `schred` / `wipe` commands
   - `shred -v -z -n 10 /dev/sdXY`
6. Delete the Linux partitions and resize back the Windows one(s)

Again, the deletion and resizing should be done in single steps, followed by reboots to Windows !

Faculty of Mathematics and Physics
CHARLES UNIVERSITY

NOFY077

Introduction to the Linux OS

Peter Huszár
KFA: Department of Atmospheric Physics

Pavel Řezníček
ÚČJF: Institute of particle and nuclear physics

September 30, 2025

Quick list of Applications

Office Suites and Applications

Microsoft **Office** not available on Linux. However, there are number of alternatives:

- *LibreOffice* (former *OpenOffice*) fully included
- Gnome Office: *Abiword*, *Gnumeric*, *Gnucash*
- *Calligra Suite* (KDE focused office package)

PDF viewers:

- *Okular*: KDE PDF viewer, capable of adding annotations
- *Evince*: Gnome PDF viewer
- Old *Adobe reader* (needs special package repository)
- *Foxit reader*: not in package repositories, needs to be downloaded and installed manually

Latex packages naturally included in all distributions, with number of graphical editors allowing to debug Latex compilation and preview generated PDFs.

- *Texstudio*
- *Texmaker*
- *Kile* (KDE)
- *Visual Studio Code* + Latex plugin: free Microsoft product

Web-Browsers

The most **common browsers** are included:

- *Firefox*
- *Chrome* / Chromium
- *Opera*

There are also DE specific ones:

- *Konqueror* (KDE)
- *Epiphany* (Gnome)

As well as **terminal-based browsers** (limited use only, no javascripts etc.):

- *Lynx* / *Links* / *Elinks*
- *curl* / *wget*
- Useful for scripts to extract information from web

Number of **file browsers**, usually specific to DE:

- *Dolphin* (KDE)
- *Nautilus* (Gnome)
- *Thunar* (XFce4)
- *Krusader*: KDE-based browser with *Total Commander* like look and features
- *Midnight commander*: terminal based 2-panel file browser (*Norton/Total Commander* like); very useful when working on remote machines as it does not need to transfer graphics

Terminal Emulators

Number of **terminal emulators**, usually specific to DE:

- *xterm*: basic terminal emulator
- *rxvt-unicode*: basic terminal emulator with unicode support
- *konsole*: KDE-based terminal emulator, support for tabs, complex configuration including predefined profiles etc.
- *gnome-terminal*: Gnome-based terminal emulator, support for tabs, complex configuration including predefined profiles etc.
- *xfce4-terminal*: Xfce4-based terminal emulator, support for tabs, complex configuration

Editors

Editors **for programmers** with various level of IDE:

- *Kate*: KDE based
- *KDevelop*: KDE based
- *Geany*: Gnome based
- *Eclipse*: heavy, Java based
- *Gedit*: light
- *Visual Studio Code*: Microsoft product, but open-source and free use (both home and commercial)

Console editors (useful when working on remote machines):

- *nano*: simple
- *jed*: simple
- *vim*: very powerful, but difficult for beginners
- *emacs*: very powerful, but difficult for beginners

Figures:

- *Gwenview* / *Gthumb* as picture viewers
- *Gimp* for bitmap drawings editing
- *Inkscape* for vector drawings editing

Video:

- *VLC*
- *SMplayer*
- *Xine*

Number of **command-line programs** for figures/audio/video conversions and processing

- *convert* for figures manipulations
- *ffmpeg* for audio/video manimulation

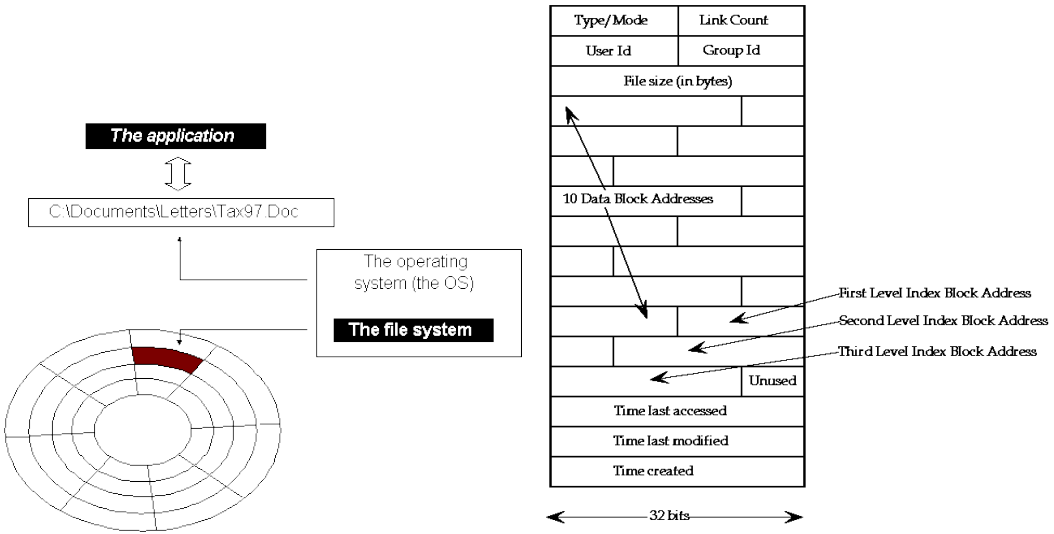
Emulators

Emulators of **other (usually older) operating systems**:

- *Wine*: emulator of Windows, many Windows applications able to run on that (not the complex ones as MS Office)
- *DosBox* / *DosEMU*: MS DOS emulators

Filesystems

System defining the way files and directories are stored on the physical disk



Command to show this file information: `stat`

- FAT: *File Allocation Table*
  - The original MS-DOS filesystem, tree structure
  - Limited depth of sub-directories, limited filename lenght
  - Follow-up systems: FAT8/16/32, VFAT, exFAT
    - VFAT and exFAT are nowadays used on external/flash USB disks
- NTFS: *NT File System*
  - First appeared on Windows NT
  - Current MS filesystem on Windows
- ISO9600: Read-only files system for CD/DVD
  - Limited subdirectory length, limited filenames
  - Enhanced systems: Joliet, ElTorito, Rock Ridge

MS filesystems in Linux

- Linux can work with all of them (R/W).
- MS formats are not open $\Longrightarrow$ reverse engineering
- Support for the newest exFAT is yet somewhat experimental

Journaling filesystems on Linux

- Keeps track of disk operations not yet performed / finished
- Minimizing loss of data

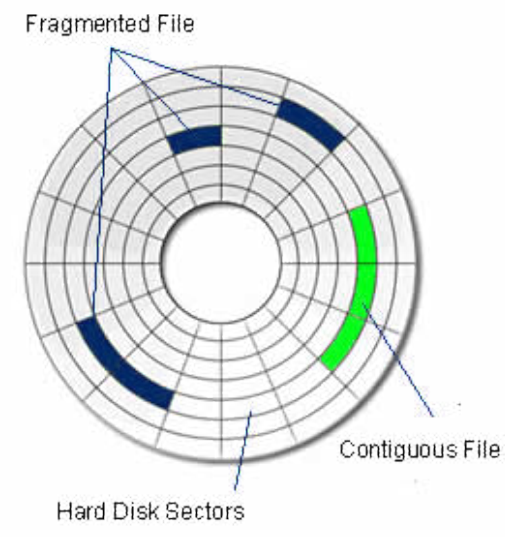
Swap

- Enhancing RAM of the system
- When RAM gets full, swap is used to store temporarily unused data
- No special structure
- Windows equivalent: pagefile.sys

- `Ext2`, `Ext3`, `Ext4`: The most common on Linux
  - `Ext3` = `Ext2` + journal
  - `Ext4` allows for larger files and for more files per dir (32000 vs 64000)
- `XFS`: Silicon Graphics for the OS *IRIX*
  - Good performance for large data handling
- `ReiserFS`, `Reiser4`:
  - Good performance for large number of small files
  - Connecting advantages of filesystems and databases
  - Developed by Hans Raiser (in 2008 convicted murderer)a
- `JFS`: IBM for the OS *AIX* and later for *OS/2* and Linux
  - First journaling system
- `AFS`: *Andrew File System*
  - Distributed file system for large networks of servers and client workstations

Filesystém	Max. velikost filesystému	Velikost bloků	Max. velikost souboru
Ext2	4 TB	1KB-4KB	2 GB
Ext3	4 TB	1KB-4KB	2 GB
ReiserFS	16 TB	až 64KB	2^10 PB *1
XFS	18000 PB *1	512B - 64KB	9000 PB *1
JFS	512 B / 4 PB *2	512B, 1024B, 2048B, 4096B	512B / 512Tb *2

Fragmentation



No defragmentation on Linux

- MS filesystems need occasional defragmentation:
  - Files are placed to the next free sector on the physical disk $\Longrightarrow$ when a file is enlarged, the next sector of physical disk is usually already occupied by another file $\Longrightarrow$ file fragmentation
- Linux filesystems defragment on the fly:
  - Files are placed on distant sections on the physical disk $\Longrightarrow$ space around existing file is usually free
  - If there is a danger file would become fragmented, the system tries to find a continuous space on the physical disk and moves the file in there
  - Fragmentation thus appears only when disk is close to full

Linux Files

- *Directory* (`d`): container of files
- *Plain file* (`-`): ordinary file (collection of bytes)
- *Symbolic link* (`l`): analogy of hypertext link or shortcut in Windows
- *Hard link*: similar to ordinary file, just another name for one file
- *Block and character device* (`b`,`c`): representation of HW devices
- *Name pipe* (`p`): named pipes for communication between applications instead through RAM
- *Socket* (`s`): duplex communication between processes

```
meop35 exam # ls -lai
total 21
1433763 drwxr-xr-x  5 petr petr     288 Oct  9 11:59 .
   4350 drwxr-xr-x 11 petr users    976 Oct  9 11:57 ..
1434761 -rw-r--r--  1 petr petr   13643 Oct  8 21:37 FILESYSTE.win
1434763 lrwxrwxrwx  1 petr petr      11 Oct  9 11:59 core -> /proc/kcore
1434735 drwxr-xr-x  2 petr petr      48 Oct  9 11:57 doc
1434764 srwxrwxrwx  1 petr petr       0 Oct  9 11:59 gpmctl
1434757 drwxr-xr-x  2 petr petr      72 Oct  9 11:58 hudba
1434758 -rw-------  1 petr petr      31 Aug 21 13:52 hymna.mp3
1434756 drwxr-xr-x  2 petr petr      80 Oct  9 11:58 prednasky
1416878 crw-rw----  1 root tty   2, 190 Oct  9 11:59 ptyae
1434762 lrwxrwxrwx  1 petr petr       9 Oct  9 11:58 sym_link -> hymna.mp3
meop35 exam #
```

List files using command `ls -lah`

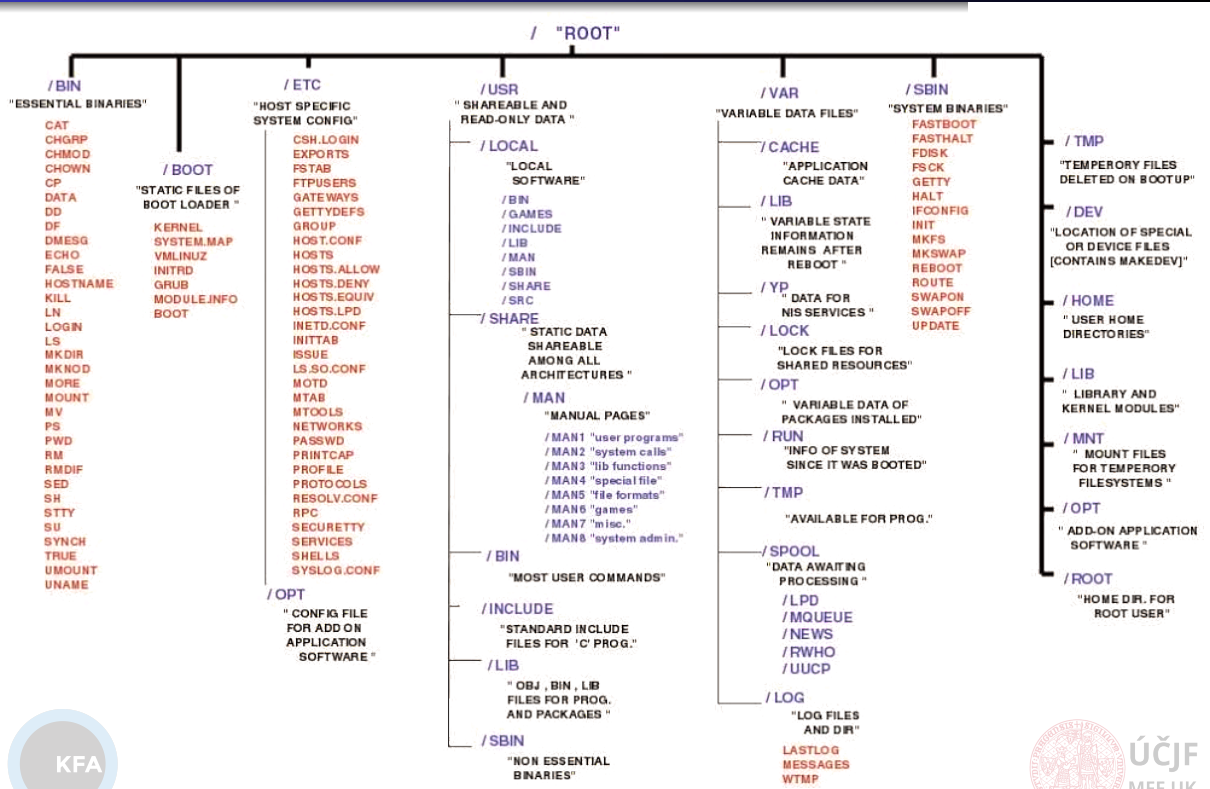
Structure of Directories in Linux

Directories and files in Linux

- Physical disks not visible in the structure (no `C:` and `D:`), disks are *mounted* to directories (admin can define what is mounted where)
  - e.g. typically "disk D:" is mounted to `/home` directory with users data
- Using '`/`' (slash) instead of Windows '`\`' (backslash)
- Linux filesystems (like `ext4`) features:
  - Directory and file names are CaSe SeNsItIvE
  - Native support of access control (user/group/all, read/write/execute)
  - Links (a bit similar to shortcuts in Windows, but for applications behave as real files)
- Hidden dirs and files starting with dot (`.*`, e.g. `.config`), usually keeping configuration of applications
- Special files representing e.g. HW devices and their configurations
- Special directory names:
  - Root directory '`/`' (top directory of the Linux system)
  - Home directory of a user '`~/`' (representing typically `/home/username`)
  - Current directory '`./`'
  - Directory one level up '`../`'
- **Absolute paths** starting from root of the system `/`
- **Relative paths** from current directory (may or may not start with '`./`')
- Text files has different standard for end-of-line (EOF) code from Windows/DOS/Old MACs
  - Conversion via commands `dos2unix` and `unix2dos`

KFA MFF UK
ÚČJF MFF UK

Diagram of the Directories

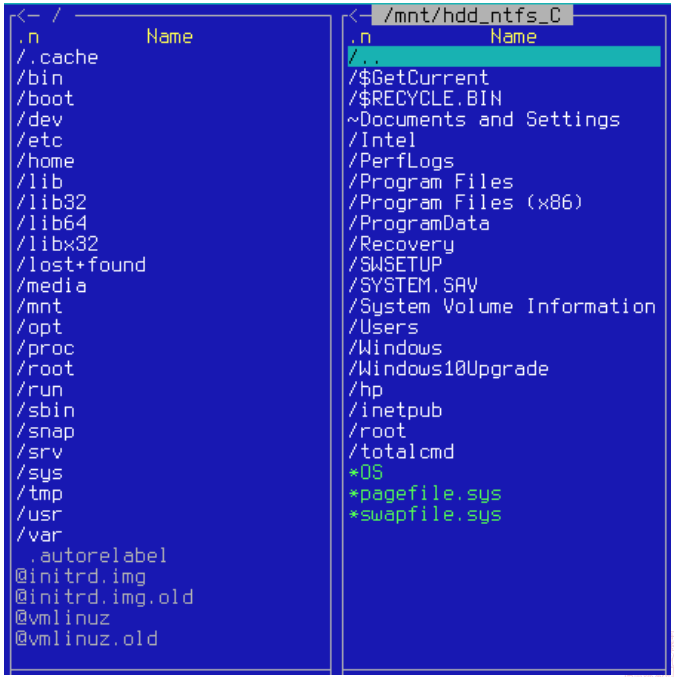


KFA
MFF UK
ÚČJF
MFF UK

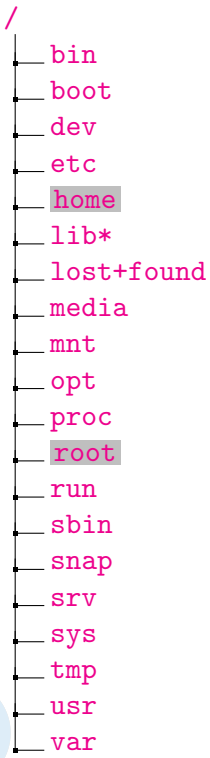
Linux Root Directory

- /
  - bin
  - boot
  - dev
  - etc
  - home
  - lib*
  - lost+found
  - media
  - mnt
  - opt
  - proc
  - root
  - run
  - sbin
  - snap
  - srv
  - sys
  - tmp
  - usr
  - var

Snapshot of top directories in Linux and Windows 10:



User Home Directory



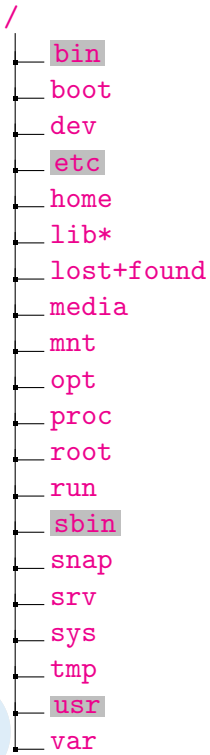
/home

- Home directory of users in `/home/$(USER)`
- Place used by all ordinary users to store data, configurations, possibly also local applications
- Other users can usually read files of other users, unless manually disabled
- `/home/$(USER)/Desktop`: Any file/directory placed here will appear on GUI desktop
- `/home/$(USER)/.*`: User configuration files of various applications
  - Especially in `.config` subdirectory
  - Files associations, user shortcuts etc. in `.local/share/` (`applications`) subdirectory
- Analogy of `C:\Users`, resp. `C:\Documents and Settings` in Windows

/root

- Home directory of administrator (*superuser*/*root* user)
- Same structure as for users in `home/$(USER)` directory

Application Installation



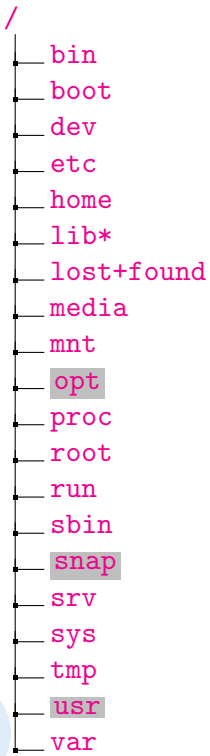
Ordinary application files

- Executable files in `/usr/bin`
- Data files in `/usr/share`
- Documentation, licence, examples etc. in `/usr/share/doc`
- Manual pages (help) for executables in `/usr/share/man`
- Libraries in `/usr/lib*`
- Header files of libraries in `/usr/include`

System application files

- Executable files in `/bin`
- *Superuser* executable files in `/sbin` and/or `/usr/sbin`
- `/etc`: Configuration of the system (services) and applications, common for all users
  - `etc/default`: Most common default settings
  - Analogy of Windows registry

Application Installation (Special, Local)



Local installations

- Usually not going through the ordinary package system, but directly copied to the system
- These programs and libraries placed in `/usr/local`
  - Contains similar sub-structure as the root `/` and/or `/usr` directories
  - I.e.: `etc`, `bin`, `lib`, `share`, ...

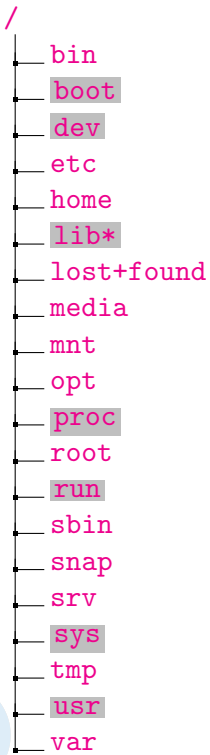
3rd party / commercial applications

- Whole application stored in `/opt`, with links to `/usr/bin` etc.
- `/snap` stores distribution-agnostic applications
  - All dependencies (libraries etc.) are included

KFA MFF UK

ÚČJF MFF UK

Boot, Linux kernel, devices



Boot configuration in /boot

- Linux kernels to boot
- Configuration of the boot manager (*GRUB*)

Linux kernel modules and source

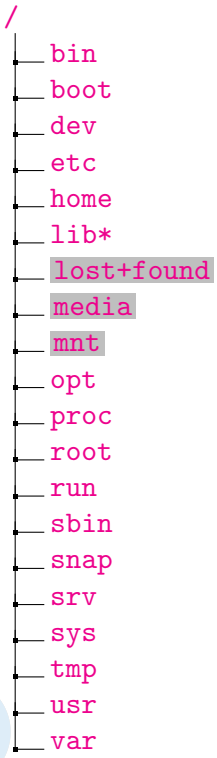
- Linux kernels sit in /boot
- Linux kernel modules in /lib/modules
- Linux kernel source code in /usr/src

Hardware devices and processes

- Connected HW devices represented by special files in /dev
  - Storage devices to mount
  - Dustbin /dev/null
  - Console, random generator device, ...
- Configuration of the HW devices in /sys, *superuser* can use the files to modify the configuration of the HW drivers
- /proc directory keeps information about running processes, HW configuration (memory, cpu)
  - Interface to the kernel internal data structures
- Runtime information about processes in /run (e.g. connected WiFi)

KFA MFF UK
ČJF MFF UK

Storage devices



Lost files

- Lost and recovered files after OS crash

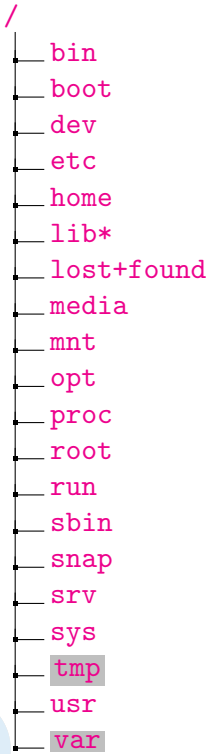
Storage devices connected to the system

- Removable devices usually appear in `/media`, resp. `/media/$(USER)`
  - Modern distributions connected them (*mount*) automatically
- Windows partitions, network storages etc. are usually mounted to `/mnt`
  - Follow rules defined in `/etc/fstab` file
  - And/or rules for auto-mounting in `/etc/auto*` files

KFA MFF UK

ÚČJF MFF UK

Temporary files



(Semi-temporary application settings in /var

- /var/cache: Temporary files of applications, e.g.:
  - Downloaded installation packages of apt system in /var/cache/apt/archives
- /var/lib: Variable system information, e.g.:
  - Network connections
  - Installed packages info in /var/lib/dpkg/info
  - Content of package repositories in /var/lib/apt/lists
- /var/spool: Data awaiting further processing, e.g.:
  - E-mails
  - Print jobs
  - Scheduled user tasks
- /var/log: System log files:
  - Crucial for administrators
  - Boot, kernel messages, logins, services (e.g. WWW servers) etc.
- /var/www: WWW server data
- /var/lock: Application and system lockers

Temporary files in /tmp

- Temporary files of applications, e.g.:
  - Opened / downloaded files from web-browsers
  - Archives unzipped in file-browsers

Work with the Command Line in Terminals

Command Line

- Very powerfull in Linux, allows to master the OS
- Commands are taken by a *shell* and given to the OS
- Allows complicated scripts including loops, macros, conditions, etc.
- Several *shells* exists:
  - `bash`: most commont
  - `dash`: minimalistic, for system scripts
  - `zsh`: programmers focused
  - `ksh`:
  - `csh`: different syntax from bash-like shells above
  - `tcsh`: enhances csh
  - `fish`: friendly interactive shell
- `chsh` command to change user shell

Shortcut Keys

- **Left/Right/Home/End**: navigate cursor through the command line
- **Up/Down-Arrows**: browsing through history of commands
- **Tabulator**: complements commands or file names
  - Search for commands in the standard executable paths
  - Shows all possibilities in case the completion is ambiguous
  - Possibility to enhance completion for specific commands (ssh and remote host names etc.)
- **Ctrl-r**: search in history of commands backward
- **Ctrl-s**: search in history of commands forward
- **Ctrl-g**: end of search (not-only search) mode
- **Ctrl-l**: clear terminal window
- **Ctrl-q**: unblocks blocked terminal
  - Some terminal emulators get blocked with the Ctrl-s command
- **Ctrl+c**: interrupt running process
- **Ctrl+d**: interrupt writing into file (e.g. in `cat > filename`)

Shell Settings

- Configuration / startup files:
  - `/.bash_history`: history of commands
  - `/.bashrc`: startup script for non-login interactive shells
  - `/.bash_profile`: startup script for `login shells` (login in text console or from remote host)
    - `/.profile` is read by bash too, to be backward-compatible with old `sh`
  - `/etc/profile`: system-wide
  - `/etc/motd`: message on login shells
- Other shells have similarly-named files
- Typical configurations in `/.bashrc`:
  - Format of command prompt: PS1 (also PS2-PS4 for 2nd etc. level of prompts)
  - PATH to executables
  - PATH to 3rd party or local libraries (LD_LIBRARY_PATH)
  - Enhanced completion and other plugins (e.g. list of commands in not-installed packages)
  - Aliases (alias/unalias commands)
  - Default limits (ulimit command)
  - Environmental variables / program setups (`export | less`)
  - History length

Faculty of Mathematics and Physics
CHARLES UNIVERSITY

NOFY077

Introduction to the Linux OS

Peter Huszár
KFA: Department of Atmospheric Physics

Pavel Řezníček
ÚČJF: Institute of particle and nuclear physics

September 30, 2025

Commands in Linux

Programs available in shell or in the executable PATHs.

- General structure: [path/]command [options] [arguments]
- List of options and arguments (e.g. for `systemctl` command):
  - `command --help`, resp. `command -h`: basic syntax and options of the command
  - `man command`: manual pages with more detailed info
  - `info command`: interactive manual pages (jump to references etc.)
  - `command -v`. resp. `command --verbose`: be verbose about what is being done
- Run in background, while continuing work in shell: `command ... &`
- `fg`: Put command already running in background back to foreground of the shell
- **Ctrl+z**: Suspend process / command (stops processing)
- `bg`: Put command into background (typically used to let suspended command running again in the background)
- When command / program ends, it usually returns a *return code*
  - 0 ... usually means "no error"
  - non-zero is usually indication of premature exit of the command because of some problem
- Find full path to a command: `which command` (or search for it in aliases: command `alias` lists existing aliases)

Scripts: Basics

Sequence of commands can be put into script files.

- `#` are used for comments
- Special header "comment": `#!/usr/bin/zsh` instructs the script to be run by the `zsh` shell. Not only for shells, but also for interpreters like `python`
- `exit` [`number`] to quit script [and possibly return a *return code*]
  - Not needed at the very end of a script, it will end by itself
- `set -x` command inside a script instruct to show the commands being run by the script
- Two ways how to run a script:
  - *./script.sh*: starts a new shell and runs the script in it
  - *source ./script.sh* (or also `. ./script.sh`: runs the commands from the script one by one in the current shell → i.e. as if one would write them manually in the current terminal
  - PS: The *'./'* makes sure it runs `script.sh` in the current directory, and not somewhere from the `$PATH` paths

```
#!/bin/sh
echo $SHELL
sleep 0.5
```

Commands: User Information

- `login`: Login user to the session, normally not directly invoked
- `logout`: Logout user from the session
- `exit`: Exit from the terminal/shell/script
- `id` [user]: User and group id information about myself
- `groups` [user]: List groups user is in
- `whoami`: Return user-name
- `who`: Show who is logged in
- `w`: Show who is logged in and what is he/she doing
- `last`: Show list of last logged-in users
- `su/sudo`: Become superuser
  - `su -` to make sure superuser paths are set
  - `sudo -i` to start superuser shell

Commands: File / Directories

- `ls` [path/file]: List files/directories
  - `ls -1`: List one file/dir per line
  - `ls -l`: List full information:

```
-rw-r--r--  1 reznicek reznicek 15360 Oct 14 19:40 osnova_Uvod_do_Linuxu.doc
drwxr-xr-x 18 reznicek reznicek  4096 Jan 31 2019  skola
```

    - File type (file='-', dir='d', ...), access rights
    - Number of links (copies)
    - User name
    - Group name
    - Size in bytes
    - Date and time of last modification
    - File name
  - `ls -h`: Human readable file size
  - `ls -a`: List also hidden files and directories (`.*`)
  - `ls -d`: Show information about directory instead of listing its content
  - `ls -R`: List recursivelly also content of sub-directories
  - `ls -t`: Sort by modification time
  - `ls -r`: Reverse sort order (name, time)
- `stat file`: Disk-storage info about `file`
  - Physical place on disk
  - Creation, modification and access times

Commands: File / Directories

- `cd` [path]: Change directory
  - `cd ../`: Go one directory up
  - `cd ./`: Go to (stay in) current directory
  - `cd /`: Go to root (top filesystem) directory
  - `cd ~/`, `cd`: Go to home directory
  - `cd -`: Go to previous directory (back from last cd command)
  - Change directory *relatively* to the current dir or *absolutelly* specified path from the root dir
    - *relative* path example: `cd work/papers` or `cd ./work/papers`
    - *absolute* path example: `cd /home/username/work/papers`
- `pwd`: Print current directory
- `readlink -f file`: Print full path to the given `file`
- `cat file`: Dump content of `file` on the screen
- `less file`: Browse `file` content on the screen
  - Enhanced `more` command (also on Windows)
  - Search for content ('/' key)
- `touch file`: Update *modification time*
- `diff file1 file2`: Show difference between two files, GUI versions exist too (`meld`, `diffuse`)
- File and directory manipulations (see later lectures in details):
  - `cp`, `mv`, `rm`: copy, move/rename, remove files
  - `mkdir`, `rmdir`: create directory, remove (empty!) directory
- Terminal-based file browser: `mc`

KFA MFF UK
ÚČJF MFF UK

Work on Remote Machine via SSH

Secure Shell Connection, assuming the host PC is running `sshd` server (just install the ssh-server package...)

Client can:

- `ssh -Y username@hostname`
- Gains access to login shell on remote machine
- Can send graphical windows (e.g. xterm)
  - `-Y` allows to transfer graphics
- With sudo/su commands can manage the computer (including reboots, poweroff etc.)
- Configure automatic options for ssh via `$HOME/.ssh/config` file:

```
Host lxplus
  user reznicek
  Hostname lxplus.cern.ch
Host ipnp-stick
  User mlive
  Hostname 10.77.0.11
```

At MFF there is a faculty computing Linux cluster `chimera` for all faculty members:
https://www.mff.cuni.cz/en/hpc-cluster/general-information

Manipulation with users and groups

Users:

- Users stored in `/etc/passwd`

```
student:x:1001:1001:Student linuxu,T11,1234,5678,Poznamka:/home/student:/bin/bash
```

- `adduser`: add new user, copy `/etc/skel` content to the newly created home directory. Default options in `/etc/adduser.conf`
- `deluser`: remove user (home directory is kept by default)
- `usermod`: modifications to user settings (home dir, id, password-expiry, groups, shell, lock-password)
- `chfn`: changer user information (full name etc.)
- `passwd`: change user password
- `chsh`: change user shell

Groups:

- Groups stored in `/etc/group`
- `groupadd`: add new group
- `groupdel`: delete group
- `groupmod`: name, id
- `gpasswd`: groups can have passwords, administrators

KFA MFF UK
ÚČJF MFF UK

Change ownership of a file/dir

Users:

- `chown username:groupname file`: change user and group ownership of a file/dir
  - `chown -R`: recursively for all sub-directories
  - `chown --dereference`: for symbolic links "jump" to the real file the link is pointing to
- `chgrp groupname file`: changer group ownership of a file/dir
- Username change mostly only for superuser, user cannot "give" file to another user. But owner can modify group =¿ give file for r/w to other users via groups

Change read/write/execute/access rights to files/dir

```
meop35 exam # ls -lai
total 21
1433763 drwxr-xr-x  5 petr petr     288 Oct  9 11:59 .
   4350 drwxr-xr-x 11 petr users    976 Oct  9 11:57 ..
1434761 -rw-r--r--  1 petr petr   13643 Oct  8 21:37 FILESYSTE.win
1434763 lrwxrwxrwx  1 petr petr      11 Oct  9 11:59 core -> /proc/kcore
1434735 drwxr-xr-x  2 petr petr      48 Oct  9 11:57 doc
1434764 srwxrwxrwx  1 petr petr       0 Oct  9 11:59 gpmctl
1434757 drwxr-xr-x  2 petr petr      72 Oct  9 11:58 hudba
1434758 -rw-------  1 petr petr      31 Aug 21 13:52 hymna.mp3
1434756 drwxr-xr-x  2 petr petr      80 Oct  9 11:58 prednasky
1416878 crw-rw----  1 root tty   2, 190 Oct  9 11:59 ptyae
1434762 lrwxrwxrwx  1 petr petr       9 Oct  9 11:58 sym_link -> hymna.mp3
meop35 exam #
```

Command `chmod`

- `chown ugoa+/-rwxst file`
  - `u`: user rights
  - `g`: group rights
  - `o`: rights of others
  - `a`: rights of all (user, group, others), not specifying who defaults to "all"
  - `+`: add rights
  - `-`: remove rights
  - `r`: read rights
  - `w`: write/modify rights, for dirs only works when 'x' is set too
  - `x`: executable (file), directory enter and listing allowed (directories)
  - `s`: setuid bit, allows to run command with file-owner privilegies (`/etc/passwd`)
  - `t`: sticky bit for directory (o+t): everyone can create files, but files can be deleted only by the owners (Despite the "xw" rights for all in the directory - e.g. `/tmp`)

KFA MFF UK
ÚČJF MFF UK

Change read/write/execute/access rights to files/dir

```
meop35 exam # ls -lai
total 21
1433763 drwxr-xr-x  5 petr petr     288 Oct  9 11:59 .
   4350 drwxr-xr-x 11 petr users    976 Oct  9 11:57 ..
1434761 -rw-r--r--  1 petr petr   13643 Oct  8 21:37 FILESYSTE.win
1434763 lrwxrwxrwx  1 petr petr      11 Oct  9 11:59 core -> /proc/kcore
1434735 drwxr-xr-x  2 petr petr      48 Oct  9 11:57 doc
1434764 srwxrwxrwx  1 petr petr       0 Oct  9 11:59 gpmctl
1434757 drwxr-xr-x  2 petr petr      72 Oct  9 11:58 hudba
1434758 -rw-------  1 petr petr      31 Aug 21 13:52 hymna.mp3
1434756 drwxr-xr-x  2 petr petr      80 Oct  9 11:58 prednasky
1416878 crw-rw----  1 root tty   2, 190 Oct  9 11:59 ptyae
1434762 lrwxrwxrwx  1 petr petr       9 Oct  9 11:58 sym_link -> hymna.mp3
meop35 exam #
```

Command `chmod`

- `chown 0644 file`
- Rights can be also defined by octal numbers:
  - 777 = (binary) 111 111 111 = rwx rwx rwx
- Default permissions driven by `umask` (022 = files ar 644, dirs are 755 =¿ i.e. it is subscracted from 666 and 777)
- Some filesystems support more advanced file-atributes (e.g. immutable, nodeletable, no copy, no write, ...) via commands `chattr` (and listing these attributes via `lstattr`)

KFA MFF UK
ÚČJF MFF UK

Faculty of Mathematics and Physics
CHARLES UNIVERSITY

NOFY077

Introduction to the Linux OS

Peter Huszár
KFA: Department of Atmospheric Physics

Pavel Řezníček
ÚČJF: Institute of particle and nuclear physics

September 30, 2025

Users and Groups

Users and Groups:

- /etc/passwd: users configutation: home, shell, title, uid
- /etc/shadow: users password hash
- /etc/groups: groups

Superusers:

- su / sudo (-i)
- su needs further config to run graphical applications
- visudo
- separate home dir in /root

Processes in Linux

Running applications and system processes can be inspected, killed, ... (aka TaskManager in Windows)

- `ps axuf | less`: inspect currently running processes (one-time)
- `top`, `htop`: online inspection of processes
  - Frequency of update
  - Search
  - Sort by memory usage instead of CPU
  - Can kill or change priority of a process
- `kill, killall, xkill`: kill process or application
- `nice`, `renice`: change application priority (0 by default, only root can increase it)
  - Priorities: -20 ... 19
  - `nice -12 large-job`: run large-job with priority 12
  - `nice --12 large-job`: run large-job with priority -12
  - `renice 17 -p 1134`: change priority of process with id 1134 to 17
- `cpulimit`: limit cpu usage of a process
- `ulimit`, `unlimit`: limit resources for a process (resp. in active shell): e.g. memory, number of opened files, ...
  - User can't override defaults in `/etc/security/limits.conf`
  - `ulimit -a`: see current limits
  - `unlimit`: set limits to the system defaults (not further limits for user)
  - `ulimit -Sv`: limit memory usage in kilobytes

KFA MFF UK

ÚČJF MFF UK

GUI processes

- `xrestop`: GUI processes
- `xkill`: GUI kill
- `wmctrl`: Change GUI window position, size, etc.
- `xev`: Monitor / debug key press (special keys)
- `xprop`: Properties of GUI window
- `xdotool`: Properties of GUI window, including emulation of keyboard/mouse

Other resources monitoring

- `iotop`: Disk usage per process
- `jnettop`: Network usage

Administration

Power and Services

In general: search for solutions of problems e.g. at *stackoverflow.com*, but be careful advices include modification of system files (configs are OK; reshufling binaries, libraries etc. NOT)

- `shutdown`, `poweroff`, `halt`, `reboot`,
- `hibernate`, `hibernate-ram`, `hibernate-disk`, `s2disk`, `s2ram`
- `systemctl` command:
  - `list-units`: list running system services
  - `stop`, `start`, `restart`: handle given service, only works till restart
  - `disable`, `enable`: enable/disable service completely (even after restart)

System configuration:

- in `/etc` directory
- most common default options in `/etc/default` directory

Logging of system events: in `/var/log` directory

- `syslog`, `daemon.log`, `messages`, `kern.log`, `debug`: system messages
- `auth.log`: info about logging of users (including virtual ones)
- `Xorg.0.log`: log of the GUI system
- Log files are usually backuped per month and started freshly again (`/etc/logrotate.conf`, `/etc/logrotate.d`)

KFA MFF UK

ÚČJF MFF UK

Handled by the Linux Kernel (includes HW drivers)

- Info about HW in pseudofiles: `/dev/*`, `/sys/*`, `/proc`
- Commands to list HW:
  - `lspci`: HW connected to the PCI bus
  - `lshw`: detailed info about all HW
  - `lsusb`: USB devices
  - `lsblk`: Block devices (disks)
  - `lsscsi`: SCSI devices (CD/DVD)
- Adding removing driver (= kernel module):
  - `lsmod`: list loaded kernel modules
  - `modinfo`: detailed info about kernel module
  - `modprobe`: add, or remove (`-r`) from kernel
  - `insmod`, rmmod: simple add / remove module from kernel
  - PS: Not all drivers are as separated modules, but can be builtin in the kernel =¿ the only way to disable them is via kernel option in GRUB boot loader
- Configuring kernel modules:
  - `/etc/modules`: force load of kernel modules not picked up automatically (special HW, very new HW, ...)
  - `/etc/modprobe.d`: add options to drivers, blacklist drivers

Network

Nowadays most systems use *NetworkManager* (NM) application to handle the net connection

- `/sbin/ifconfig`: Show info about network devices and connection (IP address, MAC address)
- NM allows complex configuration, including scripts
- NM recently generates random WiFi MAC address for security on public sites: might need to be disabled in corporate networks through `/etc/NetworkManager` settings

Useful commands:

- `ping`: Check remote host is online
- `traceroute`: Show full communication path to remote host
- `ssh`: connect to remote host via Secure Shell
- `ftp, sftp, scp, nfs`: copy files from/to remote host (see later lectures)
- `netstat`: info about connections, opened ports etc. (`netstat -natulp | less`)

Print

The linux system mostly rely on CUPS printing system:

- `/etc/cups`: configuration of the print client and server
- `localhost:631` in web-browser: WEB-based configuration of CUPS

However, recently some GUI applications can ignore CUPS settings and search for available printers by themselves...

Automatic tasks in background

CRON system:

- /etc/crontab: basic file to run tasks per hour/day/week/month
- /etc/cron.hourly
- /etc/cron.daily
- /etc/cron.weekly
- /etc/cron.monthly
- /etc/cron.d: more complicated rules

```
# /etc/cron.d/renew_prak0x: crontab entries for reweval of the prak0x user home directories
# Execute only during the period of the exercises (01.Oct - 20.Jan)
# TODO ?: Add entry in between day in case of 2 excercises per single day

SHELL=/bin/bash

# m  h   dom   mon              dow  user  command
32   01   *    OCT,NOV,DEC,JAN  SUN  root  /home/prak_template/bin/reboot.cron.sh
# NO!!! (studenti by po rebootu nenasli sva data !)
#@reboot                            root  /home/prak_template/bin/renew_prak0x.cron.sh
12   03   *    OCT,NOV,DEC       *   root  /home/prak_template/bin/renew_prak0x.cron.sh
12   03  1-20  JAN               *   root  /home/prak_template/bin/renew_prak0x.cron.sh
```

Packages (Ubuntu/Debian based distibitions)

APT system to download and install packages from repositories

- `apt-get`, `aptitude`: commands to run the tasks (install, remove, update package list)
- `/etc/apt`: configuration of the packaging system
- Temporary files (package list, downloaded installation files) in `/var/lib/apt/lists` and `/var/cache/apt` files
- Dependencies are flagged as auto-installed (`/var/lib/apt/extended_states`
- Simulate action (complicated upgrades): `apt-get -s`

DPKG is used to handle already installed packages:

- `-i`: install local package
- `-r`/`-P`: remove/purge installed package (purge = remove also config files)
- Allows to install even conflicting packages (`--force-depends`, `--force-conflicts`): use with care !
- Packages can ask for configuration options during installation process, reinvoke these questions by `dpkg-reconfigure` command
- Packages can provide "alternatives" for similar tasks: `update-alternatives` command to handle these selections

Linux on net is also target of attacks, especially through the ssh connection

- `sshguard`: service that stops ssh login after several failed attempts
- `rkhunter`: complex check of the most important system utilities
- `chkrootkit`: simple check of most known "viruses" in the system utilities
- `logcheck`: scans log files and reports suspicious activities

Many of these commands are set to run regularly and send reports by e-mail to the system admin (see `/etc/aliases` file)

Introduction to the Linux OS

Peter Huszár
KFA: Department of Atmospheric Physics

Pavel Řezníček
ÚČJF: Institute of particle and nuclear physics

November 10, 2025

Overview and Organization

Introduction to the Operation system Linux, focus on the command line, scripting, basic services and tools used in (not only) physics: tasks automation in data processing and modeling

Organization

- Graded Assessment (KZ): attendance to the lectures, worked out homeworks

Literature

- C. Herborth: Unix a Linux - Názorný průvodce, Computer Press, Praha, 2006
- D. J. Barrett: Linux - Kapesní přehled, Computer Press, Praha, 2006
- M. Sobell: Mistrovství v RedHat a Fedora Linux, Computer Press, Praha, 2006
- M. Sobell: Linux - praktický průvodce, Computer Press, Praha, 2002
- E. Siever: Linux v kostce, Computer Press, Praha, 1999
- Number of online sources...

Study materials and homeworks

- http://kfa.mff.cuni.cz/linux

KFA MFF UK

ÚČJF MFF UK

Syllabus

1. UNIX systems, history, installation, basic applications
2. Structure of the Linux OS, file systems, hierarchy of the file system
3. Command line, shells, remote access (ssh, ftp)
4. Processes and their administration, basic system commands, packages, printing
5. Users, file and directory permissions
6. Work with files and directories, file compression, links, partition
7. Text-file processing commands, redirection, pipeline
8. Regular expressions
9. Command line based text editors
10. User and system variables, output processing
11. Scripts: basic construction, conditionals, loops, functions, automation
12. Networking, server-client services: http, (s)ftp, scp, ssh, sshfs, nfs
13. Programming in Linux (examples of Fortran, C/C++, Python), version control systems, documents in Latex

File and Directory Manipulation

File and directory manipulation

In theory, files and directories (folders) can be:

- **Deleted** - remove file(s) and/or directory(-ies) from the filesystem
- **Copied** - copy file(s) and/or directory(-ies) to different part of the filesystem
- **Moved** - move file(s) and/or directory(-ies) to different part of the filesystem = copy and delete the original
- **Linked** - link file(s) and/or directory(-ies) to different part of the filesystem
- **Created** - create a new file(s) and/or directory(-ies)

i.e. operations on whole files, not on part of the data in the files (next lesson)
Wildcards – useful constructions to perform actions on more than one file/directory

Removes files and directories from the FS.

- For files

```
rm [OPTION]... [FILE]...
rm /my/file # one or many files
rm -i files/in/my/dir/file1 files/in/my/dir/file2 # interactive
rm -f /my/files # force removal
```

- For directories

```
rm -r [OPTION]... [DIRECTORY]...
rm -r /my/directory # one or many directories
rm -ri dirs/in/my/dir/directory1 dirs/in/my/dir/directory1 # interactive
rm -rf /my/dir # force removal
# combination of file(s) and directory(-ies)
rm -r files/in/my/dir/file1 dirs/in/my/dir/directory1
rmdir dir # for empty directories only
```

There is no undelete!!!

Once you delete with rm, your data are (mostly) gone! rm -rf / when logged as 'root' will delete everything ;-) For some filesystems, the rm keeps the content only delete the record.

Copy file(s) and/or directory(-ies) to different part of the filesystem or to different name

- For files

```
cp [OPTION]... [-T] SOURCE DEST
cp /my/file /another/file # DEST is a file
cp /my/file /another/directory # DEST is a directory
cp /my/file1 /my/file2 /another/directory # copy many file to destination directory
cp -i file1 file2 # interactive, if file2 already exists, it asks if to overwrite
```

- For directories

```
cp -r /my/dir1 /another/dir # if dir exists, it will contain dir1
 # otherwise dir is creared and will have the same content as dir1
# combination
cp -r /my/file /my/dir1 /dest/dir # dir has to exists
# if the DEST dir is the actual working directory (‘pwd‘) then
cp -r /my/file /my/dir1 .
cp -r /my/file /my/dir1 /my/dir2 # if dir2 does not exists, error is raised
```

- Does not show progress - use rsync for that purpose

Moves file(s) and/or directory(-ies) to different part of the filesystem or to different name

- For files

```
mv [OPTION]... SOURCE... DEST
mv /my/file /another/file # DEST is a file
mv /my/file /another/directory # DEST is a directory
mv /my/file1 /my/file2 /another/directory # move many file to destination directory
mv -i file1 file2 # interactive, if file2 already exists, it asks if to overwrite
```

- For directories

```
mv /my/dir1 /another/dir
# combination
mv /my/file /my/dir1 /dest/dir
# if the DEST dir is the actual working directory (‘pwd‘) then
mv /my/file /my/dir1 .
mv /my/file /my/dir1 /my/dir2 # if dir2 does not exists, it raises an error
```

Hard links

- hard links reference a physical(!!!) file location. Each file is actually a hardlink.
- ls -l 2nd column shows the number of links (at least 1!!!) (number of subdirs in case of dir)
- Links have actual file contents
- Removing any link, just reduces the link count, but doesn't affect other links.
- We cannot create a hard link for a directory to avoid recursive loops.
- If original file is removed then the link will still show the content of the file.
- Command to create a hard link is:

```
ln  [original filename] [link name]
ln  /my/file1 /other/file2
```

Soft links

- A soft link or symlink is similar to the file shortcut feature which is used in Windows Operating systems. Each soft points to the original file but not to the physical localtion of the data. As similar to hard links, any changes to the data in either file is reflected in the other. Soft links can be linked across different file systems, although if the original file is deleted or moved, the soft linked file will not work correctly (called hanging link).
- ls -l command shows all links with first column value 'l' and the link points to original file.
- Soft Link contains the path for original file and not the contents.
- Removing soft link doesn't affect anything but removing original file, the link becomes 'dangling' link which points to nonexistent file.
- A soft link can link to a directory.
- If you want to link files across the filesystems, you can only use soft links.

```
      ln -s [original filename] [link name]
      ln -s /my/file1 /other/file2 # link file to different filename
      ln -s /my/file /my/dir # link file to different directory
      ln -s /my/dir /my/other/dir # linking directories
ln -s /my/dir/or/files . # linking to the actual dir
```

Files

- A new file is created by many ways, some of them were seen on previous slides, others will be shown during next lessons
- Commands cp, mv, ln create new files (a symbolic link is a file too)
- Using touch - a new empty file is created:

```
touch myfile
```

- Redirecting the standard output (stdout) and standard error output (stderr) to a file (see next lesson)

```
command > /my/file
ls -l /etc > etc.content # the long listing of the
                         # /etc directory is saved in file etc.content
```

- Created by running programs (Text Editors, Photoeditors, Games etc.)

Directories

```
mkdir /my/dir # the dir /my MUST exist
mkdir -p /my/dir1/dir2/dir3/dir4 # if dir1 (and so on) does not exists,
                                 # it will create parent dirs (-p)
```

KFA MFF UK
ÚČJF MFF UK

Wildcards

Standard wildcards (also known as globbing patterns) are used by various command-line utilities to work with multiple files, especially usefull with file/directory manipulation.

- ? (question mark) – this can represent any single character (i.e. only one character). If you specified something at the command line like "hd?" GNU/Linux would look for hda, hdb, hdc and every other letter/number between a-z, 0-9 etc.
- * (asterisk) – this can represent any (including zero) number of characters. If you specified a "cd*" it would use "cda", "cdrom", "cdrecord" and anything that starts with "cd" also including "cd" itself.
- [](square brackets) – specifies a range a[a-c]d = aad,abd, acd, same as a[abc]d, [0-9] all numbers between 0 and 9. [A-Z]. etc.
- { } (curly brackets) – terms are separated by commas and each term must be the name of something or a wildcard, {abc, def, ghi} mean one of the strings inside
- [!]– [!0km] = any other character than 0, k or m
- Wildcards can be combined

```
ls -l /etc/??? # list all files/dirs in /etc with three character long names
cp -r ~/* /tmp # copy all files/dirs from the home directory to /tmp
               # (~ is a shortcut for the home directory)
rm  /etc/[nm]* # removes all config files from /etc starting with n or m
cp /tmp/[ab]{*.txt,*.doc,*.pdf} ~ # copy or files starting with a or b with
                                   # txt, doc and pdf extension from /tmp directory to home
```

File Manipulation/Wildcards - Excercises

- Excercise 1: create directories ~/cviceni/{dnesni datum} and copy all files starting with b or c from /usr
- Excercise 2: make symlink of all file ending with "t" in ~/cviceni/{dnesni datum} in a new directory ~/cviceni/{dnesni datum}/t
- Excercise 3: move all files from ~/cviceni/{dnesni datum} to ~/tmp and check the content of dir ~/cviceni/{dnesni datum}/t
- Excercise 4: remove dir ~/cviceni/{dnesni datum} and everything from ~/tmp

File and directory (de)compression

File and directory (de)compression

Prupose: creating one file to replace many files and to reduce the size of files/directories Commands to be learnt: zip, tar, gzip, bzip2, 7z ...

- ZIP - similar to WinZIP

```
# Compression
zip [options] zipfile files_list
zip myfile.zip  my_files
zip -r student_home.zip /home/student
zip -P password zipfile files_list # zipping with "password"
# Decompression
unzip myfiles.zip
unzip -l myfiles.zip # shows the zip content without unzipping
```

- TAR - create a tar archive (originally tape archive = tar). Merges whole directories into one file called 'tarball'. No compression!!!

```
# "Compression" - no compression in fact
tar -cvf myfilesdirs.tar /my/dirs /and/files # make a tarball from dirs and files
                                             # -c means compress
# Uncompress - untar
tar -xvf myfilesdirs.tar # -x means eXtract
# does not compresses
tar -cvf downloads.tar Downloads/
ls -l downloads.tar
976005120
du -cb Downloads
975353493
```

Tar can be combined with compression to reduce the size of the tarball file. Using compression algorithms from GZIP and BZIP2 utilities

- TAR with GZIP

```
# Compression
tar -zcvf myfiles.tar.gz /my/files /and/my/dirs # the extension is .tar.gz or .tgz
# Decompression
tar -zxvf  myfiles.tar.gz
tar -tvf myfiles.tar.gz # view contents
```

- TAR with BZIP2

```
# Compression
tar -jcvf myfiles.tar.bz2 /my/files /and/my/dirs # the extension is .tar.bz2 or .tbz2
# Decompression
tar -jxvf  myfiles.tar.bz2
tar -tvf myfiles.tar.bz2 # view contents
```

- gzip and bzip2 can be used directly for files

```
# Compression
gzip /my/file # creates file.gz
bzip2 /my/file # creates file.bz2
# Decompression
gunzip file.gz
bunzip2 file.bz2
```

- 7z - From man pages "7-Zip is a file archiver with the highest compression ratio. Supports LZMA, LZMA2, XZ, ZIP, Zip64, CAB, RAR (if the non-free p7zip-rar package is installed), ARJ, GZIP, BZIP2, TAR, CPIO, RPM, ISO, most filesystem images and DEB formats. Compression ratio in the new 7z format is 30-50% better than ratio in ZIP format."

```
# Compression
7z a dir.7z  /my/dir
# Decompression
7z e dir.7z
```

- RAR and UNRAR

```
# Compression
rar a myfiles.rar /my/files /my/dirs
# Decompression
unrar x myfiles.rar
```

- XZ - general-purpose data compression tool with command line syntax similar to gzip and bzip2, contained by 7z

```
# Compression
xz /my/files
# Decompression
xz -dk files.xz
```

- Excercise 1: Archive all files and directories in your home folder starting with . (dot) with tar+gzip
- Excercise 2: make a tarball from /etc directory - will it work for all the files
- Excercise 3: Compare the sizes of gzip, bzip2, zip, 7z, rar and xz for this file http://meop3.troja.mff.cuni.cz:8010/linux/netcdf/PHA03_STS.2007-2011DJF.nc
- Excercise 4: How to use 7z with password? Is it safe?

Homework 2

1. Create the "homeworks/HW2/myetc" directory(-ies) in your home directory with one(!) command and copy all files from /etc ending with "f" and containing a dot ".", or starting with "u" and the second character being "p" to this newly created directory.
2. Compress this newly created directory with zip and uncompress it again into a NEW directory homework/HW2/myetc_tmp (see man unzip)
3. List all files in /usr/bin that does not(!) start with "c" or "d", but they can start with "cd" or "dc". The command should have only one argument, so a smart combination of wildcards is needed.
4. Link (soft/sym link) all files starting with "d" and the second letter NOT BEING "x" from both /usr/bin and /sbin to homework/bin (using one command)
5. What happens if you move a softlink that is created using absolute and relative path to a different directory. Will it point to the right file in both cases?
6. It was not mentioned on the lecture, but what happens to input file if gzip (or bzip2) is used and how to change this behavior? (try it out and then check the man pages)

Introduction to the Linux OS

Peter Huszár
KFA: Department of Atmospheric Physics

Pavel Řezníček
ÚČJF: Institute of particle and nuclear physics

November 12, 2025

Overview and Organization

Introduction to the Operation system Linux, focus on the command line, scripting, basic services and tools used in (not only) physics: tasks automation in data processing and modeling

Organization

- Graded Assessment (KZ): attendance to the lectures, worked out homeworks

Literature

- C. Herborth: Unix a Linux - Názorný průvodce, Computer Press, Praha, 2006
- D. J. Barrett: Linux - Kapesní přehled, Computer Press, Praha, 2006
- M. Sobell: Mistrovství v RedHat a Fedora Linux, Computer Press, Praha, 2006
- M. Sobell: Linux - praktický průvodce, Computer Press, Praha, 2002
- E. Siever: Linux v kostce, Computer Press, Praha, 1999
- Number of online sources...

Study materials and homeworks

- http://kfa.mff.cuni.cz/linux

KFA MFF UK

ÚČJF MFF UK

Syllabus

1. UNIX systems, history, installation, basic applications
2. Structure of the Linux OS, file systems, hierarchy of the file system
3. Command line, shells, remote access (ssh, ftp)
4. Processes and their administration, basic system commands, packages, printing
5. Users, file and directory permissions
6. Work with files and directories, file compression, links, partition
7. Text-file processing commands, redirection, pipeline
8. Regular expressions
9. Command line based text editors
10. User and system variables, output processing
11. Scripts: basic construction, conditionals, loops, functions, automation
12. Networking, server-client services: http, (s)ftp, scp, ssh, sshfs, nfs
13. Programming in Linux (examples of Fortran, C/C++, Python), version control systems, documents in Latex

Disk/partition/filesystem handling utilities

Basic disk/partition handling utilities

Files/directories are structures physically or virtually written into harddrives/flashdrives, cdrom/dvdroms, tapes, virtual memory etc. These are usually divided into partitions which enables to separate logically distinct parts of the filesystem hierarchy structure. For example /home will be placed on different partition than /.

- Each parition has its own filesystem (FS)
- Partitions are 'mounted' to directories
- df – report file system disk space usage. It shows which partition is 'mounted' to which directory.
  - /dev/sda, /dev/sdb, /dev/sdc are 'device' files corresponding to individual devices
  - /dev/sda1, sda2 mean the partition number on the disk (i.e. each partition has a separate device file in /dev)
  - tmpfs is a temporal filesystem created in the RAM, so it behaves as a normal hraddisk, but when the computer is turned off, it is gone.
  - df -Th shows the type of the filesystems mounted (ext2, ext3, ext4, reiserfs, vfat, ntfs etc).
- lsblk – very useful utility to show the disk/partitions structure/filesystems and IDs of partitions
- hdparm – get/set SATA/IDE device parameters. Tool to use when it comes to tuning your hard disk or DVD drive, but it can also measure read speed, deliver valuable information about the device, change important drive settings, and even erase SSDs securely.

```
lsblk -f # get complete information about the disks/partitions/filesystems and disk UUIDs (see /etc/fstab)
hdparm -I /dev/sda # For all kind of information about the SATA/IDE disk
```

- FDISK - manipulate disk partition table. A very powerful utility to create,modify and delete partitions on a disk. USE WITH CAUTION AS YOU CAN EASILY DAMAGE YOUR EXISTING FILES/DIRECTORIES!!! More info on: https://www.tldp.org/HOWTO/Partition/fdisk_partitioning.html or man fdisk.

```
fdisk -l # shows all disks connected to the computer (needs root privileges)
fdisk /dev/sdb # prompt to manipulate with disk /dev/sdb, press m to get all the commands
```

- Once partitions are created on a disk, we can create different filesystems (Linux File System list https://static.javatpoint.com/linux/images/linux-file-system2.png, https://en.wikipedia.org/wiki/List_of_file_systems mkfs.filesystem

```
mkfs.ext4 /dev/sdb1 # create Ext4 filesystem on disk /dev/sdb and partition 1.
```

- fsck - check and repair a Linux filesystem

```
fsck.ext4 /dev/sdb1 # check and repairs the ext4 filesystem on disk /dev/sdb and partition 1.
```

In order to access the files/directories on the partitions, this partition has to be 'mounted'.

- mount - mount a filesystem

```
mount -t type device dir # The standard form of the mount command
# type is the filesystem type, device is the device file in /dev and dir is the directory where to mount
mount /dev/sdb1 /data # -t type may be omitted as the system will find it out automatically
mount -o ro /dev/sdb1 /data # "-o" = adding mount options, ro = read-only
```

- mounting ISO filesystem (CD/DVD *.iso image)

```
mount /path/to/image.iso /media/iso -o loop
```

- Mounting NFS - network filesystem

```
mount -t nfs 10.76.120.40:/volume1/d01 /home/nas01
```

- tmpfs - temporal/virtual filesystem represented in the RAM

```
mount -F tmpfs -o size=1G swap /mount/tmp # this takes 1G from the RAM and allocate it to /mnt/tmp
```

- unmounting filesystems

```
umount /mount/point
```

The `/etc/fstab` file is a system configuration file that contains all available disks, disk partitions, virtual disks and their options. Each file system is described on a separate line. Each line contains six fields separated by one or more spaces or tabs.
The `mount` command reads each line at system boot and mounts the disks according to the options given on the lines.

```
# <file system> <mount point> <type> <options> <dump> <pass>
# filesystem root disk
UUID=f3b12014-faea-49c8-9e9a-f397b002454d / ext4 errors=remount-ro 0 1
# boot partitions
UUID=515B-6053     /boot/efi vfat umask=0077  0 1
UUID=141ad68b-b7f4-4074-a89d-43801222d5fb none      swap sw   0       0
# 2T data disk
UUID=6f3ffadf-418f-4e15-baaf-93ca06ffaf54 /data   ext4  errors=remount-ro 0       0
# NAS1 and NAS2
10.76.120.40:/volume1/d01   /home/nas01 nfs defaults  0 0
10.76.120.40:/volume2/d02   /home/nas02 nfs rw   0 0
# cesnet high capacity storage center Jihlava
sshfs#huszarpet@ssh.du4.cesnet.cz:/tape_tape/archive/VO_cuni_mff_meteo/home/huszarpet \
            /mnt/cesnet fuse defaults,idmap=user,IdentityFile=/home/huszi/.ssh/id_rsa,allow_other 0 0
```

- Disks can be specified either by the device file e.g. `/dev/sda2`, however, this is not uniq and can change if disk are connected to the computer in a different order.
- A much safer option is to specify the disks by their UUID number (`lsblk -f /dev/sda2`)
- Further options: type = FS type; options = mount options (like ro for read only); dump = backup operation with 'dump' (0/1); pass = FS check with fsck (root always 1, other 2, nocheck = 0)
- to mount the "lines" of fstab (when changes made) use `mount -a`.

Excercise - disk/partition handling utilities

A simple but frequent example

Suppose we got an old (or new) disk and want to format it as a single partition to the `ext4` filesystem, then mount it to /mnt/mydisk.

- fdisk to repartition (delete old partitions)
- mkfs.ext4 to create a FS
- mount it (for one time use) and permanently via fstab

Text manipulation

Basic text manipulation commands

Commands to view and transform text(files); and to extract information about the text

- cat, more and less - to view text file contents (read-only!)

```
cat /my/text/file.txt # writes the contents of the file
                      # on the terminal - the standard output (stdout) and returns to prompt
                      # seemingly useless command in case of long files, but wait-for-it;-)
                      # (e.g. for putting two file ater "each other"  cat files1 file2 > file_final)
more /my/text/file.txt # view text by pages ("Space"), q or ctrl-c to quit
less /my/text/file.txt # view text by pages, PageUp/PageDown,Up/Down keys, Space -- 'q' to quit
                       # less is more than more and unlike more, reads only that part of the file
                       # which is shown
```

- head and tail - we are oftne interested in what the begining/ending of text file contains

```
head -n 20 /my/text/file.txt # prints the first 20 lines of the textfile
head -n -20 my/text/file.txt # prints all but the last 20 lines
tail -n 20 /my/text/file.txt # prints the last 20 lines of the textfile
tail -n +20 /my/text/file.txt # prints the last lines starting with line number 20
                              # if -c is used instead -n in both cases
                              # printing applies for bytes (first N/last N bytes)
```

- wc - word count (counts bytes/characters/words/lines)

```
wc  /my/text/files.txt # Print newline, word, and byte counts for each file
                       # if more files provided, print info for each and the total
wc -l /my/text/files.txt # number of lines
wc -w /my/text/files.txt # number of words
wc -c /my/text/files.txt # number of bytes, -m number of characters (bytes and chars are not the same)
```

Basic text manipulation commands (cont'd)

Commands to view and transform text(files)

- cut - cuts parts of each line of a text ("vertical" version of head/tail). It uses "deliminator", default is the tabulator. Very useful to extract columns in a text file, where columns are separated by a certain character (e.g. comma)

```
cut -d"{deliminator}" -f1 /my/text/file.txt # for each line prints everything before the first appearance
                                            # of the {deliminato} when it is the tabulator, -d is not import
cut -d":" -f1,3,5,10-15 # takes the first column, the 3rd, the 5th and than the 10-15.
                        # --complement, select the rest
                        # -d".", -d";", -d" ", -d"-", -d"a" etc.
cut -c X-Y              # cut from Xth to the Yth character on each line of the input file
```

- paste - line-by-line merge files "next" to each other. This requires a delimiter, tabulator is again the default one

```
paste -d"{deliminator}" /my/text/file1.txt /my/text/file1.txt  # for each line1/2 of file1/2
                                                               # prints line1/2 next to each other
                                                               # separated by {deliminator}
```

- fmt - simple text formatter

```
fmt /my/file.txt # by default puts all words  in a single line and prints
fmt -w 10 /my/file.txt # print the text in the specified width (does not wrap words!!!)
fmt -t file.txt # add indentation for the first line different from others
fmt -u file.txt # uses one space between words and two spaces after sentences for formatting
```

Basic text manipulation commands (cont'd 2)

- sort - sort lines of file according to dictionary,numerical value etc.

```
sort -d /my/files.txt # dictionary sort
sort -b /my/files.txt # ignore leading blank characters
sort -n /my/file-with-numbers.txt # numeric sort
sort -r /my/file.txt # reverse the result
sort -u /my/file.txt # print only unique lines
sort -k 2 /my/file.txt # sort according to the second "column"
```

- uniq - report/omit repeated lines (related to sort -u)

```
uniq -c /my/file.txt # prefix lines by the number of occurrences
uniq -d/-u /my/file.txt # prints only duplicated lines/unique lines
```

- rev - a trivial utility that reverses each line characterwise

```
rev /my/file.txt # the output will be the file with same size but all lines in
                 # reversed order (you can doubt the usefullness :-))
```

- join - join lines of two files on a common field (joins two "column" files with different columns as fields delimited by a delimiter based on values in selected columns)

```
join -t, -1 FILE1COLUMN -2 FILE2COLUMN FILE1 FILE2 # joins lines from both file based on
                                                   # the value of FILE1COLUMN equals FILE2COLUMN
join -t, -1 1 -2 2 file1 file2 # join lines of file1 with file2 where the value of columns 1 from
                               # 1st file and column 2 from 2nd file equal;-t is the delimiter (e.g. -t,)
```

- For join to be successfull, files must be sorted, unless –nocheck-order is used
- –headers will skip the first line on each file as header line

KFA MFF UK
UČJF MFF UK

Introduction to the Linux OS

Peter Huszár
KFA: Department of Atmospheric Physics

Pavel Řezníček
ÚČJF: Institute of particle and nuclear physics

November 18, 2025

Overview and Organization

Introduction to the Operation system Linux, focus on the command line, scripting, basic services and tools used in (not only) physics: tasks automation in data processing and modeling

Organization

- Graded Assessment (KZ): attendance to the lectures, worked out homeworks

Literature

- C. Herborth: Unix a Linux - Názorný průvodce, Computer Press, Praha, 2006
- D. J. Barrett: Linux - Kapesní přehled, Computer Press, Praha, 2006
- M. Sobell: Mistrovství v RedHat a Fedora Linux, Computer Press, Praha, 2006
- M. Sobell: Linux - praktický průvodce, Computer Press, Praha, 2002
- E. Siever: Linux v kostce, Computer Press, Praha, 1999
- Number of online sources...

Study materials and homeworks

- http://kfa.mff.cuni.cz/linux

KFA MFF UK

ÚČJF MFF UK

Syllabus

1. UNIX systems, history, installation, basic applications
2. Structure of the Linux OS, file systems, hierarchy of the file system
3. Command line, shells, remote access (ssh, ftp)
4. Processes and their administration, basic system commands, packages, printing
5. Users, file and directory permissions
6. Work with files and directories, file compression, links, partition
7. Text-file processing commands, redirection, pipeline
8. Regular expressions
9. Command line based text editors
10. User and system variables, output processing
11. Scripts: basic construction, conditionals, loops, functions, automation
12. Networking, server-client services: http, (s)ftp, scp, ssh, sshfs, nfs
13. Programming in Linux (examples of Fortran, C/C++, Python), version control systems, documents in Latex

Text manipulation

Basic text manipulation commands

Commands to view and transform text(files); and to extract information about the text

- cat, more and less - to view text file contents (read-only!)

```
cat /my/text/file.txt # writes the contents of the file
                      # on the terminal - the standard output (stdout) and returns to prompt
                      # seemingly useless command in case of long files, but wait-for-it;-)
                      # (e.g. for putting two file ater "each other"  cat files1 file2 > file_final)
more /my/text/file.txt # view text by pages ("Space"), q or ctrl-c to quit
less /my/text/file.txt # view text by pages, PageUp/PageDown,Up/Down keys, Space -- 'q' to quit
                       # less is more than more and unlike more, reads only that part of the file
                       # which is shown
```

- head and tail - we are oftne interested in what the begining/ending of text file contains

```
head -n 20 /my/text/file.txt # prints the first 20 lines of the textfile
head -n -20 my/text/file.txt # prints all but the last 20 lines
tail -n 20 /my/text/file.txt # prints the last 20 lines of the textfile
tail -n +20 /my/text/file.txt # prints the last lines starting with line number 20
                              # if -c is used instead -n in both cases
                              # printing applies for bytes (first N/last N bytes)
```

- wc - word count (counts bytes/characters/words/lines)

```
wc  /my/text/files.txt # Print newline, word, and byte counts for each file
                       # if more files provided, print info for each and the total
wc -l /my/text/files.txt # number of lines
wc -w /my/text/files.txt # number of words
wc -c /my/text/files.txt # number of bytes, -m number of characters (bytes and chars are not the same)
```

Basic text manipulation commands (cont'd)

Commands to view and transform text(files)

- cut - cuts parts of each line of a text ("vertical" version of head/tail). It uses "deliminator", default is the tabulator. Very useful to extract columns in a text file, where columns are separated by a certain character (e.g. comma)

```
cut -d"{deliminator}" -f1 /my/text/file.txt # for each line prints everything before the first appearance
                                            # of the {deliminato} when it is the tabulator, -d is not import
cut -d":" -f1,3,5,10-15 # takes the first column, the 3rd, the 5th and than the 10-15.
                        # --complement, select the rest
                        # -d".", -d";", -d" ", -d"-", -d"a" etc.
cut -c X-Y              # cut from Xth to the Yth character on each line of the input file
```

- paste - line-by-line merge files "next" to each other. This requires a delimiter, tabulator is again the default one

```
paste -d"{deliminator}" /my/text/file1.txt /my/text/file1.txt  # for each line1/2 of file1/2
                                                               # prints line1/2 next to each other
                                                               # separated by {deliminator}
```

- fmt - simple text formatter

```
fmt /my/file.txt # by default puts all words  in a single line and prints
fmt -w 10 /my/file.txt # print the text in the specified width (does not wrap words!!!)
fmt -t file.txt # add indentation for the first line different from others
fmt -u file.txt # uses one space between words and two spaces after sentences for formatting
```

Basic text manipulation commands (cont'd 2)

- sort - sort lines of file according to dictionary,numerical value etc.

```
sort -d /my/files.txt # dictionary sort
sort -b /my/files.txt # ignore leading blank characters
sort -n /my/file-with-numbers.txt # numeric sort
sort -r /my/file.txt # reverse the result
sort -u /my/file.txt # print only unique lines
sort -k 2 /my/file.txt # sort according to the second "column"
```

- uniq - report/omit repeated lines (related to sort -u)

```
uniq -c /my/file.txt # prefix lines by the number of occurrences
uniq -d/-u /my/file.txt # prints only duplicated lines/unique lines
```

- rev - a trivial utility that reverses each line characterwise

```
rev /my/file.txt # the output will be the file with same size but all lines in
                 # reversed order (you can doubt the usefullness :-))
```

- join - join lines of two files on a common field (joins two "column" files with different columns as fields delimited by a delimiter based on values in selected columns)

```
join -t, -1 FILE1COLUMN -2 FILE2COLUMN FILE1 FILE2 # joins lines from both file based on
                                                   # the value of FILE1COLUMN equals FILE2COLUMN
join -t, -1 1 -2 2 file1 file2 # join lines of file1 with file2 where the value of columns 1 from
                               # 1st file and column 2 from 2nd file equal;-t is the delimiter (e.g. -t,)
```

- For join to be successfull, files must be sorted, unless –nocheck-order is used
- –headers will skip the first line on each file as header line

KFA MFF UK
UČJF MFF UK

Input/output redirection and command chaining

Input/output redirection

standard output, standard error output, standard input

When a linux command is launched, three data streams are relevant/created: `stdout`, `stderr`, `stdin`, the standard output, what we saw so far in the terminal, the standard error, what we also saw in the terminal if an error occured and standard input what is read in.

- Redirect the stdout and stderr to a file

```
any-linux-command [1]> /my/output.log # the stdout is saved to file /my/output.log,
                                      # if exists, rewrites it (.log is not neccesary!)
                                      # 1 is the default value, so does not need to be written
any-linux-command 2> /my/output_errors.log # the stderr is saved to file /my/output_errors.log
any-linux-command &> /my/output_all.log # the stdout+stderr is saved to file /my/output_all.log
any-linux-command 1> /my/output.log 2> /my/output_errors.log # split the output into two files
any-linux-command 2>&1 # everything comes out as stdout (can be redirected further)
any-linux-command >> /my/output.log # if output.log exists, it will append the output to existing content
                                    # can be used with 1,2,&
# a command without redirection is equivalent to any-linux-command 1> /dev/stdout  2> /dev/stderr
Flush the output: any-linux-command > /dev/null
```

- Redirecting from the standard input

```
any-linux-command <  /my/input # the command 'any-linux-command' will take the file
                               # /my/input as input (e.g. rev < /my/input.txt will
                               # reverse all lines of /my/input.txt and writes it out)
any-linux-command <( other_command ) # redirect the output of other command to any-linux-command
any-linux-command <  /my/input > output.log # you can combine redirection
# Redirect to file and the standard output? No problem for 'tee'.
command | tee output.log # puts the output of command in output.log
                         # and the terminal too. | is a pipe, see in next slides
```

- ; (semicolon) – execute commands in chain independently to the previous

```
command1 ; command2 ; command3 # first, command1 is executed and then command2, ...
```

- && (double AND) - logical "and" between commands. From now on, we will regard commands as logical expression which are evaluated TRUE, if executed without errors (i.e. is successfull), or FALSE, when executed with error.

```
command1 && command2 && command3 # if command1 is successfull (TRUE), command2 will be executed.
                                 # if command2 is successfull too, command3 is executed
                                 # if e.g. command1 is FALSE, than nor command2 neither command3 is executed
                                 # Why? False && anything && anything will be always false, and the Linux
                                 # command line environment tries to evaluate the whole expression but
    # at  the first FALSE it knows that the whole expression will be FALSE
    # too and stops the evaluation
```

- || (double vertical bar) - logical "or" between commands. Commands in a || sequence will be executed until the last successfull

```
command1 || command2 || command3 # if command1 is successfull (TRUE), command2 will
                                 # NOT be executed.
                                 # command3 will be executed ONLY IF commands 1 and 2 are FALSE
```

- ! command – negating the command

- & (single AND) – execute the command before in the background

```
command1 & command2 # command2 will be started immediately after command1 is started
                    # (because it runs in the background)
```

- Note: ctrl-z stops executing the running command, bg resumes it and sends it to background, i.e. command & == command + ctrl-z + bg
- command == ctrl-z + fg
- | - the pipe, one of the most important way combining commands. It is pipe because it serves as a "medium" for data stream. In particular, it takes the output from the command before the pipe and provides it as input for the command that follows the pipe.

```
command1 | command2 | command3 # the output of command1 goes as input for command2. The output of command2
                              # goes to command3 as input.
                              # if we did not have pipe:
                              # command1 > out1 && command2 < out1 > out2 && command3 < out2
```

- By default, stderr is not "piped" and is written out to the terminal. From BASH v4 use |&. Or use 2 >&1 | for older BASH.

- ( ) (paranthesis) – execute the command(s) in the paranthesis in a separate sub-shell. Way how to combine all the methods for command chaining.

```
( command1 && command2 ) || ( command3 || command4 )
# If command1 is TRUE and command2 is TRUE, the other commands are not executed
# if command1 is TRUE but command2 is FALSE, the command line tries to execute one of commands3/4
# if command1 is FALSE, does not execute command2, and tries to execute one of command3/4
# More complex, but in theory working construction as an example:
(( command1 && command2 ) &> log_c1+2.log  || ( command3 || ! command4 2> error_c4.log )&  )
```

- { }(braces) – Placing a list of commands between curly braces causes the list to be executed in the current shell context. No subshell is created. The semicolon (or newline) following list is required.

```
{ command1 && command2 ; } ||  command3
```

- Excercise 1: How to redirect the stdout to a file, the stderr to a different file and send the stderr to the terminal too (hint: 2>&1 > /dev/null)
- Excercise 2: Write out the second largest file from directory /usr/bin (hint: ls, head or tail. and pipe)
- Excercise 3: Count the number of letters in the name of the first user sorted alphabetically (hint: cut, sort, tail or head, wc)
- Excercise 4: What will be the output of: true || echo aaa && echo bbb
- Excercise 5: what will be the standard output of the following crazy command and the content of the out.log file? Without tying to write to the prompt...
- `(! date > /dev/null && cat /etc/passwd ) || ( ( ! ls -l ~ || head -n 1 /etc/passwd | cut -d: -f1 ) ... ... ||( df -h &> df.log) ) | tail -n 1 > out.log`

Faculty of Mathematics and Physics
CHARLES UNIVERSITY

NOFY077

Introduction to the Linux OS

Peter Huszár
KFA: Department of Atmospheric Physics

Pavel Řezníček
ÚČJF: Institute of particle and nuclear physics

November 25, 2025

Overview and Organization

Introduction to the Operation system Linux, focus on the command line, scripting, basic services and tools used in (not only) physics: tasks automation in data processing and modeling

Organization

- Graded Assessment (KZ): attendance to the lectures, worked out homeworks

Literature

- C. Herborth: Unix a Linux - Názorný průvodce, Computer Press, Praha, 2006
- D. J. Barrett: Linux - Kapesní přehled, Computer Press, Praha, 2006
- M. Sobell: Mistrovství v RedHat a Fedora Linux, Computer Press, Praha, 2006
- M. Sobell: Linux - praktický průvodce, Computer Press, Praha, 2002
- E. Siever: Linux v kostce, Computer Press, Praha, 1999
- Number of online sources...

Study materials and homeworks

- http://kfa.mff.cuni.cz/linux

KFA MFF UK

ÚČJF MFF UK

Syllabus

1. UNIX systems, history, installation, basic applications
2. Structure of the Linux OS, file systems, hierarchy of the file system
3. Command line, shells, remote access (ssh, ftp)
4. Processes and their administration, basic system commands, packages, printing
5. Users, file and directory permissions
6. Work with files and directories, file compression, links, partition
7. Text-file processing commands, redirection, pipeline
8. Regular expressions
9. Command line based text editors
10. User and system variables, output processing
11. Scripts: basic construction, conditionals, loops, functions, automation
12. Networking, server-client services: http, (s)ftp, scp, ssh, sshfs, nfs
13. Programming in Linux (examples of Fortran, C/C++, Python), version control systems, documents in Latex

Further essential commands

- echo - Write arguments to the standard output. very simple but useful command. Especially with pipe.

```
echo Our first echo command # will write the arguments to the stdout, including a \newline character.
echo -n No newline # newline is not added
echo '$+-/*.]!@*(#*' # if the arguments contain "special" characters,
                     # which have reserved meaning in the scripting language (bash)
                     # than the safest way is to put single quote(' ').
echo $HOME "$HOME" '$HOME' # a good example
# how to print out single quotes?
echo \'text in single quotes\' # backslash \ turns off the special meaning
echo $HOME \$HOME. # special meaning of $ turned off (see later)
```

- tr - translate or delete. An utility which takes the standard input (supplied via pipe) and either translates some characters (called SET) to others (e.g. small letters to capital ones) or deletes some characters from SET. Now we will see why 'cat' is so useful.

```
# translation
cat /my/file | tr "[a-z]" "[A-Z]" #  (or  tr "[:lower:]" "[:upper:]"), lower to upper case
# data stream can be generated by echo too (not only cat /some/file)
echo "text   with  spaces" | tr "[:blank:]" " " # all horizontal space changed to single space
echo "text with numbers 2019" | tr "[:digit:]" "?" # digits changed to "?"
echo "text with Xsome Wunusual Qletters" | tr "XQW" "O"
# deleting
echo "text with numbers 2019" | tr -d "[:digit:]" # delete all digits
                              #   -d "[:alpha:]" # delete all letters
                              #   -d "xyz" # deletes all occurences of x,y,z
```

- xarg - build and execute command lines from standard input. In its basic usage, takes the standard input and puts it as arguments to command it calls. Very powerful, see man xargs!

```
echo str1 str2 | xargs command # this is equivalent to 'command str1 str2'
echo /dev/sda1 /dev/sda2 | xargs df -h # prints the size and free space on the
                                       # two indicated device (partitions)
                                       # can be useful when having a file with
                                       # each line as future argument for a command
```

- awk - scans patterns in the input and processes it. AWK is a separate programming language callable from command line. We will learn only a few basic usages. For more info e.g. http://www.grymoire.com/Unix/Awk.html.

```
# suppose to have a file with columns separated by unspecified whitespace (sometimes tab or multiple space)
# and one wants to take only some of the columns
# cut -d" " will not work, because it considers a specific delimiter
cat /my/file/with/columns.txt | awk '{print $1" "$3" "$4}' # this prints columns 1, 3 and 4 separated
                                                           # by single space
cat /my/file.txt | awk  'NR % 2 == 0' # print even lines
cat /my/file.txt | awk '/'pattern'/ {print $2}' # print the second columns
                                                # of lines containing the 'pattern'
```

Further commands

- dirname - returns the directory part of the path if the argument is a file. If the argument is directory, returns the parent directory's path

```
dirname /path/to/my/file # returns /path/to/my
dirname /path/to/my/dir # returns /path/to/my
dirname relative/path/to/something # returns  relative/path/to
```

- basename - returns the file part of the path, or the last dir, if the path is for directory. Optionally removes the suffix from filename.

```
basename /path/to/my/file # returns file
basename /path/to/my/dir  # returns dir
# with two arguments, 2nd as suffix to remove
basename /path/to/my/file.txt .txt #  returns file
```

KFA MFF UK

ÚČJF MFF UK

Command line calculators

- expr - the most common expression evaluator used for integer and logical operations. See man expr for the full list of operators.

```
expr \( 100 + 1 - \( 50 / 10 \) \) % 7 \* 1000
    # spaces between relation and numbers
    # (,),* must be backslashed because they have a different meaning in bash
expr \( 100 \> 101 \) \& \( "xyz" = "xy" \) # >,<,& must be backslashed too in logical expressions
```

- bc - An arbitrary precision calculator language. Much more capable than expr because you can set arbitrary precision. The default is integer precision. Moreover, life is easier with bc as no backslashes needed.

```
echo " expression " | bc # expression can be aritmetic or logical one
echo "((1+2)*3/4)-15+sqrt(100)" | bc # sqrt is  builtin function
-3
# to use the matlib (extended support of builtin functions and float point precision)
# use bc -l
echo "s(2)*c(2) / 2.0" | bc -l # s - sinus, c - cosinus (see man bc for others)
-0.18920062382698206283
# use bc for number base conversion
echo "obase=2;100" | bc -l # Convert Decimal to Binary
echo "obase=16;ibase=2;1011011010000101010110" | bc -l # Convert Binary to Hexadecimal
echo "a=1;b=2;a*b" | bc # user can define variables
```

- apcalc - another Arbitrary Precision CALCulator - the most powerful with much more builtin functions. The number of digits in the output is unlimited (limited only by the computer's internal memory).

```
echo "2**1000" | calc -pd # ** means exponent, power of 1000 - to demonstrate the unlimited precision/digits
echo "comb(100,4)" | calc -pd # combinatorial number 100!/4!(100-4)! # some more "exotic" functions
```

  List of apcalc builtin functions: "calc help builtin"

- Simple aritmentic expression as variables (see later in details)

```
var=$(( 10*80-5+1+(2+3) )) # This evaluates the expression in the double-paranthesis ((expression))
                           # and saves the value in variable 'var' (bash variables to come later)
```

Date/time related commands

Date/time related commands

Commands to work with datetime values/calendars

- cal - displays a calendar

```
cal # displays the calendar for the actual month
cal 2025 # displays the calendar for 2025
cal 11 2025 # displays the calendar for November 2025
```

- date - a date/time "calculator"

```
date # shows the current date in the default format
date +"Today date is %Y-%m-%d, and time is: %H:%M:%S" # see man date
date --date="1620-11-08" +%A # info o konkretnim datumu
date --date="-100000 days" # what happened 100000 days ago
date --date="+1000000 hours" # what is the date 1000 000 hours from now
# it can handle complicated date/time calculations
date --date="2000-01-01 12:00 +10 days -1000 seconds + 9898 hours - 1 year" +"%Y-%m-%d,%H:%M:%S"
```

- time - an utility to measure the amount of time it takes a program to execute. It also measures CPU usage and displays statistics.

```
time command arguments # basic usage
time find ~ -name '*.pdf' # this will find all pdf files in the home directory
                          # and the time command will write out how long it took
```

File/directory search

Search commands

Commands to search files and directories

- locate - search files based on a pre-built database by updatedb. Locate searches the whole directory tree but search in a database of files which is updated e.g. once-per-week.

```
locate name # locate all files with 'name' in their name ()
locate -c name # prints the number of found files
locate -e name # prints only those files which really exists in the moment of search.
```

- find - a powerfull search engine for files and directories with the possibility to logically combine search criteria

```
# basic usage
find /my/dir "search criteria" # search files/directories in /my/dir (and deeper)
                               # based on the search criteria
find /my/dir -name '*jpg'      # search based on name, files ending with jpg.
find /my/dir -type d -name 'a*' # search directories starting with "a" (or files only -type f)
find /my/dir -empty # find all empty files/directories
find /my/dir -perm 664 # find files/directories with permissions 664 (-perm u+rw)
find /my/dir -user student # find files/directories with "student" as owner
find /my/dir -size 5M # files larger than 5Megabytes
# you can combine search criteria
find /my/dir -size 5M -and -perm 664 -or  -empty -and -not -name '*jpg' # -or,-and,-not
```

Further actions on found items

```
find /my/dir "search criteria" -exec rm -rf {} \; # this will find files/dirs and applies
                                                  # the command after the -exec for each file found
                                                  #  (one-by-one!!)
find /my/dir -iname "*dvorak*mp3" -exec mplayer {} \; # listen to all dvorak mp3-s, mplayer is started
                                                      # separately for each file found
find /my/dir "search criteria" -exec command {} + # start the command only once for all files as parameters
find /my/dir -iname "*dvorak*mp3" -exec mplayer {} + # mplayer is started once and plays all files found
```

- whereis/which - find the whole path to the called binary/command. Can be useful if one has multiple instalations of a program and the specific binary can be executed from different directory. In this case, it is good to know which one is executed.

```
whereis python3 # this will show the full path for the ls command
which   python3 # shows exactly which binary is executed if more "python3"-s are installed
```

Pattern search in texts - regular expressions

- grep - One of the most useful and versatile commands in a Linux terminal environment is the "grep" command. The name "grep" stands for "global regular expression print". This means that grep can be used to see if the input it receives matches a specified pattern.

```
cat /my/input/file(s) | grep "pattern" # this will print all lines with the word 'pattern'
 # This is of course equivalent to grep "pattern" /my/input/file(s)
cat /my/input/file(s) | grep --color "pattern" # occurences are 'colored'
cat /my/input/file(s) | grep -o "pattern" # print only matches (-c will print the count)
 # useful options
 # -i - case insensitive, -v invert search; -l -- prints only files with matches
 # -L - print files without match
```

- However, the real power of grep comes with the introduction of regular expressions!!!

We see that with grep, we can search for some characters, words, but what about more complicated patterns???
For example:

- words that start to/end/contain a specific set of letters
- words starting with capitals or having certain number of characters
- email addresses
- IP address
- special numbers (e.g. real numbers)
- specific parts of computer code
- webpage address ...
- The above examples cannot be searched with simple `grep "word" /my/file`
- The solution is "regular expressions"
- A quick example: regular expression and search for a valid email address within a textfile

```
grep -E ^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$ /my/file
```

Introduction to the Linux OS

Peter Huszár
KFA: Department of Atmospheric Physics

Pavel Řezníček
ÚČJF: Institute of particle and nuclear physics

December 2, 2025

Overview and Organization

Introduction to the Operation system Linux, focus on the command line, scripting, basic services and tools used in (not only) physics: tasks automation in data processing and modeling

Organization

- Graded Assessment (KZ): attendance to the lectures, worked out homeworks

Literature

- C. Herborth: Unix a Linux - Názorný průvodce, Computer Press, Praha, 2006
- D. J. Barrett: Linux - Kapesní přehled, Computer Press, Praha, 2006
- M. Sobell: Mistrovství v RedHat a Fedora Linux, Computer Press, Praha, 2006
- M. Sobell: Linux - praktický průvodce, Computer Press, Praha, 2002
- E. Siever: Linux v kostce, Computer Press, Praha, 1999
- Number of online sources...

Study materials and homeworks

- http://kfa.mff.cuni.cz/linux

KFA MFF UK

ÚČJF MFF UK

Syllabus

1. UNIX systems, history, installation, basic applications
2. Structure of the Linux OS, file systems, hierarchy of the file system
3. Command line, shells, remote access (ssh, ftp)
4. Processes and their administration, basic system commands, packages, printing
5. Users, file and directory permissions
6. Work with files and directories, file compression, links, partition
7. Text-file processing commands, redirection, pipeline
8. Regular expressions
9. Command line based text editors
10. User and system variables, output processing
11. Scripts: basic construction, conditionals, loops, functions, automation
12. Networking, server-client services: http, (s)ftp, scp, ssh, sshfs, nfs
13. Programming in Linux (examples of Fortran, C/C++, Python), version control systems, documents in Latex

Single character

pattern	meaning	example regexp	example matches
`.`	any (!!!) single character	`a.c`	aac akc aZc a?c a+c ...
`\`	turns off special character	`\.`	. (dot)
`[ ]`	any of the characters in brackets	`[+mFf2019!]`	any of m,F,f,2,1,0,9,+,!
`-`	any character within the range	`[a-zA-Z3-6]`	any of a-z, A-Z, 3-6
`[^ ]`	negation of the above	`[^mFf2019]`	any except mFf2019
		`[^A-Z]`	any character except capital letters
Quantificators/repetition			
`?`	occurs 0x or 1x	`ab?c`	ac, abc
		`0[0-9]?1`	01, 011, 021, 031 ..
`*`	occurs arbitrary times (0-inf)	`ab*c`	ac, abbc, abbbbbbbc
		`0[0-9]*1`	01, 091, 011535451 ..
		`x.*x`	"xx", "x13 +-*x", "x 34-+ x 123 x"
`+`	occurs at least once	`ab+c`	abbc, abbbbbbbc
		`0[0-9]+1`	091, 011535451 ..
		`x.+x`	"x13 +-*x", "x 34-+ x 123 x"
`{n}`	occurs n-times	`ab{2}c`	abbc
		`0[0-9]{2}1`	0991, 0181 ..
		`x.{2}x`	"x13x", "x zx", "xxxx"
`{n,m}`	occurs n-m times	`ab{2,4}c`	abbc, abbbc
		`0[0-9]{2,4}1`	0991, 018231
		`x.{2,4}x`	"x13x", "x zx", "xxxxx"
		`x.{2,}x`	two or more occurences...

KFA MFF UK

ÚČJF MFF UK

Anchor characters

pattern	meaning	example regexp	example matches
`\<`	beginning of word	`\<[A-Z][a-z]+`	Paul, Judit, but not 09Tom
`\>`	end of word	`a\>`	all words ending on "a"
`^`	beginning of the line	`^[0-9].*`	all lines starting with a digit
`$`	end of the line	`*.[0-9]$`	all lines ending with a digit

- Selection
  - `(r1|r2|r3)` – any of the regex r1,2 or 3
  - E.g.: `([0-9]|[a-b]|xyz)` – 0,8,a,xyz
- Grouping
  - `(r1)+` – group with regex1 with at least 1 occurence
  - E.g.: `((r1)+(r2){2}){3}` – grouping regexps
  - E.g.: `([A-Z](\.|[a-z]+)){2,}` – (maybe) abbrevated names
- Remembering
  - `(r1)r2\1` – the match for the first regex will be saved and revoced by `\1`
  - E.g.: `([a-z])([a-z])([a-z])\3\2\1` – this finds all palindroms of length 6 (abccba, xyzzyx)

Regular expressions - Excercises

- Find all users with names starting with "r" (/etc/passwd)
- Find all latitude/longitude definition in AirBase-CZ-v8-stations.csv (regex for real numbers)
- Find all "acid" names in 'chemicals'
- In further, just use the echo "any_string the-test-string any_string2" — grep --color -Eo 'regex' to test, if the regex is correct
- Construct a regex for valid date in YYYYMMDD format (expect Feb has 28 days)
- Find regex for email address
- Find regex for whole sentences (Starts with capital letter, ends with one of '.?!')

sed - stream editor

- SED is the ultimate stream editor. It can substitue strings, add parts of text and delete part of text according to rules given by the users.
- SED can be used in two basic ways:

```
# 1
cat /my/input/file.txt | sed -e  'sed-script' # this generate output on stdout
# 2
sed -i 'sed-script' /my/input/file.txt # this will change the input file inline.
```

- The most used functionality of `sed` is string substitution

```
# on each line finds the first match for regex and replaces with 'string'
cat /my/file.txt | sed -e 's/regex/string/'
      # "/" here serves as a command delimiter
      # one can use different one too
cat /my/file.txt | sed -e 's:regex:string:'
cat /my/file.txt | sed -e 's/regex/string/2' # replace the 2nd occurence on line
cat /my/file.txt | sed -e 's/regex/string/g' # replace all occurence on the line
```

- In subtitution, one can remember the substituted string

```
echo "123 abc" | sed 's/[0-9][0-9]*/& &/'   # & will stand for the match (123)
                                             # -> 123 123 abc
sed -re 's/([a-z]*) ([a-z]*)/\2 \1/' # \1 and \2 memorize the searched string
                                     # and replace them with \2 \1
                                     # -r extended regex !!!
```

- By default, sed prints all lines in the output, not only those where the replacement occured

```
cat /my/file.txt | sed -n -re 's/regex/string/g p'
# p - print, sed will print the matched lines (not only replacing)
# -n, suppress printing lines, overwritten by "p" (effective for matched lines)
cat /my/file.txt | sed  -re 's/regex/string/g p' # will print matched lines 2x
```

- So far we have learn substitution and printing to all lines
- However, in sed, you can specify, for which line (or line range) to do it. We call it restriction.

- Specifying lines and line range - *restrictions*. The sed command than look like *sed restriction command*

```
cat /my/file.txt | sed -e '5 s/[0-9]/x/' # do the substitution on the 5th
cat /my/file.txt | sed -e '$ s/[0-9]/x/' # do the substitution on the last line ($)
...  | sed -e '10,30 p' # print lines 10-30 twice (with -n only those lines)
...  | sed -e '10,$ any-sed-command' # perform the command from the 10line till end
...  | sed -e '/pattern/ any-sed-command' #  perform the command on lines that match pattern
...  | sed -e '10,/pattern/ any-sed-command' # from line 10 to the 1st matched line (including)
...  | sed -e '/pattern1/,/pattern2/ any-sed-command' #
              # pattern1 switches the 'any-sed-command' on which is switched off by pattern2
```

- i - insert, before the restricted lines (if no restriction is present insert before each line)
- a - append, after the restricted lines (if no restriction is present appned after each line)

```
cat /my/file.txt | sed -e '10,30 i ???' # insert ??? before lines 10-30
cat /my/file.txt | sed -e '/pattern/ a ###' # append ### after lines matching the 'pattern'
```

- A superb SED howto: http://www.grymoire.com/Unix/Sed.html#uh-0

Faculty of Mathematics and Physics
CHARLES UNIVERSITY

NOFY077

Introduction to the Linux OS

Peter Huszár
KFA: Department of Atmospheric Physics

Pavel Řezníček
ÚČJF: Institute of particle and nuclear physics

December 9, 2025

Overview and Organization

Introduction to the Operation system Linux, focus on the command line, scripting, basic services and tools used in (not only) physics: tasks automation in data processing and modeling

Organization

- Graded Assessment (KZ): attendance to the lectures, worked out homeworks

Literature

- C. Herborth: Unix a Linux - Názorný průvodce, Computer Press, Praha, 2006
- D. J. Barrett: Linux - Kapesní přehled, Computer Press, Praha, 2006
- M. Sobell: Mistrovství v RedHat a Fedora Linux, Computer Press, Praha, 2006
- M. Sobell: Linux - praktický průvodce, Computer Press, Praha, 2002
- E. Siever: Linux v kostce, Computer Press, Praha, 1999
- Number of online sources...

Study materials and homeworks

- http://kfa.mff.cuni.cz/linux

KFA MFF UK

ÚČJF MFF UK

Syllabus

1. UNIX systems, history, installation, basic applications
2. Structure of the Linux OS, file systems, hierarchy of the file system
3. Command line, shells, remote access (ssh, ftp)
4. Processes and their administration, basic system commands, packages, printing
5. Users, file and directory permissions
6. Work with files and directories, file compression, links, partition
7. Text-file processing commands, redirection, pipeline
8. Regular expressions
9. Command line based text editors
10. User and system variables, output processing
11. Scripts: basic construction, conditionals, loops, functions, automation
12. Networking, server-client services: http, (s)ftp, scp, ssh, sshfs, nfs
13. Programming in Linux (examples of Fortran, C/C++, Python), version control systems, documents in Latex

Command Line Editors

- VIM works effectively with keyboard only. Usage of mouse is discouraged. Extensive syntax highlighting support.
- Openning a (new file)

```
vim /my/file # this open the file for writing or even creates it if nonexistent
```

- At this moment, user is in the *command mode*. I.e. any keyboard typed is interpreted as command.
- Let's start with simple editing the file - press `Insert` or `i`. The `-- INSERT --` appears on the bottom. This is the *insert mode*, the basic mode for typing text.
- Pressing `i` or `Insert` again takes one to the Replace mode. The text is replaced from the cursor.
- To save the changes, press Esc to exit the insert mode and return to the command mode. However, your file is not saved yet!!!
- To save the file, press `:` to enter the *last line mode*. Now, the last line became a command line and waits for a command.

```
:w #  write the changes to the file
:w different_file # 'save as' option to different file
:q # quit (wq - write changes and quit)
:x # write and quit
:q! # ignore the changes and quit
```

- To copy/move parts of text, press `v` or `V` from the command mode to enter the Visual mode.
- Now you can select the text you want to copy/move.
- When the selection is done, press `y` for copy or `d` for cut(move)
- Move the cursor to the place where you want to insert the copied text
- Press `p` for paste
- To copy/move one line press `yy` or `dd` and then `p` for eventual paste
- Simple `d` delete texts. `dd` deletes the line where the cursor is.

- To substitute text, enter the last time command

```
:%s/pattern/replace/g #  global replace of pattern to replace (% = each line)
                      # g = each occurence on line
:1,10 s/pattern/replace/ # replace "pattern" with "replace" on lines 1 to 10
```

- Undo the last change, `:u`
- Search some pattern `/pattern`
- :set nowrap - unwraps long lines
- :syntax on
- :10 - jump to the 10th line
- `Shift + G` jumps to the end of the file
- Very useful tab identation for e.g. Python (tab = 4 spaces):
  `:set tabstop=4 expandtab shiftwidth=4 softtabstop=4`
- `:help shiftwidth`

For almost complete VIM tutorial, see:
https://www.tutorialspoint.com/vim/index.htm

NANO (Nano is ANOther editor) is a small, free and friendly editor

- NANO is part of every Linux distribution's base instalation (which is not true for VIM)
- Easier control over editing. Using `Ctrl+` combination

```
GNU nano 2.5.3              File: myfile                          Modified
Our first text in Nano.
The second line.
^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify   ^C Cur Pos
^X Exit      ^R Read File ^\ Replace   ^U Uncut Text^T To Spell  ^_ Go To Line
```

- To save the just edited "buffer": `Ctrl+O` then enter the filename to save
- Insert content of different file: `Ctrl+R`, to delete the current line: `Ctrl+K`
- To exit: `Ctrl+X`, if one does not want to write the file, press `N`
- NANO supports syntax highlighting

```
ucitel@mypc:~$ ls /usr/share/nano/
# add this line to ~/.nanorc for every language
include /usr/share/nano/python.nanorc # e.g.
```

For almost complete NANO tutorial, see:
https://www.howtogeek.com/howto/42980/
the-beginners-guide-to-nano-the-linux-command-line-text-editor/

EMACS is a GNU project originally created by Richard Stallman. A very powerfull command line editor (with possible graphical interface)
Not covered with this course, for more information:
http://xahlee.info/emacs/emacs/emacs.html

Shell Variables

You can use variables as in any programming languages. There are no data types. A variable in bash can contain a number, a character, a string of characters. You have no need to declare a variable, just assigning a value to its reference will create it.

- Creation and assigning a variable

```
STR="Hello World!"
echo ${STR} # to refer to variable value, use $
MYVAR=1000000
echo ${MYVAR} MYVAR # this prints '1000000 MYVAR'
```

- There are system variables that control the behavior of the system/shell/GUI:
- The command `set` will list all the system/shell variables (and functions - see later)
- E.g. $HOME - the HOME directory, $LANGUAGE - the system language, $PS1 - the look of the prompt
- E.g. $PATH - the list of paths, where BASH looks for binary files
- User can define his own system variables by setting them in .bashrc (in your home directory)
- use `export MYVAR="value"` in order the variable behaves as global
- A variable can be in three states: defined with a value (MYVAR=value), defined with NULL value (MYVAR=) and unset. To unset a variable, use `unset MYVAR`.
- You can define new variables with existing ones: `NEWVAR=${OLDVAR1}${OLDVAR2}` (this example merges two strings)

- Bash enables numerous operations on variable value and gathering information on the variable (besides "asking" for its value)

```
${#MYVAR} # the length of variable value
${!prefix*} # prints all variables with their names starting with "prefix"
${MYVAR#pattern} # removes the match for pattern from the beginnig of MYVAR value
${MYVAR%pattern} # same as above but from the end of value
${MYVAR/pattern/string} # replaces pattern in MYVAR with string
${MYVAR^^} and ${MYVAR,,} # makes variables characters upper/lower case
```

- In the above examples, variables are "expanded" to a new value, which can be written out (with echo) or just saved to different variable(s).
- In the followig example, we rename all jpg files in a directory to JPG

```
for f in *.jpg; do # we will learn later
 echo "Renaming $f ..."
 mv $f ${f/.jpg/.JPG}
done
```

For a full list of variable expansion possibilities, see https://www.gnu.org/software/bash/manual/html_node/Shell-Parameter-Expansion.html

- The output of any command can be assigned to a variable as value in two syntactical way:

```
MYVAR=$( mycommand ) # preferred
MYVAR=‘ mycommand ‘ # not preferred
# the first way enables nesting:
MYVAR=$( mycommand $( anothercommand )   ) # mycommand takes the output of another command as argument
```

- eval - is a built-in Linux command which is used to execute arguments as a shell command. It combines arguments into a single string and uses it as an input to the shell and execute the commands.

```
MYVAR="ls -l /mydir"
eval $MYVAR
MYVAR='$'
MYVAR2=value
eval echo ${MYVAR}MYVAR2
```

- Use variable as a (part of) name for another variable.

```
MYVAR_A="123"
i=A
echo $((MYVAR_$i))
eval MYVAR_$i="456"
echo $MYVAR_A
```

- Bash supports 1-dimensional arrays with arbitrary integer indexing

```
MYARR=( 1 2 a b ahoj abc) # definition of an array, in this case indexing is starting from zero
echo ${MYARR[0]} -> 1 etc.
MYARR[100]=value # we can define/add arbitraty index (if MYARR exists, this new index is added)
MYARR=([7]=a [10]=b [100]=c) # possibility of defining arbitrary index
MYARR+=(newelement1 newelement2) # extenstion of array
```

- Different information can be retreived of arrays, including their length, list of elemets, list of indexes

```
echo ${MYARR[*]} # prints all the elements
... ${#MYARR[*]} # number of elements
... ${!MYARR[*]} # the list of indexes
```

Introduction to the Linux OS

Peter Huszár
KFA: Department of Atmospheric Physics

Pavel Řezníček
ÚČJF: Institute of particle and nuclear physics

December 16, 2025

Overview and Organization

Introduction to the Operation system Linux, focus on the command line, scripting, basic services and tools used in (not only) physics: tasks automation in data processing and modeling

Organization

- Graded Assessment (KZ): attendance to the lectures, worked out homeworks

Literature

- C. Herborth: Unix a Linux - Názorný průvodce, Computer Press, Praha, 2006
- D. J. Barrett: Linux - Kapesní přehled, Computer Press, Praha, 2006
- M. Sobell: Mistrovství v RedHat a Fedora Linux, Computer Press, Praha, 2006
- M. Sobell: Linux - praktický průvodce, Computer Press, Praha, 2002
- E. Siever: Linux v kostce, Computer Press, Praha, 1999
- Number of online sources...

Study materials and homeworks

- http://kfa.mff.cuni.cz/linux

KFA MFF UK

ÚČJF MFF UK

Syllabus

1. UNIX systems, history, installation, basic applications
2. Structure of the Linux OS, file systems, hierarchy of the file system
3. Command line, shells, remote access (ssh, ftp)
4. Processes and their administration, basic system commands, packages, printing
5. Users, file and directory permissions
6. Work with files and directories, file compression, links, partition
7. Text-file processing commands, redirection, pipeline
8. Regular expressions
9. Command line based text editors
10. User and system variables, output processing
11. Scripts: basic construction, conditionals, loops, functions, automation
12. Networking, server-client services: http, (s)ftp, scp, ssh, sshfs, nfs
13. Programming in Linux (examples of Fortran, C/C++, Python), version control systems, documents in Latex

Scripts

Scripts (reminder)

Sequence of commands to be processed.

- Allows functions, loops, conditions, call external commands
- Two ways how to run a script:
  - `./script.sh`: starts a new shell and runs the script in it (script file must be executable: `chmod +x ./script.sh`
  - `source ./script.sh` (or also `. ./script.sh`: runs the commands from the script one by one in the current shell → i.e. as if one would write them manually in the current terminal
  - `*.sh` used for *bash*-compatible scripts
  - `*.csh` used for *csh*-compatible scripts
- `#` are used for comments
- Special header "comment": `#!/usr/bin/zsh` instructs the script to be run by the `zsh` shell. Not only for shells, but also for interpreters like `python`
- `exit` [`number`] to quit script [and possibly return a *return code*]
  - Not needed at the very end of a script, it will end by itself
- `set -x` command inside a script instruct to show the commands being run by the script (i.e. for debugging)

Special characters (reminder)

- '' (single quotes) do no interpret special chars, while "" (double quotes) do
  - e.g. `echo '$i'` vs. `echo "$i"`
- ‘‘ (single backquotes) to insert output of command between the quotes
  - But better use `$(command)` instead
- ; (semicolon) allows to put more commands on single line
  - e.g. `echo "ahoj" ; echo "abc"`
- & at the end of line to run program in the background, while continuing in the script
- \ (backslash) cancels meaning of a special character
  - e.g. `echo "\$i"`
  - e.g. not to interpret space (`./script.sh ahoj\ abc` = `./script.sh "ahoj abc"`)
  - e.g. to allow quotes inside quotes (`echo "var = \"ahoj\""`)
  - at the end of line means wrapping - the line continues and the next line. Otherwise end-of-line is interpreted as delimiter of next command (equivalent of ;)

```
echo \
"ahoj"


for myfile in filename1 \
              filename2 \
              filename3 \
do
  echo $myfile
done
```

Script special variables

Input arguments

The arguments passed with script are accessiable via special variables

- ./script.sh arg1 arg2 arg3 ...

```
$1, $2, $3, ...   Individual arguments on command line (positional parameters)
$#                Number of command-line arguments
$*                All arguments on command line ("$1 $2 ...")
$@                All arguments on command line, individually quoted ("$1" "$2" ...)
$0                Command name
```

- Use `shift` command to "destroy" the first argument and shift the list of arguments to left, i.e. $1 becomes what was $2, $2 what was $3 etc., while original content of $1 is lost

Control of run commands in script (as well as in shell)

```
$?                Return value (exit code) of the last preceding command
$!                Process ID number (PID, see 'ps axuf' of the last preceding command
$$                Process ID number of the current process (the shell running the script)
```

Quick check of input variables content (script: $var replace by $1)

```
${var:-value}     Use var if set; otherwise, use value
${var:=value}     Use var if set; otherwise, use value and assign value to var
${var:?value}     Use var if set; otherwise, print value and exit
${var:+value}     Use value if var is set; otherwise, use nothing
```

Test expressions

`test EXPRESSION`: compare values, check file types, same as `[ EXPRESSION ]`
`[[ EXPRESSION ]]`: more versatile version of `[ ]`
`(( ))`: arithemtic tests (e.g. comparision of numbers)

- Return code `$?` is `0` if true, `1` if false

`( EXPRESSION )`	EXPRESSION is true
`! EXPRESSION`	EXPRESSION is false
`EXPRESSION1 -a EXPRESSION2`	both EXPRESSION1 and EXPRESSION2 are true
`EXPRESSION1 -o EXPRESSION2`	either EXPRESSION1 or EXPRESSION2 is true
`-n STRING`	the length of STRING is nonzero (also without -n)
`-z STRING`	the length of STRING is zero
`STRING1 = STRING2`	the strings are equal
`STRING1 != STRING2`	the strings are not equal
`INTEGER1 -eq INTEGER2`	INTEGER1 is equal to INTEGER2
`INTEGER1 -ge INTEGER2`	INTEGER1 is greater than or equal to INTEGER2
`INTEGER1 -gt INTEGER2`	INTEGER1 is greater than INTEGER2
`INTEGER1 -le INTEGER2`	INTEGER1 is less than or equal to INTEGER2
`INTEGER1 -lt INTEGER2`	INTEGER1 is less than INTEGER2
`INTEGER1 -ne INTEGER2`	INTEGER1 is not equal to INTEGER2

`FILE1 -nt FILE2`	FILE1 is newer (modification date) than FILE2
`FILE1 -ot FILE2`	FILE1 is older than FILE2
`-d FILE`	FILE exists and is a directory
`-e FILE`	FILE exists
`-f FILE`	FILE exists and is a regular file
`-L FILE`	FILE exists and is a symbolic link
`-r FILE`	FILE exists and read permission is granted
`-w FILE`	FILE exists and write permission is granted
`-x FILE`	FILE exists and execute (or search) permission is granted
`-s FILE`	FILE exists and has a size greater than zero

- .... and other file flags (ownership, types)

Expressions

- Arguments in EXPRESSION typically contain output of commands

```
test $(cat /etc/passwd | cut -d: -f1 | wc -l) -gt 100
test `cat /etc/passwd | cut -d: -f1 | wc -l` -gt 100
```

- Be careful to treat cases when arguments in expression can contain spaces, better always use "" for string arguments (works for integers too though), especially when argument is an output of command with not-well predictable result ! (e.g. filenames can contain spaces...)

```
i="ahoj abc"
test  $i  = "ahoj abc"      # results in: bash: [: too many arguments
test "$i" = "ahoj abc"      # OK
```

Tests (cont'd)

`[ ] vs [[ ]]`: Using the `[[ ... ]]` test construct, rather than `[ ... ]` can prevent many logic errors in scripts. For example, the `&&`, `||`, `<`, and `>` operators work within a `[[ ]]` test, despite giving an error within a `[ ]` construct. Arithmetic evaluation of octal or hexadecimal constants takes place automatically within a `[[ ... ]]` construct.

```
[[ -L $file && -f $file ]]   works in [[ ]]
[ -L "$file" ] && [ -f "$file" ]
[[ a < b ]]: lexicographical comparison
[ a \< b ]: Same as above. \ required or else does redirection like for any other command.
[[ a = a && b = b ]]: true, logical and
[ a = a && b = b ]: syntax error, && parsed as an AND command separator cmd1 && cmd2
[[ (a = a || a = b) && a = b ]] vs. [ ( a = a ) ]: syntax error, () is interpreted as a subshel
```

`(( EXPRESSION ))`

```
(( 5 > 4 ))
(( 5 == 5 ))
(( t = 40 < 45?7:11 ))   # C-style trinary operator.
echo "If 40 < 45, then t = 7, else t = 11."
```

Conditions - if/then/else

Use result of `test`

- Notation using square brackets `[ EXPRESSION ]`

```
if [ EXPRESSION ]
then
  command1
elif [ EXPRESSION ]
then
  command2
else
  command3
fi
```

```
if [ EXPRESSION ] ; then
  command1
elif [ EXPRESSION ] ; then
  command2
else
  command3
fi
```

- Short one-command condition using `&&` and/or `||`:

```
[ EXPRESSION ] && command1 || command2
```

- is equivalent to:

```
if [ EXPRESSION ] ; then command1 ; else command2 ; fi
```

Conditions - case

Equivalent of `if/then/elif/elif/.../else/fi` statements chain

- Can use shell pattern matching (e.g. `*`)
- Use `|` for OR of matches
- On match the sequence of commands is run till `;;`
- `*)` typically used for safety `else` with an error message that there was no match

```
case $varname in
  pattern1)
    command1
    ;;
  pattern2|pattern3|pattern4)
    command2
    ;;
  *)
    command_error_no_match
esac
```

Loops - while/until/do/done

Keep looping (un)till EXPRESSION is valid

- Assuming the arguments in the EXPRESSION are changing during the sequence of commands in the loop, thus allowing the loop to stop at some point
- Can immediately stop the loop with `break` command
- Can immediately jump to next iteration with `continue` command

While

Stop looping if EXPRESSION becomes false

```
while [ EXPRESSION ]
do
   commands
   if [ ... ] ; then break ; fi     # alternative way to stop the loop
done
```

Until

Stop looping when EXPRESSION becomes true

```
until [ EXPRESSION ]
do
   commands
   if [ ... ] ; then break ; fi     # alternative way to stop the loop
done
```

For cycle

Loop over predefined list of items

- The list of items to cycle over is space-separated
- Can immediately stop the loop with `break` command
- Can immediately jump to next iteration with `continue` command
- `seq 1 100` to generate list of indexes from 1 to 100

```
for var_i in item1 item2 item3
do
   commands
   if [ ... ] ; then break ; fi     # possible way to stop the loop prematurelly
done
```

Space separation in list

- Potentially dangerous when list contains items with space, e.g. weird filenames
- For files use `find` command instead of `for` cycle
- Or replace spaces by a defined string and inside the loop revert this replacement:

```
# Would not work for files with space
for i in $(ls -1) ; do
  echo $i
done

# Works:
for ii in $(ls -1 | sed 's, ,__mezera__,g') ; do
  i=$(echo $ii | sed 's,__mezera__, ,g')
  echo $i
done
# Works
find . -maxdepth 1 -name '*' -exec echo {} \;
# Works
find . -maxdepth 1 -name 'a*' | while read i ; do echo $i ; done
```

For cycle

Loop over predefined list of items - cont'd

- The list of items to cycle over can be defined alternatively like:

```
for i in {1..5};do echo $i ;done
# from BASH v4.0+,  {START..END..INCREMENT} syntax
for in {0..10..2};do echo $i ;done
# control the width of the loop item:
for i in {001..500};do echo $i ;done
# or combining with other character and multiple ranges
for i in a{001..500} {700..999};do echo $i ;done

## The C-style Bash for loop
for (( initializer; condition; step ))
for (( c=1; c<=5; c++ ));do echo $c ;done
```

Functions

Similar behaviour as in other programming languages

- Mostly to help organization/readabilty of the code
- Accept parameters, treated in similar way as input parameters of scripts (i.e. $1, $2, etc.)
- Output transfered via `echo` command or e.g. my modifying a "global" variable

```
x=0

myfunc() {
  for i in $@ ; do
    echo $i
  done
  x=1
}

echo $x
myfunc a bb cc 123
echo $x
x=0
str=‘myfunc dd ee‘ # x is not changed, myfunc is run in separate shell !
echo $str
echo $x
```

Automatic script options

Use of `getopt` command

- Colon `:` after option letter specifies that the option is expecting an argument

```
while getopts 'ha:' OPTION; do
  case "$OPTION" in
    h)
      echo "Option h (does not expect argument)"
      ;;

    a)
      echo "Option a with value \"$OPTARG\""
      ;;

    ?)
      echo "script usage: $(basename $0) [-h] [-a somevalue]" >&2
      exit 1
      ;;
  esac
done
shift "$(($OPTIND -1))"

echo "Remaining input arguments: $@"
```

Scripts - exercises

- Exercise 1: How to compare floating-point numbers ? Hint `bc -l`, `python -c ... exit,print`
- Exercise 2: Loop through all links in current directory (and sub-directories), check the file really exists (link is valid)
- Exercise 3: Store script input parameters into variables array. Iteratively destroy input parameters one by one and print the remaining on the screen (try all `for`, `while` and `until` loops)

Scripts - exercises

- Exercise 1: How to compare floating-point numbers ? Hint `bc -l`, `python -c ... exit,print`
- Exercise 2: Loop through all links in current directory (and sub-directories), check the file really exists (link is valid)
- Exercise 3: Store script input parameters into variables array. Iteratively destroy input parameters one by one and print the remaining on the screen (try all `for`, `while` and `until` loops)
- Exercise 4: For cycle to generate N random numbers (N=1000 if no argument passed to the script) and print the highest value. Hint: `$RANDOM`.
- Exercise 5: Select random 500 lines from mcData.txt (make sure the lines do not repeat)
- Exercise 6: Loop through archives `backup*`, search for files named `Invariant_masses.txt`, join their content with `mcData.txt` and remove duplicated lines
- Exercise 7: Batch analysis: script triggering a computation jobs
  - Job = generate 100 random numbers with given seed in `rnd.txt`, sleep 1 sec between the generation
  - Run max. 5 jobs in parallel
  - Allow the script to run more than once without breaking the rule above
  - Hint: use flag-files or `ps axuf` to find out which jobs are running, which are finished

Scripts running after logout

nohup

- Most simple way to keep process running after logout (or killing mother terminal)
- Syntax: `nohup command arguments`
- Output goes to `nohup.out` file

screen

- More complex system, behaving as a virtual terminal, allowing to:
  - Detach and re-attach to running session
  - After re-attaching one can see the output of the session
  - Works better on remote machines with complex authentication
  - Can name sessions
  - `screen` allows to send command to a running detached session
- `screen` to start a session
  - *CTRL-a d* to detach from session
  - `screen -list` to list sessions (either attached or detached)
  - `screen -r` to attach to a sessions

tmux

- Similar functionality to `screen`, but more actively developed
- `tmux` to start a session
  - *CTRL-b d* to detach from sessions
  - *tmux ls* to list sessions
  - *tmux attach* to attach

Crontab - Reminder

CRON system:

- /etc/crontab: basic file to run tasks per hour/day/week/month
- /etc/cron.hourly
- /etc/cron.daily
- /etc/cron.weekly
- /etc/cron.monthly
- /etc/cron.d: more complicated rules

```
# /etc/cron.d/renew_prak0x: crontab entries for reweval of the prak0x user home directories
# Execute only during the period of the exercises (01.Oct - 20.Jan)
# TODO ?: Add entry in between day in case of 2 excercises per single day

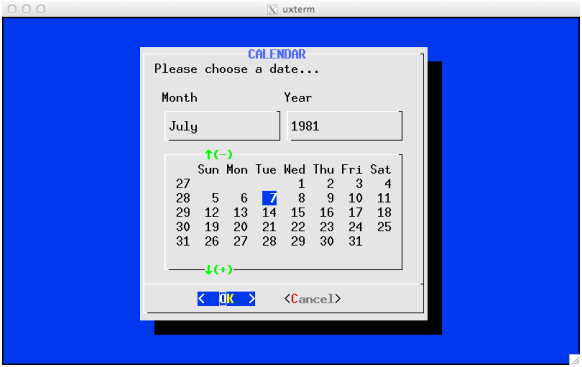
SHELL=/bin/bash

# m  h   dom   mon              dow  user  command
32   01   *    OCT,NOV,DEC,JAN  SUN  root  /home/prak_template/bin/reboot.cron.sh
# NO!!! (studenti by po rebootu nenasli sva data !)
#@reboot                            root  /home/prak_template/bin/renew_prak0x.cron.sh
12   03   *    OCT,NOV,DEC       *   root  /home/prak_template/bin/renew_prak0x.cron.sh
12   03  1-20  JAN               *   root  /home/prak_template/bin/renew_prak0x.cron.sh
```

Graphical interface to scripts

Programs to easily create simple graphics interfaces:

- Calendar
- File selection
- Forms
- Messages
- Lists
- Progress bars
- Text entry



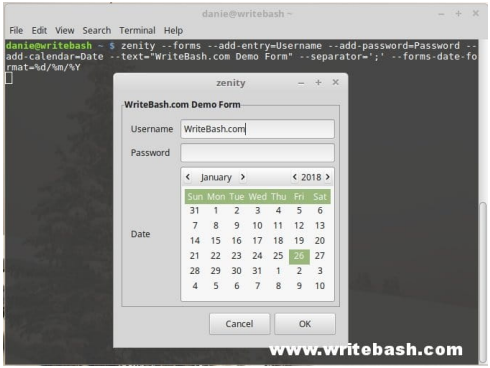
dialog

- Terminal-based graphics
- See number of exampes in `/usr/share/doc/dialog/examples`

Graphical interface to scripts

Programs to easily create simple graphics interfaces:

- Calendar
- File selection
- Forms
- Messages
- Lists
- Progress bars
- Text entry



dialog

- Terminal-based graphics
- See number of exampes in `/usr/share/doc/dialog/examples`

zenity / gdialog

- Graphical windows (GTK)
- See examples at https://help.gnome.org/users/zenity/3.32/

Faculty of Mathematics and Physics
CHARLES UNIVERSITY

NOFY077

Introduction to the Linux OS

Peter Huszár
KFA: Department of Atmospheric Physics

Pavel Řezníček
ÚČJF: Institute of particle and nuclear physics

September 30, 2025

Overview and Organization

Introduction to the Operation system Linux, focus on the command line, scripting, basic services and tools used in (not only) physics: tasks automation in data processing and modeling

Organization

- Graded Assessment (KZ): attendance to the lectures, worked out homeworks

Literature

- C. Herborth: Unix a Linux - Názorný průvodce, Computer Press, Praha, 2006
- D. J. Barrett: Linux - Kapesní přehled, Computer Press, Praha, 2006
- M. Sobell: Mistrovství v RedHat a Fedora Linux, Computer Press, Praha, 2006
- M. Sobell: Linux - praktický průvodce, Computer Press, Praha, 2002
- E. Siever: Linux v kostce, Computer Press, Praha, 1999
- Number of online sources...

Study materials and homeworks

- http://kfa.mff.cuni.cz/linux

KFA MFF UK

ÚČJF MFF UK

Syllabus

1. UNIX systems, history, installation, basic applications
2. Structure of the Linux OS, file systems, hierarchy of the file system
3. Command line, shells, remote access (ssh, ftp)
4. Processes and their administration, basic system commands, packages, printing
5. Users, file and directory permissions
6. Work with files and directories, file compression, links, partition
7. Text-file processing commands, redirection, pipeline
8. Regular expressions
9. Command line based text editors
10. User and system variables, output processing
11. Scripts: basic construction, conditionals, loops, functions, automation
12. Networking, server-client services: http, (s)ftp, scp, ssh, sshfs, nfs
13. Programming in Linux (examples of Fortran, C/C++, Python), version control systems, documents in Latex

Crontab - Reminder

CRON system:

- /etc/crontab: basic file to run tasks per hour/day/week/month
- /etc/cron.hourly
- /etc/cron.daily
- /etc/cron.weekly
- /etc/cron.monthly
- /etc/cron.d: more complicated rules

```
# /etc/cron.d/renew_prak0x: crontab entries for reweval of the prak0x user home directories
# Execute only during the period of the exercises (01.Oct - 20.Jan)
# TODO ?: Add entry in between day in case of 2 excercises per single day

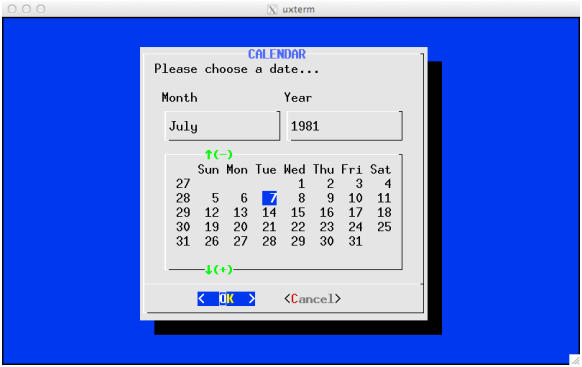
SHELL=/bin/bash

# m  h   dom   mon              dow  user  command
32   01   *    OCT,NOV,DEC,JAN  SUN  root  /home/prak_template/bin/reboot.cron.sh
# NO!!! (studenti by po rebootu nenasli sva data !)
#@reboot                            root  /home/prak_template/bin/renew_prak0x.cron.sh
12   03   *    OCT,NOV,DEC       *   root  /home/prak_template/bin/renew_prak0x.cron.sh
12   03  1-20  JAN               *   root  /home/prak_template/bin/renew_prak0x.cron.sh
```

Graphical interface to scripts

Programs to easily create simple graphics interfaces:

- Calendar
- File selection
- Forms
- Messages
- Lists
- Progress bars
- Text entry



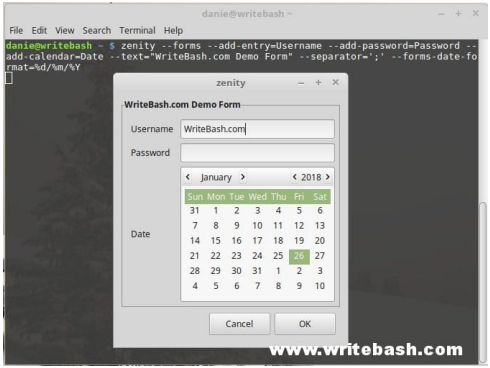
dialog

- Terminal-based graphics
- See number of exampes in /usr/share/doc/dialog/examples

Graphical interface to scripts

Programs to easily create simple graphics interfaces:

- Calendar
- File selection
- Forms
- Messages
- Lists
- Progress bars
- Text entry



dialog

- Terminal-based graphics
- See number of exampes in `/usr/share/doc/dialog/examples`

zenity / gdialog

- Graphical windows (GTK)
- See examples at https://help.gnome.org/users/zenity/3.32/

Conversions

Batch conversions

Mostly, conversions between various file-formats can be done with graphical applications. The next slides list commands for command-line conversions, for use in scripts (multiple files conversions etc.)

- Figures
- Video files
- PDF files manipulations
- Office documents, txt files
- Number of `xxx2yyy` commands/packages on linux for various file formats...

Conversions of figures

- Graphical applications: `Gimp` for bitmaps, `Inkscape` for vector graphics
- Batch: `convert` command from `Imagemagick` package
  - See `https://www.imagemagick.org/Usage/`
  - image format
  - resize, rotate, ...
  - image quality
  - animations
  - text → image
- `identify -verbose file` to dump image information

Conversions of videos

- Number of graphical applications: `Openshot`, `Avidemux`, ...
- Batch: `ffmpeg`, `mencoder`
  - audio/video formats
  - resolution, bitrate
- `ffmpeg -i file` to dump video file information

PDF files

- Beside commercial Adobe SW only basic operations are possible
- Annotations (graphical applications): `okular`, `Xournal++`
- Editing graphical applications: `Inkscape`, `PDFedit`
- Batch: `pdftk`, `ghostscript`
  - PDF $\leftrightarrow$ PS conversions (postscript files - partly editable in text editors)
  - splitting / joining pdf files
  - handling of embedded fonts
- `pdfimage` to extract images from PDF files
- `pdfcrop` to crop white space around PDF file

Text and office files

- WIN ↔ UNIX text files conversion (end-of-line characters): `dos2unix`, `unix2dos`
- Text to postscript file: `a2ps input.txt -o output.ps`
- Text files encoding: `iconv -f "UTF-8" -t "CP1250" input.txt -o output.txt`
- Libreoffice formats: `unoconv -f odt input.doc`

Client-Server Services

Services for networking

- `ssh` for remote login to a command line
- `kerberos` for more secure and complex remote logins
- `telnet` old, now insecure, remote login to a command line
- `scp` to copy files via ssh
- `ftp` to copy files between PCs
  - `sftp` as it secure replacement (uses `ssh`, but emulates `ftp` commands)
- Web-server
- Command line tools to access web-pages
  - `wget`, `curl`: downloading www-page source
  - `lynx`, `elinks`: text-based www browsers, i.e. translating www-page source to the www-page
  - Limited usage on web-pages with e.g. java scripts etc.

Remote login to a command line, remote file transfer via `scp`

- SSH allows for password-less login using public/private keys
  - Create key pair (of type rsa) via `ssh-keygen -t rsa`
  - Key will be by default saved to `/home/$USER/.ssh/id_rsa`
  - Will ask for password (not the login one, but an optional new one guarding the key)
  - Copy public key `/home/$USER/.ssh/id_rsa.pub` to `/.ssh/authorized_keys` file on the remote server(s)
  - If password for the key was set, one will be asked for it on login on the local PC
- Comments on SCP
  - Works like `cp` command
  - To interpret wildcards on remote machine, use `""` or backslash `\`
  - Syntax or remote path: `username@machine:path`