Linux, command line & MetaCentrum

Use of Linux command line not only for MetaCentrum of CESNET

Vojtěch Zeisek

Department of Botany, Faculty of Science, Charles University in Prague
Institute of Botany, Czech Academy of Sciences, Průhonice
https://trapa.cz/, zeisek@natur.cuni.cz

January 28 and 29, 2016

Outline I

Outline II

Outline III

The course information

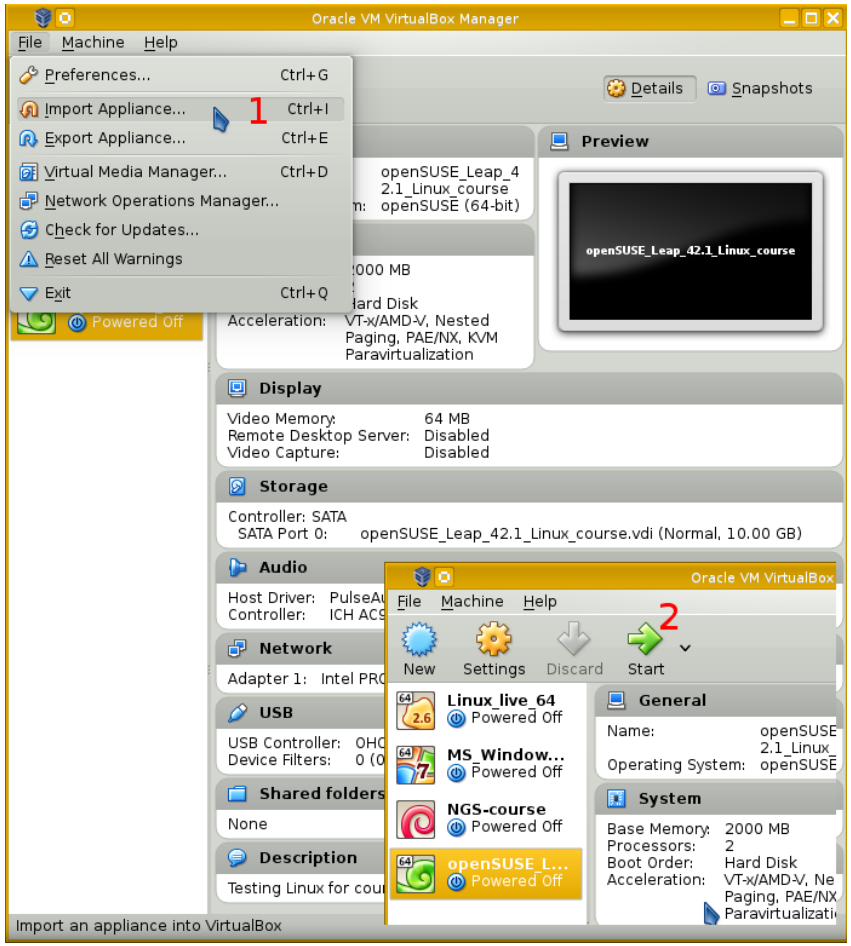
- The course page: https://trapa.cz/en/course-linux-command-line-2016
  - Česky: https://trapa.cz/cs/kurz-prikazove-radky-linuxu-2016
- Subject in SIS: https://is.cuni.cz/studium/eng/predmety/index.php?do=predmet&kod=MB120C17
  - Česky: https://is.cuni.cz/studium/predmety/index.php?do=predmet&kod=MB120C17
  - For students having subscribed the subject, active participation and presence whole course is required
- Working version is available at https://github.com/V-Z/course-r-mol-data – feel free to contribute, request new parts or report bugs

Materials to help you...

- Download the presentation https://trapa.cz/sites/default/linuxcourse/linux_bash_metacentrum_course.pdf
- Download the scripts and toy data https://trapa.cz/sites/default/linuxcourse/scripts.zip
  - Note: Open the scripts in some good text editor – showing syntax highlight, line numbers, etc. (NO Windows notepad); the file is in UTF-8 encoding and with UNIX end of lines (so that too silly programs like Windows notepad won't be able to open it correctly)

Virtual machine for learning

- If you do not have Linux installed, download VirtualBox from https://www.virtualbox.org/
- Download openSUSE Leap 42.1 Linux distribution for this course from ftp://botany.natur.cuni.cz/openSUSE_Linux_course.ova (1.9 GB)
- Launch VirtualBox and go to menu **File** | **Import appliance...** to import it. When do, launch it (**Start**)



What it is UNIX, Linux and GNU... I

- UNIX
  - Originally developed in Bell labs of AT&T in 1696, written in C, since then huge radiation, hybridization, HGT, ...
  - Trademark – only systems passing certain conditions (paid certification) can be called "UNIX" – Solaris, HP-UX, AIX, ... – commercial systems for big servers
  - main principles: simple, multitasking, hierarchical, network, for more users (takes cares about permissions etc.), configuration written in plain text files, important relationships among applications (generally one application = one task – they are chained), work primarily with text, has kernel and API (interface to communicate with the rest of the system)
  - unix-line (UN*X) – systems compatible with UNIX (Linux, BSD and its variants, Mac OS X, ...)
    - Mainly open-source (UNIX is commonly commercial – source code is not available, but specification is)

What it is UNIX, Linux and GNU... II

- Nowadays prevailing over "old" UNIX systems, used in many devices from tiny embedded toys to huge data centers
- Try to provide same service as paid systems, but (mostly) for free

- Many courts about copyrights, parts of code, patents, ... – USA allow software patents, EU not – GNU, Linux, BSD, ... try to ensure to have only code not covered by any patents to avoid possible courts

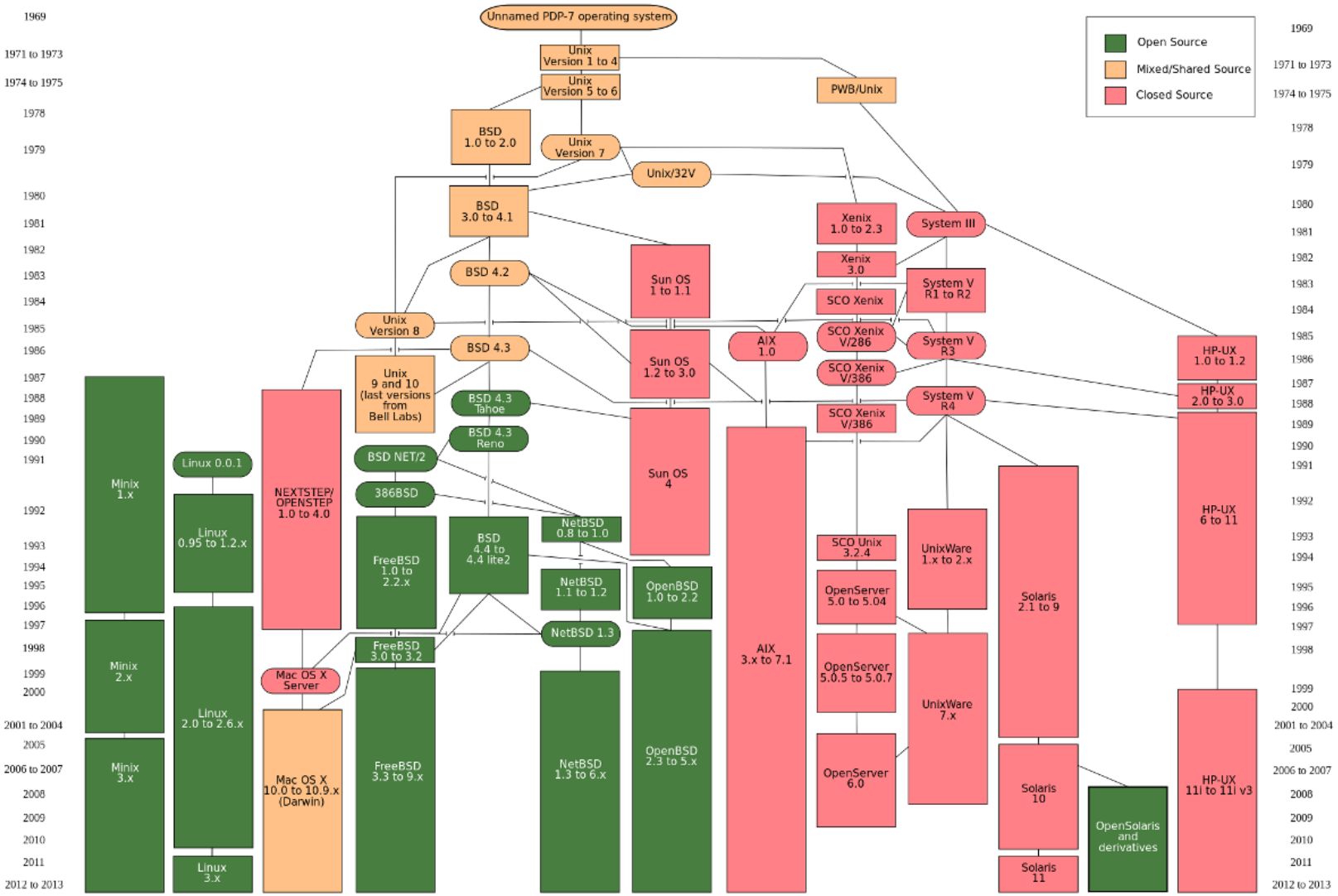
- GNU
  - "GNU's Not Unix!" – but they are compatible
  - Since 1984 Richard Stallman (founder of Free Software Foundation) tried to make new kernel (Hurd – not finished yet...)
  - Generally set of basic system tools – working with many kernels (Linux BSD*, Mac's Darwin, ...), also present in many commercial paid UNIX systems
  - Source code is free – anyone can study it (Security!), report bugs, contribute, modify, share it, ...

What it is UNIX, Linux and GNU... III

- GNU General Public License (GPL) – free spirit of open-source – license, idea, how to share software

- Linux
  - First version of kernel written by Linus Torvalds in Helsinki in 1991
  - Kernel was in principle inspired by various UNIX systems and using GNU tools for work
  - Quickly became popular – anyone can take it and use for any needs
  - Used in small embedded (commonly network) devices, mobile devices (book readers, Android, ...), personal computers, servers (from home level to biggest data centers), ...
  - Nowadays powering most of the Internet
  - Anyone can contribute – not only code, also documentation, design, translations, ...

Extremely simplified UNIX phylogeny



Cathedral vs. market place

What is principal difference between free open-source and commercial software

- Commercial software is like a cathedral
  - Pay big money and get it in the state which the architect like
  - User can not modify it (or it is terribly expensive)
  - Might be you don't need everything – but still paying whole set
- Free open-source software (FOSS) is like a market place
  - Find there many producers of same tools – pick up those you like – freedom of choice
  - Take exactly the tools you need – any combination is possible
  - Much cheaper to shop there
- Both have pros and cons – depends what you wish...

Free and open-source software I

- Free like freedom of speech, not like free beer!
- Not every OSS (generally less strict conditions) has to be **F**OSS (you can do with it (almost) whatever you like)– source code might be available under some circumstance (only to look), but modification and/or reuse of the code prohibited (and then it is not **free**)
- Open-source – source code can be seen by the holder of the license – many variants what he can do with the code then
- GNU GPL ("copyleft") – probably most common OSS license, strict, viral – derived code has to keep the license – surprisingly not fully "free" as it doesn't allow changes of license
- LGPL – Lesser GPL – more permissive
- BSD license – permissive – allow derived code to became closed-source (commonly used by Apple Mac OS X, Safari browser, ...)

Free and open-source software II

- Apache license, Mozilla license, ...– many variants... for specific use in particular software
- Creative Commons (CC) – software licenses above not suitable for multimedia, text, etc. – CC has many options (including denial of reuse of the product), see https://creativecommons.org/
- And many more...
- Orientation might be tricky, but practical output for users is more or less same – the software can be independently checked for bugs, backdoors, malware, can be improved and under some circumstances, new software can be derived, and usually, it is available for free
- Aim is to "liberate" software to keep open sharing of ideas, mutual improve and security control – although the point is clear, there are debates how to reach it...

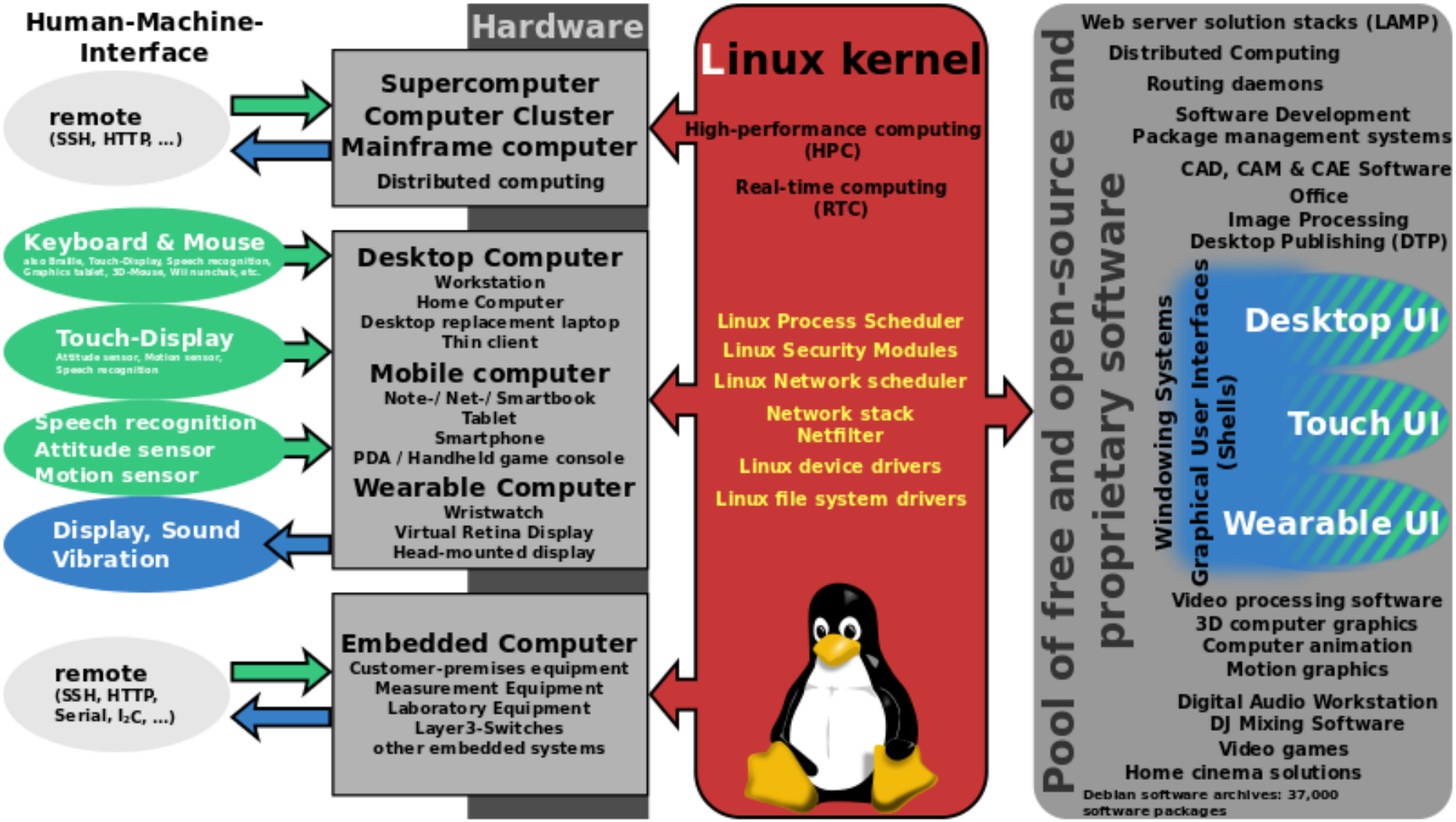
How to make money with free open-source software?

- Traditional model – user rents right ("buys a license") to use the software (and sometimes for support – usually for extra money)
- Common mistake – software is not "bought" – only license is rented
- Software as service
  - (F)OSS is available for free – user can use it as it is or buy a support – help
  - No vendor lock-in – user has the code, so he can modify the software himself, change solution, ...
  - Cheap for user as well as company – company specialized for one task, let's say server database, doesn't have to take care about the rest of the system – someone else does; user pays only what he needs
  - Our faculty is using Plone system for web pages – anyone can use it for free, someone (like we) found company to help, and if we'd decided, we could keep Plone and maintain it ourselves or find another company to help us with it

What it is a “Linux”

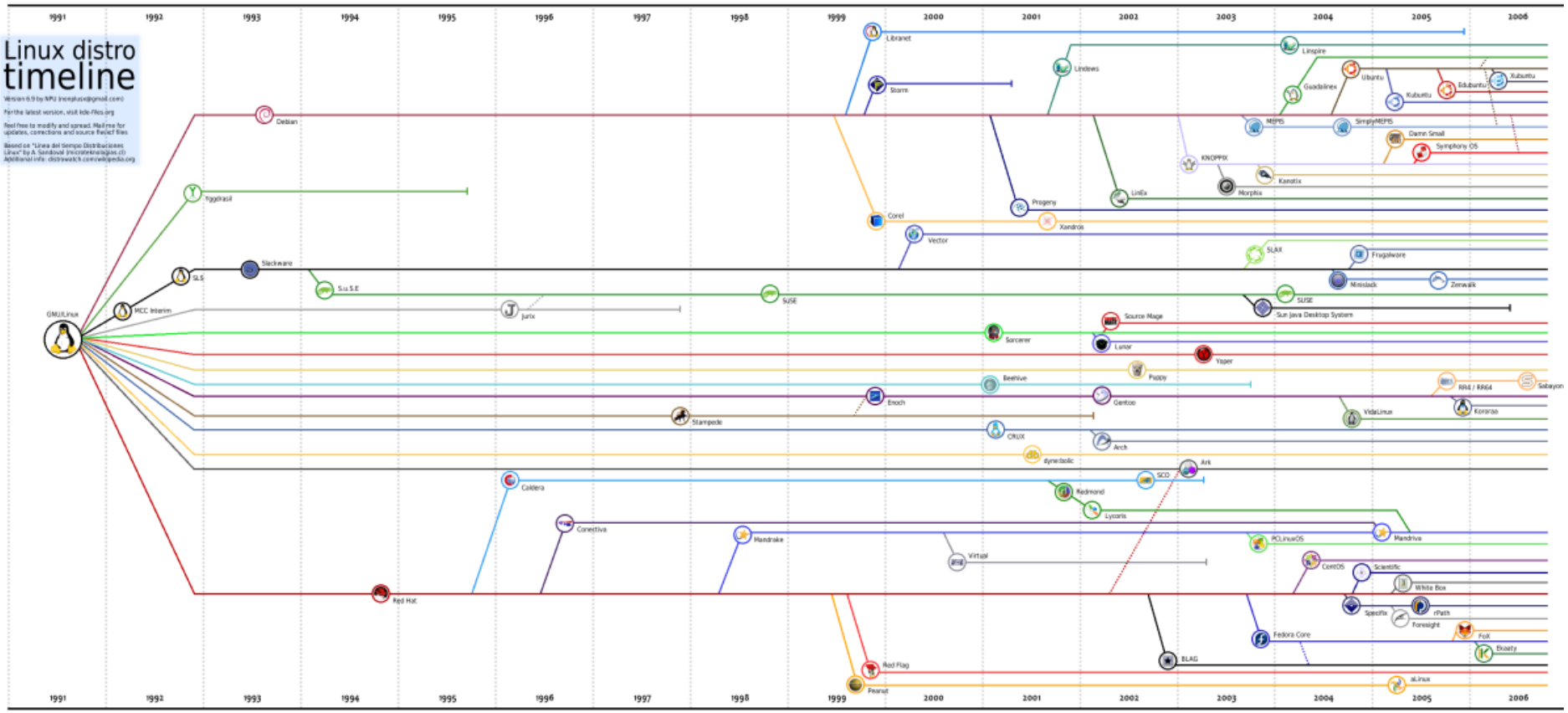
- Operating system respecting principles of UNIX
- Components
  - Linux kernel – basic part of the system responsible for hardware and very basic low-level running of the system
  - GNU core utilities – basic applications
  - Graphical user environment (GUI) – many choices
  - Many other applications – according to use – whatever imaginable
- Linux distribution?
  - Somehow assemble Linux kernel, basic tools and some applications
  - Optionally add some patches and extra tools and gadgets
  - Make your own design! (very important)
  - If lazy, remake existing distribution ;-)
  - Still surprised there are hundreds of them?
  - It is like Lego – pieces are more or less same across distributions, but result is very variable
  - From “general” for daily use (pick up whatever you like) to very specialized – special hardware devices, network services, rescue, ...

Linux kernel and other part around it



https://en.wikipedia.org/wiki/Linux_distribution

Extremely simplified adaptive radiation of Linux distributions



https://en.wikipedia.org/wiki/List_of_Linux_distributions
and https://distrowatch.com/ (currently ∼760 distributions)

Most common Linux distributions

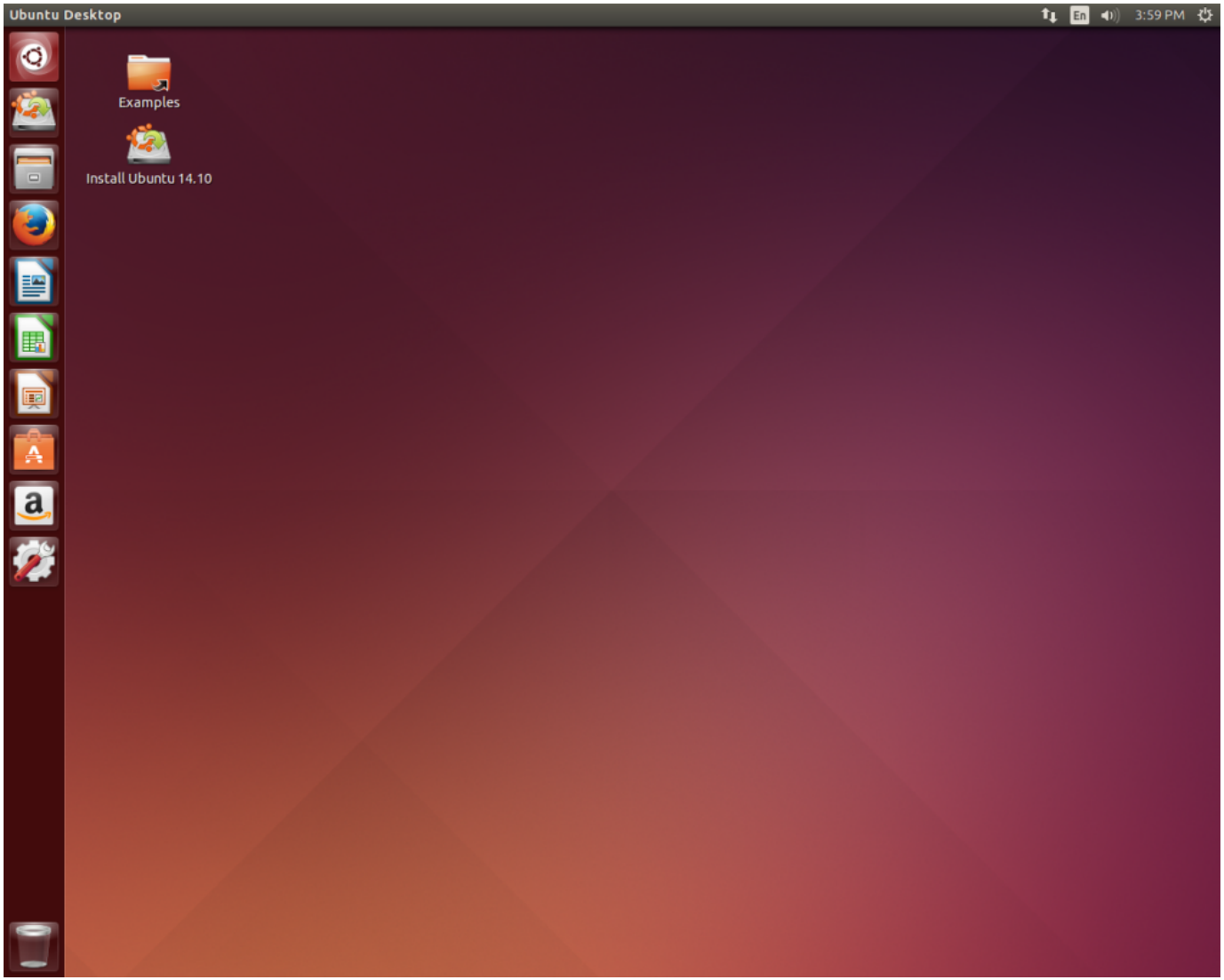
- Debian (DEB) based
  - Debian – one of oldest and most common, especially on servers
  - Ubuntu (nowadays probably the most popular on PCs and notebooks) and derivatives – Kubuntu, Xubuntu, Lubuntu, ... (according to GUI used)
  - Mint – Based on Ubuntu as well as Debian, very user-friendly
  - Kali, KNOPPIX, ...
- Red Hat (RPM) based
  - Red Hat – probably the most common commercial
  - Fedora – "playground" for Red Hat – very experimental
  - Centos – Clone of Red Hat
  - openSUSE – SUSE is second largest Linux company, openSUSE is community distribution (free)
  - Scientific Linux, Mageia, PCLinuxOS, ...
- Android
- For experienced users: Arch, Slackware, Gentoo, ...

Graphical User Interfaces

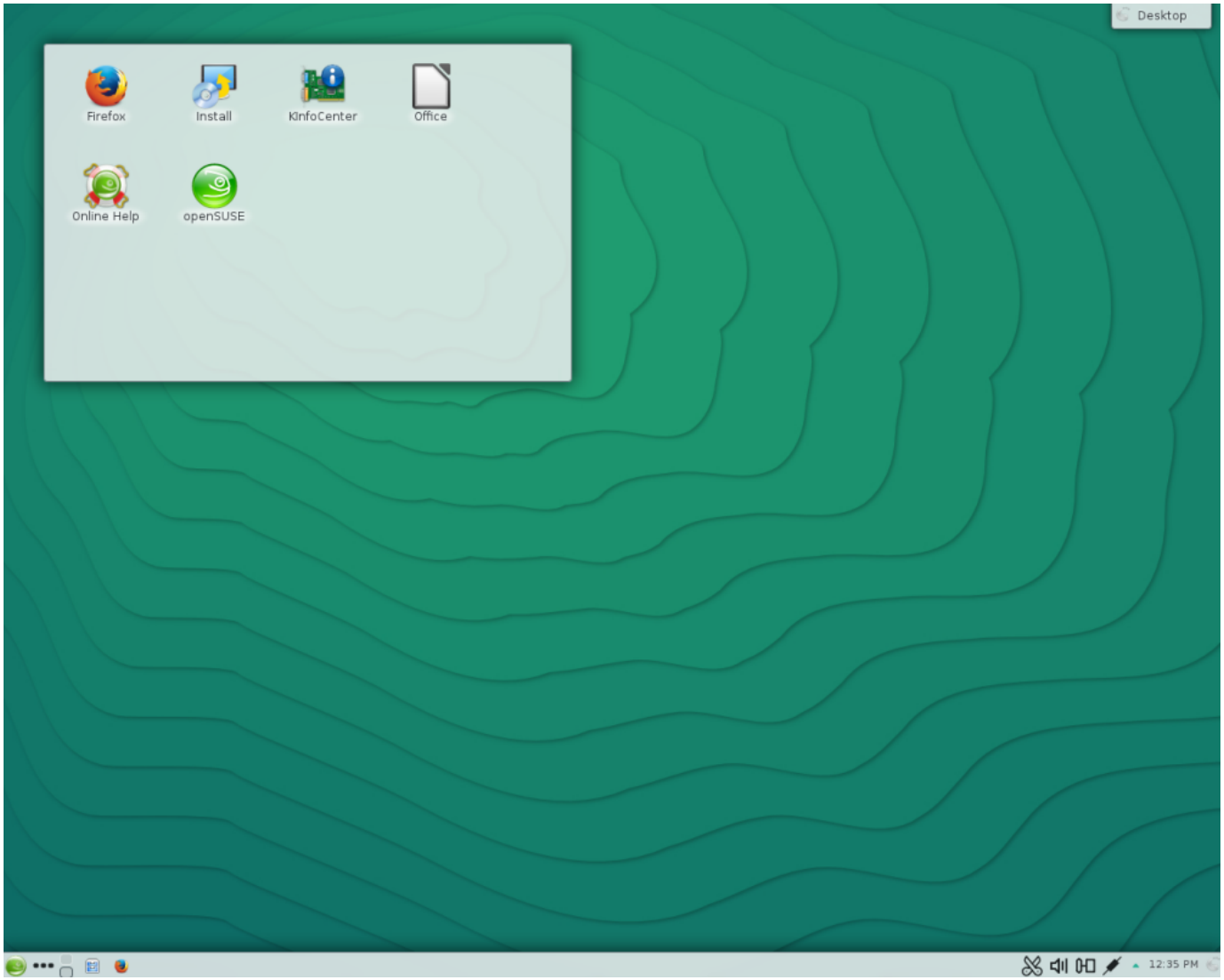
More like “Mac-style”, “Windows-style” or something else? Feature rich or minimalistic?

- Most of GUIs are available for most of common distributions – one is picked as default and “only” color style is different
- Unity – developed by Ubuntu, relatively specific (not common outside Ubuntu), “Mac-style”
- KDE – one of the most common, feature extremely rich, basically “Windows-like” (can be changed)
- GNOME – one of the most common, relatively simplistic interface, but still feature rich, “Mac-like”
- XFCE – lightweight version of older GNOME – for older computers or users not willing to be disturbed by graphical effects, basically “Mac-like” looking, but panels can be moved to “Windows style”
- Cinnamon – remake of GNOME to look more like Windows...
- And much more...
- Choose what you like – doesn’t matter much which one...

Ubuntu with Unity



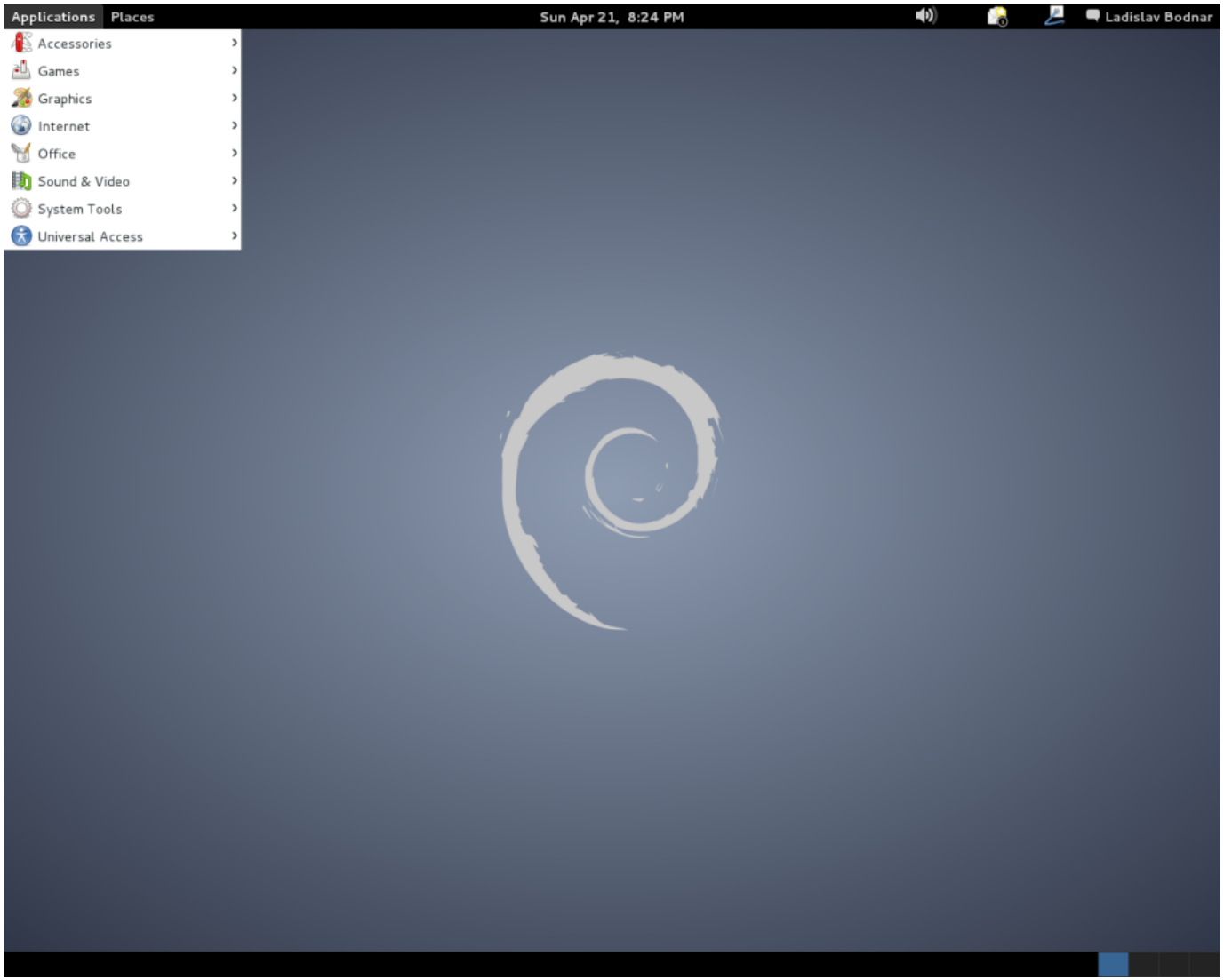
openSUSE with KDE – Kubuntu is same, but blue...



Fedora with GNOME

Linux Mint with Cinnamon

Debian with XFCE – Xubuntu has more “modern” design



How to try it?

- Install it on some computer together with or instead of Windows
  - If you can use whole disk, just boot from CD/USB and click "Next"...
  - If you don't have whole disk, you need at least one (commonly more) disk partition(s) – if you don't know how to manage them, ask someone skilled...
- Live CD/USB
  - The most easy – burn ISO image of CD from web of almost any Linux distribution or use for example UNetbootin to prepare bootable flash
  - You only have to know how to boot from CD/USB (usually press `ESC`, `DEL`, `F2`, `F10`, `F12`, ... when starting computer – varies according to manufacturer)
- Virtualization
  - Requires relatively powerful computer (preferable Intel i5 or i7 and over 4 GB of RAM)
  - Install virtual machine (probably the most easy is VirtualBox) – allows install and run another operating system inside host as an ordinary application – very easy and comfortable

The Linux diversity...

- Try several distributions and just choose one you like...
- Unless selecting among the most common, it doesn't matter much which one you pick up
- Which design do you like?
- Which distribution is your friend or colleague using?
- You can change GUI without change of distribution
- Applications are still same – no difference in Firefox across distributions – keep your settings when changing distribution
- Everyone using Android is using Linux ;-)
- Special use – FreeNAS for home as well as business file server, Parted Magic and/or SystemRescueCD to repair broken system (disk failure) and save data, ...

Differences among (common) Linux distributions

- Design and colors ;-)
- Default GUI
- Applications available right after installation
- Default settings (not much)
- Package management – especially in command line
- Management of system services (how to start/stop certain server service like database or web) – not important for daily usage for most of users
- Sometimes in location of some configuration files (for system services) – also not important in daily usage of most of users
- Kernel is almost same, applications are used in same way
- Command line is almost same across Linux, and almost same as in other UNIX systems

Short overview of hard disk layout

- Physical disk (piece of hardware) has at least 1 partition – division seen in Windows as "disks" (`C`, `D`, ...) and mounted directory in UNIX
- MBR – older description of disk division, up to 4 primary partitions (OS typically requires at leas one to run), one can be extended and contain more partitions, disks up to 2 TB
- GPT – newer, no relevant limits, requires UEFI (replacement of BIOS, newer computers)
- If unsure what to do, high probability to break it...
- Blank new partition has to be formatted to desired file system according to use and target operating system
- Linux distributions have easy graphical tools to manage disk partitions
- Always have backup before such management!

Put together more disks

- RAID – Redundant Array of Inexpensive/Independent Disks
- RAID 0 – stripping, no redundancy, no security, speed up (two or more disks joined into one, files divided among disks)
- RAID 1 – mirroring – even number of disks of same size – resulting capacity is half, very fast, secure
- RAID 5 – at least three disks, one is used for parity control, little bit slower
- Combinations (RAID 10, ...)
- LVM – Logical Volume Management – built over several partitions/disks – seen by OS as one continuous space, can be dynamically managed
- Functionality of RAID and LVM (and more) is more or less covered by XFS and Btrfs (next slide)

File systems

FS name	Name length	Characters in file name	Path length	File size	Partition size	Systems
FAT32	255	Unicode	No limit	4 GiB	2 TiB	Any
exFAT	255	?	No limit	16 EiB	64 ZiB	Any
NTFS	255	Variable	Variable	16 TiB	16 EiB	Windows, read-write in UN*X
HFS+	255	Unicode	?	8 EiB	8 EiB	Mac OS
ext4	255	Any, not /	No limit	16 TiB	1 EiB	UN*X
XFS	255	Any	No limit	9 EiB	9 EiB	UN*X
Btrfs	255	Any	?	16 EiB	16 EiB	UN*X

- FAT32 (including extensions) is old-fashioned and not reliable FS
- NTFS, FAT do not support UNIX permissions, so they can't be used as system partition in Linux
- ext4, XFS and Btrfs are not accessible in Windows
- XFS and Btrfs are the most advanced FS in common use

File names

- Linux allow any character in file name, except slash (`/`), so including anything on keyboard as well as line break (!). Be conservative...

```
mkdir My New Directory # Produces THREE directories (mkdir creates
                       # directories; spaces separate parameters)
                       # Solutions:
mkdir "My New Directory" # (you can use simple quotes '...' as well) or
mkdir My\ New\ Directory # "\" escapes following character
rmdir My\ New\ Directory # Same problem and solution when removing it
touch \* # Creates new empty file named just *
rm * # What would be removed? :-)
rm \* # This works...
```

- Files and directories starting by dot (`.`) are hidden by default (typically user settings and application data in user home)

```
touch .hiddenfile # Let's make empty text file hidden by default
ls # We will not see it
ls -a # We will see it
```

Directory structure in Linux I

- Similar in another UN*X systems
- Top directory "`/`" – "root"
- Everything else (including disks and network shares) are mounted in subdirectories (/...)
- `/bin` – very basic command line utilities
- `/boot` – bootloader responsible for start of system
- `/dev` – devices – disks, CD, RAM, USB devices, ...
- `/etc` – system configuration in plain text files – edit them to change settings (read documentation and comments there)
- `/home` – users' homes
- `/lib`, `/lib64` – basic system libraries
- `/lost+found` – feature of FS, after crash and recovery of FS, restored files are there

Directory structure in Linux II

- `/media` – attached disks (USB flash, ...) usually appear there (might be in `/var/run/media`) – subdirectories are automatically created when device is plugged and disappears when unplugged
- `/mnt` – usually manually mounted file systems (but can it can be mounted elsewhere)
- `/opt` – optional, usually locally compiled software
- `/proc` – dynamic information about system processes
- `/root` – root's (admin's) home
- `/sbin` – basic system utilities
- `/selinux` – SELinux is security framework
- `/srv` – FTP and WWW server data (can be in `/var/srv`)
- `/sys` – basic system

Directory structure in Linux III

- `/tmp` – temporary files – users have private dynamically created spaces there
- `/usr` – binaries (executable applications) and libraries of installed applications
- `/var` – data of most of applications and services, including e.g. database data, system logs, ...
- `/windows` – if on dual boot, Windows disks are commonly mounted here
- Can be altered, modified
- Usually, work only in your home, anywhere else modify files only if you are absolutely sure what you are doing
- Normal user doesn't have permission to modify files outside his directory (with exception of plugged removable media)
- Try `man hier` for details

Configuration in /etc (examples)

- Configuration of system services (servers, ...) and behavior
  - Apache web server, database, FTP server, networking, basic system settings, ...
- cron* – cron automatically repeatedly runs tasks
- fstab – description of mounted FS
- group – list of users and groups
- passwd – basic settings of for users (home directory, default shell, ...)
- resolv.conf – DNS settings (part of basic networking)
- shadow – users passwords in encrypted format
- skel – basic directories and configuration for new users
- Much more...

Types of files

- `ls -la`
- Regular file – ordinary file, marked by `-`
- Directory – in UNIX special type of file, marked by `d`
- Symbolic link (symlink, “soft link”) – points to another place, marked by `l`, slide 63
- Hard link – just another name for existing file, no special symbol, slide 63
- Block and character device – in `/dev`, representations of devices (hard disks, terminals, …), marked by `b` or `c` respectively
- Named pipe – pipe can be saved (by `mkfifo`), looks like a file, more at slide 71
- Socket – for communication among processes, also bidirectional, available on network

Login to remote server

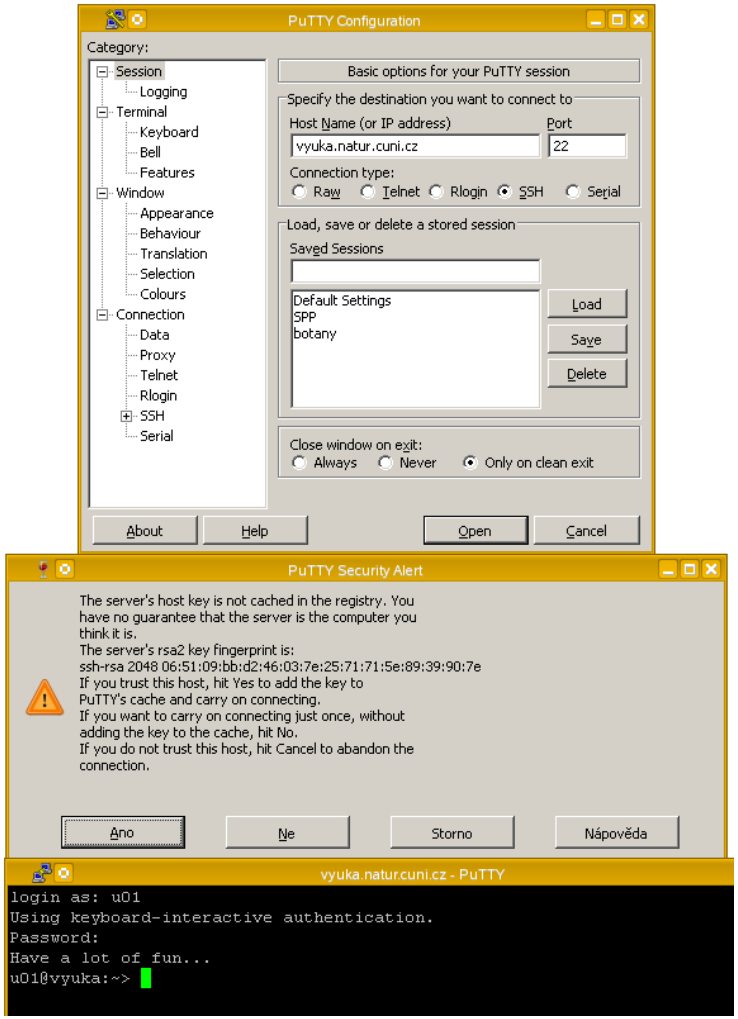
SSH – secure shell – encrypted connection

```
ssh remoteUser@remote.server.cz
# When logging first time, check
# and confirm fingerprint key
yes # And press Enter
# Type remote user's password
# (nothing is shown when typing)
# Confirm by Enter
```

Our toy server: user names from `u01` to `u30`, password: `Cuser1`

```
ssh uXY@vyuka.natur.cuni.cz
```

If fingerprint key changes, ssh complains a lot – possible man in the middle attack



File permissions

- Combination of permissions to read/write/execute for user(owner)/group/others

Permission	Number	Directory	File
r	4	Read content	Read content
w	2	Write into it	Write into it
x	1	Enter it	Launch application

- `rwxr-wr--` – 3*3 characters for permissions for owner of the file/directory, group it is belonging to, and other users (`d` on beginning marks directories, `l` links, `+` ACL, slide 42)
- `764` – same as above — numbers for each role are summed – first one is for owner, second for group and last for others

Permission examples

```
ls -l
# Only owner can read and write the file; 600
-rw-------   1 vojta users   38211 20. led 09.23 .bash_history
# Owner can write read and write the file, others read; 644
-rw-r--r--   1 vojta users    2707 29. lis 16.21 .bashrc
# Owner can enter, read and write directory, others can read
# and enter it; 755
drwxr-xr-x  41 vojta users    4096 27. pro 09.55 bin
# Only owner can read, write and enter the directory,
# others nothing; 700
drwx------  58 vojta users    4096 17. pro 15.45 .config
# Link, everyone can do everything; 777
lrwxrwxrwx   1 vojta users      37 20. led 09.33 .lyxpipe.in ->
  /tmp/kde-vojta/kilemj7d3E/.lyxpipe.in
# Executable (application) - everyone can launch it, but only
# owner can write into the file; 755
-rwxr-xr-x   1 vojta users    2187 27. lis 13.10 strap.sh*
```

Permission to “write” also means permission to **delete** it.

Check and modify permissions

```
ls -l # Long list - file names and attributes
ls -a # All, including hidden files (starting with dot)
ls -F # Add on the end of name "/" for directories and "*"
      # for executable
ls -h # Human readable size units (use with -l or -s)
ls --color ## Coloured output
ls -laFh --color # Combine any parameters you like
chmod u/g/o/a+/-r/w/x FILE # For respective user/group/others/all
                           # adds/removes permission to
                           # read/write/execute
chmod XYZ FILE # Instead of XYZ use number code of permission
chmod -R # Recursive (including subdirectories)
chmod +x script.sh # Make script.sh executable for everyone
chmod o-r mydir # Remove read permission from others on mydir
chmod 600 FILE1 FILE2 # Make both files readable and
                      # writable only by their owner
chmod 000 FILE # No one can do anything - owner or root must add
               # some permissions before any manipulation...
chmod 777 * # All permissions for everyone on everything
```

Extending permissions – ACL

Access control list

- By default, it is not possible to give specific permission to the user who is not owner, nor member of group owing the file
- In ext4 FS it has to be turned on manually (usually it is by default), it is part of XFS and Btrfs
- Command `getfacl` lists those extra permissions
- When in use, "basic" tools listing permissions (e.g. `ls -l`, ACL in use is marked by `+` on the beginning of the line) sometimes do not show correct result
- Command `setfacl` sets it

```
1 getfacl FILE # get ACL for FILE
2 setfacl -m u/g:USER/GROUP:r/w/x FILE # Add for USER/GROUP r/w/x right
3 setfacl -R ... # Recursive (including subdirectories)
4 setfacl -b FILE # Remove all ACL from FILE
```

Set default permissions for new files

- `umask` sets implicit permissions for newly created files for user
- syntax is similar to `chmod`, but reverse (e.g. `027` keeps all rights for owner, for group only reading and nothing for others)
  - `umask` number **removes** certain permissions
- `umask 027` (or other number) is typically set in `~/.bashrc`
- Typically used in network environment
- Set with care – new permissions will have plenty of consequences (different are typically needed for web pages, private files, shared files etc.)
- `umask` work recursively for all new files in user home directory – it is not possible to set new implicit rules for particular directory

Other permissions

- sticky bit – new directory/file in shared directory (where everyone can write) will be deletable only by owner (typically in `/tmp`)

```
chmod +t somedirectory
ls -la /
drwxrwxrwt 22 root root 800 21. led 18.20 tmp # "t" marks it
```

- setgid – application can have root permission even it was launched by normal user

```
chmod u+s someapplication
ls -al /bin/passwd
-rwsr-xr-x 1 root shadow 51200 25. zář 08.38 /usr/bin/passwd # Note "s"
```

- `chattr` – change of advanced attributes on Linux FS
- Usually, there is no need to modify them

```
chattr -RVf -+=aAcCdDeijsStTu files
man chattr # See explanation of attributes
lsattr # List extended attributes
```

Owner and group

- Every file has a owner and group – for finer setting of rights
- Group can have just one member – the user
- System usually shows names of groups and users, but important are IDs: GID and UID
- Commands chown requires root privileges
- Commands chgrp commonly requires root privileges – user has to be member of particular group to be able to change ownership to it
- Information about users and groups and their IDs are in `/etc/group` and `/etc/passwd`

```
1 ls -l # Shows also owner and group
2 id # Display UID and GIDs of current user
3 chown newowner:newgroup files # Change owner and group
4 chown -R newowner files # Recursively (-R) change owner
5 chgrp -R newgroup files # Recursively (-R) change group
```

Root vs. "normal" user

- Root is administrator – more than God – can do anything
- Other users have limited permissions
  - System users providing particular service (web server, database, networking service) are as restricted as possible to do the task – security
  - "Human" users don't have access to system files (at least not for modification), homes of users are separated

Gain root privileges:

```
su # Requires root password (stay in current directory)
su - # Requires root password (go to /root)
su -c "some command" # Launch one command with root permissions
su USER # Became USER (his password is required)
sudo -i # For trusted users, became root (asks for user's password)
        # User has to be listed in /etc/sudoers
        # Can be restricted for particular commands
sudo somecommand # Launch somecommand with root's privileges
```

Text and text – differences among operating systems

- Windows and UNIX have different internal symbol for end of line (new line) – EOL
  - UNIX: LF (`\n`)
  - Windows/DOS: CR+LF (`\r\n`)
  - Older Mac: CR (`\r`) (Mac up to 9 wasn't UN*X, since OS X is)
- Good text editor can open correctly any EOL, but for example execution of script written in Windows will probably fail on Linux
- Different systems use different encoding
  - UNIX: mainly UTF-8 (unicode, universal)
  - Windows: win-cp-125X (variants according to region)
  - Older UNIX: ISO-8859-X (variants according to region)
  - Other much less common types
- Text editors can usually open any encoding, but auto detection commonly fails – set it manually

Converting the text

Prevent bad display and weird errors when launching scripts

Mac OS X mostly uses same encoding and EOL as Linux (and rest of UNIX world), so there are no problems with compatibility

```
unix2dos textfile # Convert text file from UNIX to Windows EOL
unix2mac textfile # Convert text file from UNIX to old Mac EOL
dos2unix textfile # Convert text file from Windows to UNIX EOL
mac2unix textfile # Convert text file from old Mac to UNIX EOL
unix2dos --help # More information about usage, include encodings
iconv -f ISO-8859-2 -t UTF-8 infile.txt > outfile.txt
  # Converts encoding of input file (ISO-8859-2) to outfile in UTF-8
iconv -l # List of available encodings to convert
iconv --help # More information about usage
recode CP1250..UTF-8 textfile # Convert encoding from CP-1250 to UTF-8
recode ../CR-LF textfile # Convert EOL from UNIX to Windows
recode -l # List of available encodings to convert
recode --help # More information about usage
```

Launching of bash script written on Windows on Linux will probably fail (because of different EOL)

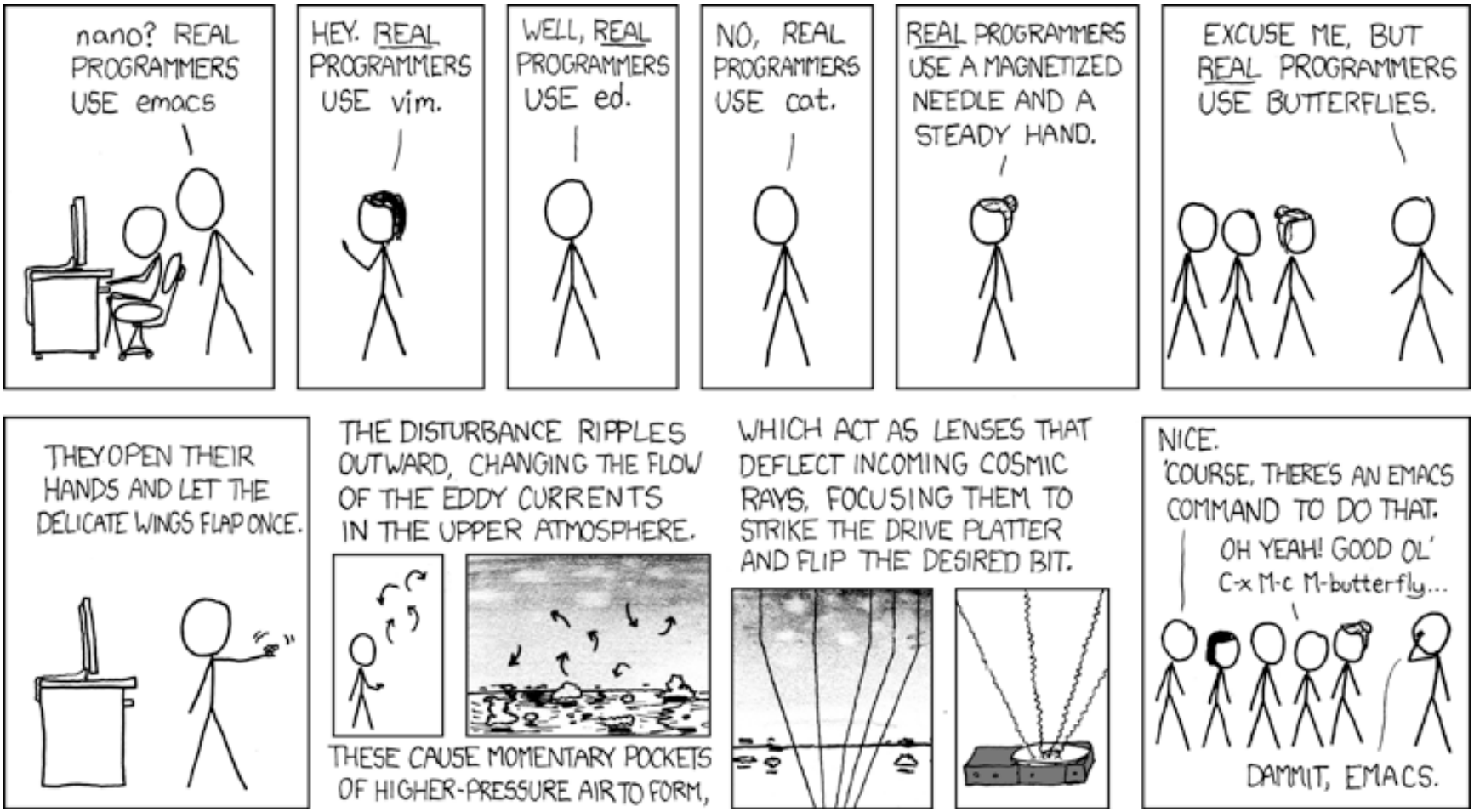
Importance of good text editor

Can your text editor...?

- Show syntax highlight
- Show line numbers
- Show space between brackets
- Open any encoding and EOL
- Fold source code
- Show line breaks
- Mark lines
- Open multiple files
- Advanced search and replace
- Use regular expressions
- Make projects
- Add notes
- Use command line
- Debug source code

- Kate
- KWrite
- Vim
- GNU Emacs
- Geany
- Bluefish
- Gedit
- Notepad++
- Sublime
- Atom
- Nano
- And more...

It **is** important to select **good** text editor...



https://xkcd.com/378/

Friendly interactive shell

- Many names, many ways how to get it, still the same thing
- Fish – the command line interface
- Terminal (console)
  - Originally machine used for connection to remote server
  - System uses old fashioned terminal for inner purposes
    - From GUI available using `Ctrl+Alt+F1` to `F12`
    - Changing terminals using `Alt+F1` to `F12`
    - Return back to GUI using `Alt+F7`
    - Some are used for log outputs etc.
  - Nowadays used "indirectly" with special applications ("emulators")
- Terminal emulator
  - Application used to get the "terminal" and work in command line
  - Every GUI has some – Konsole, Yakuake, XTerm, Gnome Terminal, Guake, LxTerminal, ...
  - Commonly allow appearance customization – font, colors, background, style of notifications, ...
  - Launch as many copies as you need (some allow tabs for easier work)

BASH and others

- Shell (sh) – feature rich scripting programming language – general specification, several variants
- So called POSIX shell – Portable Operating System Interface – transferable among hardware platforms (and UNIX variants)
- **Interpreter of our commands inserted into command line**
- **BASH** – Bourne again shell
  - Probably the most common shell, based on original sh, respecting original specification, adding new features
  - We will use it
- Other variants: **csh** (syntax influenced by C), **ksh** (younger, backward compatible with bash), **zsh** (extended bash), **ash** (mainly in BSD)
- There are subtle differences in syntax and features
- Language suitable for easy scripting and system tasks, not for "big" programming, neither for graphical applications

Nice BASH features for easier work (selection)

- Arrows up and down list in the history of commands
- List whole history by command `history`
- `Ctrl+R` – reverse search in history – type to search last command containing typed characters
- `TAB` – list command and files starting by typed characters
- `Home`/`End` – go to beginning/end of the line
- `Ctrl+L` – clear screen (like `clear` command)
- `Ctrl+Shift+C`/`V` – copy/paste the text
- `Ctrl+C` – cancel running task
- `Ctrl+D` – log out (like commands `exit` or `logout`)
- `Ctrl+U` – move text before cursor into clipboard
- `Ctrl+K` – move text after cursor into clipboard
- `Ctrl+left/right arrow` – skip words
- `Ctrl+T` – flip current and left character
- `Ctrl+X+E` – start text editor in current directory

Variables

- Variables contain various information (where to look for the executable programs, name of the computer, user settings, ...)
- Can be local (within a script for some temporal purpose) or global – available for all processes
- Commonly written in CAPITALS (just a costume)
- Popular and useful variables
  - `HOME` – location of user's home directory
  - `HOSTNAME` – network name of the computer
  - `LANG` – language settings, encoding, similarly variables `LC_*`
  - `PATH` – paths where to look for applications – all applications have to be in `PATH` or called directly
  - `SHELL` – shell in use (bash or something else)
  - `USER` – user name
  - And many more, commonly specific for particular server

Work with variables

```
printenv # Get all exported variables and their values
export -p # Get all exported variables and their values
echo $VARIABLENAME # Get value of particular variable
echo $PATH # Get path where to look for applications
VARIABLE='variablevalue' # Set new variable and its value
                         # Or replace existing variable by new value
export EDITOR=/usr/bin/vim # Set new default text editor
export PATH=$PATH:~/bin # Extend PATH -- add /home/$USER/bin
                        # Take existing PATH and add new values
                        # separated by ":"
export GREP_OPTIONS='--color=auto' # Coloured grep
unset VARIABLENAME # Drop variable and its value
```

- Exported variables will be lost when logging off
- To make variables permanent, add `export` commands into `~/.profile` or `~/.bash_profile`, or `~/.bashrc` (according to shell and its settings)
- "`~`" means home directory

The PATH variable

- Lists directories (separated by colon `:`) where the current shell searches for commands
- If some software is installed outside standard locations, the user must specify the full path (or update the `$PATH`)
- In case there are two commands with the same name (e.g. `/bin/somecommand` and `/usr/bin/somecommand`), the order of directories in `$PATH` matters – the first occurrence is used, any possible later ignored

```
1 # See the $PATH variable
2 echo $PATH # Sample output is on the next line:
3 /home/$USER/bin:/usr/local/bin:/usr/bin:/bin:/opt/bin:/sbin:/usr/sbin
4 # Adding new directory to $PATH
5 export PATH=$PATH:/some/new/directory # Ensure to add original $PATH
6 # Do not do it in the following way - it would overwrite $PATH, and
7 #   there would be only the new directory (not the original content)!
8 export PATH=/some/new/directory # Wrong! Old $PATH is missing!
```

Reading variables from command line

```
read X # We will read new variable from input (no need to use "$" here)
10 # Type any value and press Enter
echo $X # Get value of the variable
10 # It works
unset X # Destroy this variable
# Following two commands are very similar and can lead to same result
X=`command` # Set as variable output of command
X=$(cat somefile) # Read into variable from file
echo $X # X will contain content of somefile
```

- This is especially useful in scripting to read input from users or from another commands

Expressions

```
# Many operands have special meaning in BASH - must be escaped
echo `expr 1 '<' 2` # Is 1 smaller than 2? TRUE
echo `expr 1 '>' 2` # Is 2 smaller than 1? FALSE
echo `expr 5 '%' 2` # What remains after aritmetic division
echo `expr 1 '&' 0` # If both arguments non-empty and not 0, then 1
x=`expr 1 '+' 6` # Result will be in $x
echo $x
x=1 # Set x to 1
y=$x+1 # Will this add 1? Why?
echo $y # See result
y=`expr $x + 1` # This will work - note ` and space around +
echo $y # Result
echo `expr length "MetaCentrum and Linux"` # Get length of chain
# String of 10 characters starting at position 12 of the text
echo `expr substr "MetaCentrum and Linux" 12 10`
# Does 1st chain contain 2nd chain? Get position of first hit
echo `expr index "GNU Linux" "Linux"` # If no overlap, return value is 0
```

- `expr` works with various operands (see `man expr`)

Aliases and BASH settings – became lazy

Alias is short cut – instead of very long command write short alias

```
# Define new alias
alias ll="ls -l"
# Since now, instead of "ls -l" we can use just "ll"
# To make the change above permanent, write it into ~/.profile or
# ~/.bash_profile or ~/.bashrc and the launch
source ~/.bashrc # to reload BASH settings
# or "source" the file you modified
# Popular aliases
alias ls="ls --color=auto" # Make output of ls colored
alias l="ls -la" # Long list (add details) with hidden files
# Popular settings in ~/.bashrc (influencing bash, not other shells)
test -s ~/.alias && . ~/.alias || true # Check for extra alias file
eval "`dircolors -b`" # More colors for outputs
HISTCONTROL='ignoreboth' # Ignore repeated entries in bash history
HISTFILESIZE='100000' # Length of bash history
```

BASH globbing and wildcards

- BASH itself doesn't recognize regular expressions – it's wildcards have some of functions of regular expressions and can look similarly, but behave differently!
- `?` – Any single character
- `*` – Any number of characters
- `[]` – Range or a list – `[abcdef]` and `[a-f]` are same
- `[!...]` – Reverse previous case (`!`) – any character except those listed
- `{}` – Expansion (terms inside are separated by commas `,`) – all possible combinations (see next slide for examples)
- `\` – Escapes following character and it doesn't have its special meaning (e.g. `\*` means asterisk `*` and not "any number of characters")
- For details see `man 7 glob` and `man 7 regex`

Brace expansion and quotes

```
1 echo a{p,c,d,b}e # ape ace ade abe - all combinations
2 echo {a,b,c}{d,e,f} # ad ae af bd be bf cd ce cf - all combinations
3 ls *.{jpg,jpeg,png} # expansion to *.jpg *.jpeg *.png, same as
4 ls *.jpg *.jpeg *.png
```

- Text in single quotes (`'...'`) preserves the literal value of each character within the quotes
- Tex tin double quotes (`"..."`) preserves the literal value of all characters within the quotes, with the exception of dollar (`$`), back tick (`` ` ``) and back slash (`\`)
- A double quote may be quoted within double quotes by preceding it with a backslash
- Text between back ticks (`` `...` ``) will be evaluated and then used as command or argument

Exemplar quotes and more

```
1  a=abcdef # Set new variable
2  echo $A # See variable's content
3  abcdef
4  echo '$A' # Single quotes preserve literal value
5  $A
6  echo "$A" # Double quotes preserve literal value, except $, `, \
7  abcdef
8  echo `$A` # Text between back ticks is evaluated and launched
9  abcdef: command not found # There is no command "abcdef"...
10 WORKDIR=`pwd` && echo $WORKDIR # Common use of `...` and $
11 echo "Hi, dear $USER" # Compare this and following command...
12 echo 'Hi, dear $USER' # Single quotes do not evaluate variables
```

- `$` marks variables
- `\` escapes following character – it will not have its special meaning (space to separate arguments, …)
- `*` replaces any characters (`ls a*` lists all files starting with “a”)
- `?` replaces one single character

Links

- Soft links – like links on the web – short-cut to another place: `ln -s source target`
  - When we delete link, nothing happens, when target, non-working link remains

```
ls -l bin/cinema5
lrwxrwxrwx 1 vojta users 42 5. dub 2014 cinema5 -> # "l" marks link
  /home/vojta/bin/cinema5-0.2.1-beta/cinema5* # "->" points to target
```

- Hard links – only second name for file already presented on the disk (available only for files): `ln source target`

```
ln .bashrc .bashrcX
ls -l .bash* # Numbers in first column show links pointing to it
             # For directories - number of items, for files = 1
-rw------- 1 vojta users 27298 21. led 16.43 .bash_history # One link
-rw-r--r-- 2 vojta users  2707 29. lis 16.21 .bashrc # Same file as below
-rw-r--r-- 2 vojta users  2707 29. lis 16.21 .bashrcX # Two links
```

Places to store BASH settings

- `~/.bashrc` – File is loaded each time user creates new session (typically opens new terminal window)
- `~/.bash_profile` – Used specifically (not in every system) when user is using remote connection (SSH)
- `/etc/bash.bashrc` – System wide BASH settings – can be overridden by user's configuration
- `~/.profile` – Settings loaded when user logs-in (mainly for language settings), sometimes used by remote connections
- `/etc/profile` – System wide profile file – can be overridden by user's configuration
- Note: BASH scripts are non-interactive shells – they do not read settings above – there are no aliases, ... but they inherit some settings (PATH, language, ...)

Launching commands and scripts

- Parameters of commands are separated by space and preceded by one or two minus(es)
- Parameter `-h` or `--help` usually gives help for particular command
- Getting help with `man` command
  - `man somecommand`
  - Arrows to list up and down, `q` to quit
  - Type `/` and type text and hit Enter to search – next hit by `n`, quit search by ESC (twice)
  - Command `info` more advanced – type `?` for help
- Parameters can be combined, order doesn't matter (same variants: `ls -la`; `ls -al`; `ls -a -l`; `ls -l -a`)
- "Long" parameters (`-XXX`) must stay separated
- Commands must be in PATH – actual directory isn't in PATH
  - If the script is is current directory, use `./script.sh` or full path
- Custom scripts must have execute permission (`chmod +x script.sh`)

Screen

Split terminal or keep task running after logging off

- When you log off or network connection is broken, running tasks for particular terminal usually crash
- Sometimes number of connections is limited
- `screen` is solution – virtual terminals
- Launch `screen` to start new screen terminal, read some info, confirm by **Space key** or **Enter**
- To detach from the screen press `Ctrl+A, D` – screen is still running in background – you can even log off
- To return back to running screen use `screen -r` – if only one screen is running, you get back to it
- If more screens are running, use `screen -r 1234` (the number is seen from `screen -r`)
- To cancel running screen press `Ctrl+D` (or type `exit` or `logout`)

Automated launching of tasks

- `at` can run command at certain time (atd daemon must be running)

```
systemctl status/start/stop atd.service # Check/start/stop if atd runs
at HHMM # Run commands at certain time check "man at" for time settings
at> command1 # Add as many commands as you wish (separate by Enter)
# When done, press Ctrl+D to cancel giving commands to at
at HHMM <<< 'cat somescript.sh' # Run script at certain time
```

- `cron` runs tasks repeatedly (cron daemon must be running)

```
systemctl status/start/stop cron.service # Check/start/stop if cron runs
crontab -l # List cron tasks
crontab -e # Edit cron tasks:
# Minute, Hour, Day in month, Month, Day in week, Command (absolute path)
# 0-59    0-23  1-31           1-12   0-6 starting with Sunday (WTF?)
  *       *     *              *      *            /usr/bin/command
  10      22    1              *      *       # 1st day in month, 23:11
  0       */3   *              *      *       # Every 3 hours
# root can copy scripts into directories /etc/cron.*/
```

Chaining commands

- `&` – command will be launched in background, terminal is available for next typing: `firefox &` (when launching graphical application, hit **Enter** afterward if there is no active command line prompt)
- `&&` – second command is launched only when first command exits without error (exits with status `0`): `mkdir NewDir && cd NewDir`
- `;` – second command is launched regardless exit status of the first one: `kshfskcbd; hostname`
- `{...}` – commands within curl brackets are launched as one block
- `||` – second command is launched when first command fails (has non zero exit status):
  `cd newdir || { mkdir newdir && cd newdir; }`
- Behavior in shells other than bash might be little bit different
- `|` – pipe – redirects standard output of one command into standard input of second command: `ps aux | sort`

Standard input and output and redirects

- Standard input (`stdin`) is standard place where software takes input (keyboard and terminal) and writes results to standard output (`stdout`) – typically monitor
- Standard error output (`stderr`) is target of error messages – typically also monitor (but can be log file or so)
- `>` redirects output into new place (file, device, another command, ...)

```
1 cat /etc/group # Print whole file /etc/group
2 grep users /etc/group > users # Extract from /etc/group lines containing
3                               # "users" and write output into new file
4 cat users # See result
```

- `>>` adds output to the end of the file (`>` rewrites target file)

```
1 grep root /etc/group >> users # Add new information into existing file
2 cat users # See result
```

Redirects of standard input and output

```
# Write directory content into text file
# If file directory_listing.txt exists, will be overwritten
ls -la > directory_listing.txt
# If file directory_listing.txt exists, new content will be added to
# the end
ls -la >> directory_listing.txt
```

Redirects and pipes

```
# Add error output to the end of standard output file
command >> outputfile 2>&1
# Add error output to the error log text file
command >> outputfile 2>error.log
```

- `/dev/null` – "black hole" – discards everything (don't care about errors?): `command 2> /dev/null`
- `/dev/stdin` – standard input (in case application reads files, not from standard input): `echo "Žluťoučký kůň" | iconv -f utf-8 -t cp1250 /dev/stdin`
- `/dev/stdout` – standard output (we wish to see errors which would be discarded otherwise): `command 2> /dev/stdout`
- `/dev/stderr` – standard error output (right place to send errors to): `echo "error" > /dev/stderr`

Which system are we using?

```
uname -a # Information about Linux kernel (version, ...)
lsb_release -a # Information about Linux distribution release
cat /etc/os-release # Similar to above command
lscpu # Information about CPU
cat /proc/cpuinfo # Raw list of information about CPU
lsusb # List of devices on USB
lspci # List of PCI devices (graphic card, network card, ...)
lshw # Complete list of hardware
lshw -C memory # Information about RAM
hwinfo # Complete list of hardware
hwinfo --network # Information about network devices
free -h # Available memory (RAM) and swap, -h for nice units
df -h # Free space on disk partitions, -h for nice units
lsmod # List loaded kernel modules
uptime # How long is the system running, number of users, average load
date # Date and time - plenty of options for formatting
mount | column -t # Information about mounted file systems
findmnt # Display mounted devices in tree structure
```

Processes – every running program has its own process

```
htop # Nice listing of processes (better version of top), quit using "q"
pstree # See running processes with child processes, recursively
pgrep application # Return PID (process ID) of application
ps # processes related to actual terminal
ps x # All user's processes
ps aux # All processes
# kill (terminate) process by name or process ID (PID)
ps aux | grep geany # Find which PID has application to terminate
# This is the application - its PID we need
vojta 14639 9.3 0.8 2828512 134816 ?   Sl 16:12 0:01 /usr/bin/geany
# This is previous "ps aux | grep geany" command (last column)
vojta 14769 0.0 0.0   9440  1628 pts/0 S+ 16:12 0:00 grep geany
kill -SIGTERM 14639 # SIGTERM is "nice" termination, SIGKILL "brutal"
killall -SIGTERM geany # Select by name (more processes with same name)
# nice - how much resources will task use: from -20 (high priority - not
# "nice" process) to +19 (low priority - very "nice" process), default 0
nice -n 7 hard_task.sh # set priority 7 for newly launched task
renice 15 16302 # Change priority of PID 16302 to 15
sudo renice 15 16302 -u USER # Change priority of USER's process
```

Users

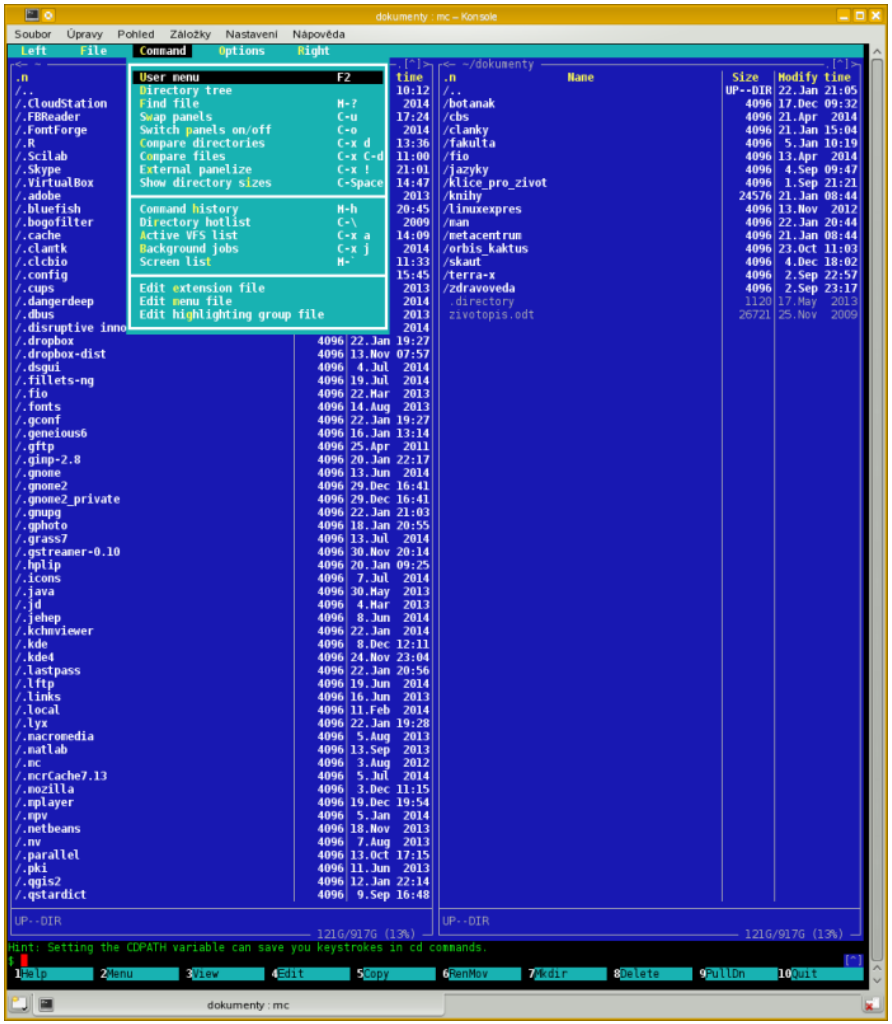
```
whoami # What is my user name
id # Information about current user (user ID and group IDs)
who # Who is logged in
w # Who is logged in, more information
users # Plain list of currently logged users
finger # Information about users on current terminals
last # Last logged-in users
passwd # Change password
passwd USER # Change USER's password
groups # List your groups
# Following commands to manage users and groups do not have to work
# on all systems - depends on authentication methods used
useradd newuser # Add new user
usermod --help # Modify user, see possible modifications
userdel user # Delete user
groupadd newgroup # Add new group
groupmod --help # Modify group, see possible modifications
groupdel group # Delete group
```

Directories

```
pwd # Print working directory - where we are right now
cd # Change directory (just "cd" or "cd ~" goes to home directory)
cd .. # One directory up; cd ../..; cd ../../another/directory/
cd relative/path/from/current/position # Go to selected directory
cd /absolute/path/from/root # Absolute path starts by "/"
tree # Tree like hierarchy of files and directories
tree -d # List only directories; see tree --help
tree -L 2 # Only up to second level; combine: tree -d -L 3
du -sh # Disk usage by current directory, -s for sum, -h for nice units
mkdir NewDirectory # Make directory
rmdir DirectoryToRemove # Remove empty directory
ls # List directory content
   # Try parameters -l, -a, -1, -F, -h (with -l or -s), --help
rm -r # Recursive delete - remove also non-empty directories
mv from to # Move files/directories (also for renaming)
cp from to # Copy, -r (recursive, including subdirectories)
           # -a (keeps all attributes), -v (verbose)
file somefile # Information about questioned file (what it is, ...)
xdg-open somefile # Open file by graphical application as in GUI
```

Midnight Commander

- `mc` to launch MC
- Move, copy, delete, files/directories, connect to SSH/(S)FTP, ...
- Can be used with mouse
- Edit text files (`F4`)
- `F2` for quick menu
- `F9` for top menu with many functions
- And much more...
- Not possible to live without it :-)



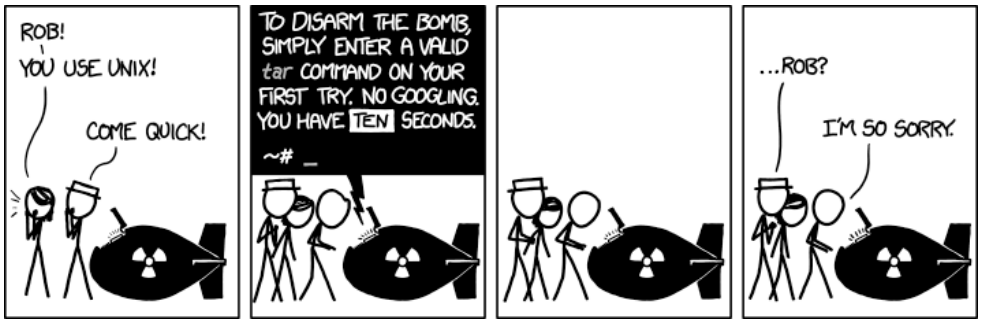
Compressing files into archives

Archive	**Compressing command**
*.tar	tar cvf archive.tar file1 file2
.tar.gz/.tgz	tar czvf archive.tar.gz/.tgz file1 file2
.tar.bz/.tbz/*.tar.bz2	tar cjvf archive.tar.bz/.tbz/.tar.bz2 file1 file2
*.tar.xz	tar cvf - file1 file2 | lzma > archive.tar.xz
*.gz	gzip file
*.bz2	bzip2 file
*.xz	lzma file
*.zip	zip -r archive.zip file1 file2
*.rar	rar a archive.rar file1 file2

- `gzip`, `bzip2` and `lzma` are able to pack only one file – use them together with `tar` to pack multiple files
- `gzip`, `bzip2` and `lzma` when used **without** `tar` move file into archive
- `lzma` has excellent compression, but sometimes is very slow

Compressing and decompressing archives

Archive	Decompressing command
*.tar	tar xvf archive.tar
.tar.gz/.tgz	tar xzvf archive.tar.gz/.tgz
.tar.bz/.tbz/*.tar.bz2	tar xjvf archive.tar.bz/.tbz/.tar.bz2
*.tar.xz	lzcat archive.tar.xz | tar xvf -
*.gz	gunzip archive.gz
*.bz2	bunzip2 archive.bz2
*.xz	unlzma archive.xz
*.zip	unzip archive.zip
*.rar	unrar x archive.rar



https://xkcd.com/1168/

Looking for files

```
locate somename # Searches for files/directories in local database
updatedb # Must be regularly launched to get "locate" to work
which # Full path to application (shell command)
whereis # Path to source code, executable and man pages for the command
# Test if executable command exists (good for scripts)
# If "Application" is missing, script ends with error
command -v Application >/dev/null 2>&1 || { echo >&2 "Application is
  required but not installed. Aborting." }
command -v find # Behaves like which, but reliable in scripts
type Application >/dev/null 2>&1 || { echo >&2 "Application is
  required but not installed. Aborting."; }
hash Application 2>/dev/null || { echo >&2 "Application is required
  but not installed. Aborting."; }
exit 1; # It can be added before the end of the bracket to send
        # term signal 1 - for better handling of various errors
```

Find

```
find # The most powerful searching tool, many parameters (man find):
find <where> -type d/f -name XXX -print
find photos/ -name *.jpg -exec mogrify -resize 1000x1000 '{}' \;
```

- First `find`'s parameter is location to search – absolute or relative, "`.`" means current directory (the only compulsory parameter)
- `-type` for only directories or only files (without this parameter, files as well as directories are looked for)
- `-name` supports wildcarts (`*`, `?` and `[…]`)
- `-print` is default action – prints list of results
- `-exec` runs some command with results (e.g. find all images and resize them)
  - All following arguments are argument of the command until "`;`" is encountered
  - `{}` is replaced by the current file name being processed
  - Those constructs might require protection by escape ("\") or quotes not to be expanded by shell

Network protocols

- SSH – secure shell – command-line connection to remote server to work there (port 22)
- Telnet – old deprecated insecure version of SSH, never ever use it (port 23)
- FTP – file transfer protocol – outdated, no encryption (port 21)
- FTPS – FTP with added connection encryption for higher security (port 21)
- SFTP – FTP over SSH – common, secure (port 22)
- SCP – secure copy – uses SSH, but has restricted possibilities, common, secure (port 22)
- NFS – network file share/server – very common in UNIX world, commonly used to permanently connect to network server, share directories, etc. (port 2049)
- webDAV – file transfer over web (using WWW server) – not so common, but good (port 80 or 443 – same as WWW)

Connect to SSH with key

- No need to remember password for every server...

```
# Create the key
ssh-keygen -t rsa -b 4096 # Good security, portable
# ECC gives better security, but not all servers/applications support it
ssh-keygen -t ecdsa -b 521 # Higher security
ssh-keygen -t ed25519 # Same security as ecdsa, higher performance
# Empty (no) passphrase will connect to server without password
# Copy public key to remote server (private key must be kept locally)
ssh-copy-id user@remote.server.cz # or
cat ~/.ssh/XXX.pub | ssh user@remote.server.cz \
  "mkdir -p ~/.ssh && cat >> ~/.ssh/authorized_keys"
# Now, public key is on the server and private key in local computer is
# unlocking the connection
# Unlock the key (no need in some distributions or if there is no
# passphrase) - must be done only once per user session
ssh-add
# Connect as usually
ssh user@remote.server.cz
```

Basic network information and testing

```
hostname # Get name of the computer
ping web.natur.cuni.cz # Ping host. Is it alive? Cancel by Ctrl+C
traceroute www.metacentrum.cz # Get route to the host
mtr hostname # Combines ping and traceroute, quit with "q"
ip a s # Information about all network devices (MAC, IP address, ...)
ifconfig -a # Older version of above command
iwconfig # List of network interfaces
ip r # Show routes
nc -vz web.natur.cuni.cz 22 # Does SSH work on the host?
  # verbose (-v), scan (-z), host, port number (22 for SSH, can be any)
man nc # See for more information; "nc" is alias for "netcat"
netstat -atn # Information about all network connections
netstat -ntplu # Show open TCP/UDP ports
netstat -anp # Show active connections
netstat -h # See for explanation of 3 above examples
nmap -r someserver # Scan someserver for opened ports
                   # If using at faculty, firewall disconnects you!
nmap botany.natur.cuni.cz --script ssh-hostkey # See SSH key
```

Transferring files from/to remote server

- `curlftpfs` allows mount FTP as local directory, but FTP is outdated, insecure and not constructed to that usage...

```
wget http://some.address.cz/internet # Download file(s) from Internet
wget --help # -r for recursive download (whole web), -k to convert links
# curl is predecessor of wget, without parameter "-o" it prints remote
# content to standard output (typically screen)
curl http://some.server.cz/some/files -o localfilename
# Copy files (-r for recursive) over SSH from local computer to
# remote server or vice versa (just flip arguments)
scp -r localfiles remoteuser@remote.server.cz:/remote/path/
scp -r remoteuser@remote.server.cz:/remote/files /local/directory/
# scp behaves like cp, but works over SSH
```

- `rsync` is synchronization tool (commonly used for backups) able to co connect to remote server (next slide)

Synchronization with rsync

- `rsync` has huge amount of possibilities (see `man rsync` or `rsync -h`)
- Works locally as well as over network
- It transmits only changes – very efficient
- Suitable for local as well as network backup
- Network address for rsync is written in same way as for `scp`
- `--delete` delete in target location files which are not in source location any more
- `--progress` show progress percentage for every file
- `--exclude=*.jpg` skip JPG files
- For incremental backups use `duplicity`

```
1 rsync -arv somedirectory otherplace # All attributes, recursive, verbose
2 rsync -arv localdirectory user@remote.server.cz:/remote/directory/
3 rsync -arv user@remote.server.cz:/remote/data local/directory/
```

Connecting file systems on remote servers

- Mounting and unmounting of remote servers require root privileges
- Target mount point must exist before mounting
- Servers can be accessed by IP address or hostname

```
# Mount Windows server
mount.cifs //windows.server.cz/Some/Directory /mnt/win -o \
  credentials=/path/to/password.file,uid=USER,gid=GROUP
# "password.file" contains login credentials to Windows server:
username=user.name
password=TopSecretPassword1
domain=DOMAINNAME
man mount.cifs # See for other connection options
# Mounting remote server over SSH (sshfs package must be installed)
sshfs user@vyuka.natur.cuni.cz:/some/dir /local/mount/point
# Mount NFS share (NFS is common protocol in UNIX world)
mount -t nfs some.server.cz:/shared/directory /local/directory
# Mount webDAV folder (requires package davfs2 to be installed)
mount -t davfs https://owncloud.cesnet.cz/directory /local/directory
```

Parallelisation with GNU Parallel

- GNU Parallel can distribute task among CPU threads of one computer, or even among different computers in network
- It is not effective for short/small tasks
- Important operands (for more see `man parallel`)
  - `{}` – input line – whole line read from input source (typically standard input)
  - `{.}` – input line without extension
  - `{/}` – base name of input line – only file name (without path)
  - `{//}` – dirname from input line (filename is removed)
  - `{/.}` – base name of input line without extension
  - `:::` – use arguments from command line instead of stdin (::: is placed after the command and before the argument)
  - `::::` – read from argument files
  - `-j` – number of jobs – if not provide, `parallel` will use all available CPU threads

GNU Parallel examples I

```
# Convert all images from JPG to PNG
ls -1 *.jpg | parallel --bar convert '{}' '{.}.png'
# Resize all images ("\" marks that command continue on next line)
find . -name '*.jpg' -print | parallel convert -resize 1000x1000 \
  -quality 75 '{}' '{.}-small.jpg' # or
parallel convert -resize 1000x1000 '{}' '{.}-small.jpg' ::: *.jpg
# Find WORD in huge text file (named "longfile" here) - this works
# but it is not possible to get line number (file is red in blocks)
parallel --pipe --block 10M -- grep --color=always WORD < longfile
# Same as above but add line numbers according to original file
nl longfile | parallel -k --pipe --block 20M -- grep WORD
# When needed to get phrase or regular expression (use parameter
# "-q" for escaping of shell special characters or extra quotes):
# "--" stops reading parameters for parallel
nl longfile | parallel -qk --pipe --block 20M -- grep "WORD TEXT" # or
nl longfile | parallel -k --pipe --block 20M -- grep '"WORD TEXT"'
```

GNU Parallel examples II

```
# Run in parallel commands from command list file (list of commands)
parallel < command_list.txt # (each command on one line) or
parallel :::: command_list.txt
# Add same text to the end of multiple files
ls -1 *.txt | parallel 'cat block_to_be_added.txt >> {}'
# Replace particular text in multiple files with sed and GNU Parallel
ls -1 *.txt | parallel 'sed -i "s/XXX/YYY/g" {}'
# Launch MrBayes for multiple nexus files and create log file with
# starting and ending date and time
ls -1 *.nexus | parallel 'echo Start > {}.log && date >> {}.log && \
  mrbayes {} | tee -a {}.log && echo End: >> {}.log && date >> {}.log'
# tee (-a for append to existing file) records output of MrBayes
command | tee record.txt # tee will record whole output of command
tee record.txt | command # tee will record user input
# If software reads commands from user, we can reuse record next time:
command < record.txt # Empty lines are interpreted as Enter key
                     # Each line is used whenever command waits for new
                     # input (instead of user typing, record.txt is used)
```

Removable media

- Mounting and unmounting of devices require root privileges
- In Linux, physical disks are named from `sda` to `sdz`, each disk has partitions (at least one) numbered from `1`, e.g. `sda1`, `sda2`, `sdb1`, … – all are accessible in `/dev` directory (`/dev/sdc3`, …)
- Target mount point must exist before mounting

```
1  eject # Open CD/DVD drive
2  mount # Which FS (disk partitions) are mounted
3  findmnt # See mounted devices in tree-like structure
4  # mount usually recognize FS of mounted device, if not, us -t FS_type
5  mount /dev/sdXY /mount/directory # Mount disk sdXY to /mount/directory
6  umount /dev/sdXY # Unmount disk sdXY
7  umount /mount/directory # Unmount disk from /mount/directory
8  dmesg | grep sd | tail # Get information about recently plugged media
9  mkdir /mnt/iso # Directory must exist prior mounting into it
10 # Mount CD/DVD ISO image file into directory /mnt/iso
11 mount -t iso9660 -o loop file.iso /mnt/iso # See CD/DVD image content
```

Other commands

```
touch filename # Creates empty text file
echo # Write empty line of text
echo $USER # Write value of variable $USER
echo "Some text" # Write text in quotes
# dd produces physical copy of whole device - including empty space
dd if=/dev/sdXY of=image.iso # Backups disk sdXY to imago.iso
dd if=image.iso of=/dev/sdXY # Used to write image of Linux live
                             # media to USB flash disk
apropos keyword # Searches for command descriptions containing keyword
dmesg # Recent entries in main system log - filter with grep, tail, ...
lnav # Comfortably browse recent logs, quit by "q"
```

Read text file

```
cat # Read or join files (-n adds line numbers, -v prints non-printable
    # characters like EOL)
cat textfile # Print content of text file
cat textfile1 >> textfile2 # Add textfile1 to the end of textfile2
nl textfile # Like cat -n, prints textfile with line numbers
tac textfile # Like cat, but prints lines in reverse order
more textfile # When textfile is long, prints screen by screen (space
              # for next screen, q to quit)
less textfile # Better version of more - you can scroll up and down by
              # PgUp, PgDown, arrows, searching by / (type searched
              # string, hit Enter, n for next, twice ESC to quit),
              # q to quit viewing
most textfile # Better version of less
fmt textfile # Basic formatting of text - joining of commented lines,
             # line breaks to break too long lines, ...
fmt textfile > formatted_file # Save output of fmt into new file
wc textfile # lines, words and bytes in text file
            # wc -l for only lines, -m for characters, -w for words
```

Get part of text file

```
head -n N textfile # Print first N lines from textfile
tail -n N textfile # Print last N lines from textfile
head -n-N textfile # Print textfile without last N lines
tail -n+N textfile # Print textfile from Nth line to the end
# Split text file on selected pattern - creates new files xxXY
csplit textfile '/pattern/' '{*}' # pattern itself is inside '/___/'
# Pattern can be regular expression - set it carefully
# {*} says to repeat operation as many times as possible
grep -parameters pattern textfile # Write lines containing pattern
grep user /etc/passwd # Write all lines in passwd containing user
cat /etc/passwd | grep user # Same as above
grep -v user /etc/passwd # Write all lines in passwd NOT containing user
grep -c user /etc/passwd # get number of lines in passwd containing user
grep -i USER /etc/passwd # -i isn't case sensitive
grep -q ... # quiet - no output - good for testing in scripts
grep -ls user /etc/* # -l print files with pattern, -s suppresse errors
grep "longer text" textfile # Extract whole phrase
```

Grep supports regular expressions, slide 105.

Get a column – cut, awk

```
cut column/delimiter+field textfile
cut -c 1 /etc/group # Get first character
cut -c 1-5 /etc/group # Get character 1-5
cut -c 4- /etc/group # Get character 4 and more
cut -c 2,5,7 /etc/group # Get characters 2, 5 and 7
cut -d ':' -f 1 /etc/group # Select 1st field separated by ":"
cut -d ':' -f 2-4 /etc/group # Select fields 2-4 separated by ":"
cut -d ':' -f 1,3 /etc/group # Select fields 1 and 3 separated by ":"
awk 'regexp { commands parameters }' file
awk '{print $NF}' textfile # Select last column (separated by tab)
awk '{print $2}' textfile # Select 2nd column (separated by tab)
awk '{print $3, $2}' textfile # Print columns 3 and 2 (in this order)
awk -F ' ' '{print $4, $1}' textfile # Print columns 4 and 1
                                     # separated by " " (space)
ls -l | awk '/^d/ { print $8 "\t" $3 }' # Separate columns by TAB
           # /^d/ for lines starting with "d" (only directories)
```

- For regular expressions, see slide 105
- BASH can not select column according to its name (Perl can do that)

More about AWK

- AWK is scripting language for text manipulations
- Only basic examples here...

```
# Print on each line ">", field 1, make new line (\n) and print field 2
awk '{print ">"$1"\n"$2}' textfile
# Print field 1, TAB (\t), length of field 2, TAB and field 2
awk '{print $1"\t"length($2)"\t"$2}' textfile
# If field 1=0, print whole line
awk '{if($1==1){print $0}}' textfile
# Field 1 is numeric (less then 5 digits) - add leading zeroes
awk '{printf "%05d\n", $1;}' textfile
# As above, but add leading zeroes to field 1 and print whole line
awk '{$1=sprintf("%05d", $1); print $0}' textfile
# Field 1 is numeric, select lines where field 1 is higher than 100
awk '$1>100' textfile
# Print field 4 (fields are separated by "_" or TAB)
awk -F '[_\t]' '{print $4}'
```

Sorting

```
sort textfile # Sort a text file
sort -d textfile # Take into account only spaces and alphanumerical
                 # characters (ignore any other characters)
sort -r textfile # Reverse order
sort -f textfile # Ignore character case (not case sensitive)
sort -m textfile1 textfile2 # Merge already sorted text files
sort -u textfile # Print only first of multiple entries
sort -b textfile # Ignore leading blanks (space on beginning of line)
# Sorting is influenced by locale setting (e.g. Czech ``ch'')
# To force use of English locale use
LC_ALL=C sort ... # Set for this command language variable to English
uniq textfile # Filters following identical lines - only unique
              # are printed (to get unique lines from whole file,
              # sort it first)
uniq -c textfile # Add number of occurrences before each line
uniq -d textfile # Print only repeated lines
uniq -i textfile # Ignore case (not case sensitive)
uniq -s N textfile # Skip first N characters
uniq -u textfile # Print only not-repeated lines
```

Replacements – tr

- `tr` replaces or deletes characters from standard input and writes result to standard output – use pipes and/or redirects:

```
# Replace space by TAB in inputtextfile, save result as outputtextfile
cat inputtextfile | tr " " "\t" > outputtextfile
# Delete "text" from each line and print it to standard output (screen)
cat inputtextfile | tr -d "text"
# Replace every occurrence of A, B, C or D by a new line (\n)
cat inputtextfile | tr "[ABCD]" "\n" > outputtextfile
tr --help # See possible replacement patterns
# Replace capital letters by small ones
tr "[A-Z]" "[a-z]" < inputtextfile > outputtextfile
# Alternative (easier reading) of previous command:
cat inputtextfile | tr "[A-Z]" "[a-z]" > outputtextfile
# Alternative solution using sed (next slide)
sed 's/[A-Z]/\L&/g' inputtextfile > outputtextfile # Vice versa:
sed 's/[a-z]/\U&/g' inputtextfile > outputtextfile
```

Replacements – sed I

```
sed 's/FindToReplace/Replace/modificator' textfile > newtextfile
# Search and replace ("s") all occurrences ("g") of "find" by "replace"
sed 's/find/replace/g' textfile
# Replace third occurrence of pattern on every line
sed 's/pattern/ToReplace/3'
# To work only on particular line, place number or range (e.g. 1,7)
# right before "s" (sed '1,7s/...')
sed '1~2n;s/F/R/g' # Work on every second line, starting by line 1
sed -n -e '2~10p' # Print every 10th line, starting with line 2
seq 1 100 | sed -n -e '2~10p' # Example of above pattern
# Replace first TAB (\t) on each line by new line (\n)
sed 's/\t/\n/' textfile
sed 's/find/replace/g' somedirectory/* # Work on all files in directory
sed -i ... # Modify processed file - otherwise output is printed on
           # standard output (screen - can be piped into text file, ...)
```

- `sed` supports regular expressions, see slide 105 (same as in `grep` and `vim`), with parameter `-e` can use extended regular expressions (Do not confuse!)

Replacements – sed II

```
# Groups to remember work in same way in sed, grep as well as vim
\(ToRemember\) # Remember expression in brackets
\Number # Use remembered expression (numbered from one)
# Take output of ls -l and replace # "users" by "users-RULEZZZ"
ls -l | sed 's/\(users\)/\1-RULEZZZ/g'
# Add "size:TAB" before file size column (we suppose it has 3-8 digits)
ls -l | sed 's/\([0-9]\{3,8\}\)/size:\t\1/g'
```

Comparisons

```
# Compare two sorted columns
comm textfile1 textfile2
  # 1st column - lines only in textfile1
  # 2nd column - lines only in textfile2
  # 3rd column - lines in both files
comm -2 textfile1 textfile2 # Don't show 2nd column (similarly -1, -3)
# Show differences between text files
diff textfile1 textfile2
  # First number shows line(s) in 1st file, then if add/delete/change
  # and last number shows line(s) in the second file, <> show direction
diff -e textfile1 textfile2 # More simple output
diff -c textfile1 textfile2 # Show context (lines around change)
diff -u textfile1 textfile2 # Better version, the most common
diff -y textfile1 textfile2 # In two columns comparing both files
colordiff # Same usage and parameters as previous, colored output
diff -u textfile1 textfile2 | view - # Launches vim (exit by :q! Enter)
vimdiff # Can show more colors, launches vim (exit by :q! Enter)
```

Joining

```
# Join compares every lines and creates all combinations -
# ensure to have sorted input files with unique text
# Compare two sorted text files and write shared lines
# (duplicitous lines are shown just once)
join textfile1 textfile2
# Add file to the end of another text file
cat file1 >> file2 # file2 will contain both files, file1 is unchanged
# Add second file as second column
paste file1 file2 > outputfile
# Output will contain two columns (consisting of file1 and file2)
# separated by TAB
```

Editors

- `nano`, `pico` and `mc` are very simple, just for very basic text editing in command line or until you learn `vim` (graphical version is **gVim**) or `emacs` (graphical version is also available, just search for **Emacs** in your distribution software manager)
- You can work most of the time in graphical editors (slide 49)
- Emacs and Vim are extremely rich, but having completely different approach – when you get use to one, you can't use the another

```
1 nano textfile # Basic simple text editor
2 pico textfile # Clone of nano, same basic thing
3 mc # Use its internal editor, just very basic
4 emacs textfile # Extremely feature rich (including file browser
5                # and many tools), Exit by Ctrl+X and Ctrl+C
6 vim textfile # Probably the most common, as rich as Emacs
7 vimtutor # Launch tutorial to learn Vim (in various languages)
```

Vim

- Modes of `vim`:
  1. "Normal" – nothing is displayed in bottom left corner, every key has some meaning (`dd` to cut current line, `r` to replace character below cursor, `v` for selection of text, `y` to copy, `x` to cut, `p` to paste, `i` or `Insert` key to enter insert mode, `:` to enter command mode, number to get to line of particular line number, `u` to undo last change(s), ...)
  2. Insert – in bottom left corner "`-- INSERT --`" is displayed – the most familiar mode – normal typing etc., exit to normal mode by `ESC` key
  3. Command – in bottom left corner `:` is displayed – awaits commands, e.g. `w` to write file, `q` to quit, `q!` to quit and discard changes, `%s/...` to search and replace as in `sed`, `syntax on/off` to turn syntax highlight on/off, `/` to search, ... Exit to command mode by `Backspace` key (delete "`:`").
- For more information see http://www.vim.org/ and http://vim.wikia.com/wiki/Vim_Tips_Wiki
- In Czech http://www.nti.tul.cz/~satrapa/docs/vim/

Regular expressions are useful...

- Find text according to a pattern
- Manipulate the text – flip, reformat, replace, ...
- Syntax is variable among programming languages and applications
- Probably the most advanced is Perl

https://xkcd.com/208/

Regular expressions I

- . – any single character
- * – any number of characters/occurrences of pattern (including 0)
- + – one or more occurrences of the preceding reg exp
- ? – zero or one occurrences of the preceding reg exp
- [...] – any character in the brackets
- [^...] – reverse case – all characters except newline and those listed in brackets
- ^ – first character of reg exp – beginning of the line
- $ – last character of reg exp – end of the line
- \{n,m\} – range of occurrences of single character
- \{n\} – exactly *n* occurrences
- \{n,\} – at least *n* occurrences

Regular expressions II

- \ – escape following special character
- | – either the preceding or following reg exp can be matched (alternation)
- \(...\) – group reg exp (numbered, starting with 1) – can be called by *n*, where *n* is number of the group (starting with 1)
- \< \> – word boundary
- [[:alnum:]] – alphanumerical characters (includes white space), same like [a-zA-Z0-9]
- [[:alpha:]] – alphabetic characters, like [a-zA-Z]
- [[:blank:]] – space and TAB
- [[:cntrl:]] – control characters
- [[:digit:]] – numeric characters, like [0-9]
- [[:graph:]] – printable and visible (non-space) characters

Regular expressions III

- [[:lower:]] – lowercase characters, like [a-z]
- [[:print:]] – printable characters (includes white space)
- [[:punct:]] – punctuation characters
- [[:space:]] – white space characters
- [[:upper:]] – uppercase characters, like [A-Z]
- [[:xdigit:]] – hexadecimal digits
- ^$ – blank line
- ^.*$ – entire line whatever it is
- * – one or more spaces (there is space before asterisk)
- & – content of pattern that was matched
- Implementation in `vim`, `sed`, `grep`, `awk` and `perl` and among various UNIX systems is almost same, but not identical...

Regular expressions IV

- **grep**, **sed** and **vim** require escaping of +, ?, {, }, (and) – **egrep** (extended version, launched as `grep -E` or `egrep`) and **perl** not
- Read http://www.regular-expressions.info/, in Czech http://www.nti.tul.cz/~satrapa/docs/regvyr/, http://www.root.cz/serialy/regularni-vyrazy/ and http://www.regularnivyrazy.info/, and manuals for Grep, Vim, Sed, Awk, Perl, ...
- Mac OS X has by default very outdated version of `sed` and another tools – it does not have all advanced features – users need to install e.g. `gnu-sed` formulae from Homebrew

Basic script

- Every script begins with `#!/bin/bash` (or alternative for another shells, Perl, ...)
- Add any commands you like...
- Every script should end with `exit` (but it is not necessary)
- After writing the script, add execution permission (`chmod o+x noninteractive.sh` or `chmod +x noninteractive.sh`)
- The most simple script:

```
1 #!/bin/bash
2 # Simple non-interactive script - no communication with user
3 # only list of commands
4 echo "Hi, $USER, today is `date` and your PATH is $PATH."
5 echo
6 exit
```

Functions in BASH

- Pieces of code, which can be used repeatedly

```
# Declare new function
function MyNewFunction1 {
  echo "Hello, $USER on $HOSTNAME!"
  }
# Use it in a script as any other command:
...
MyNewFunction1
...
# Use with variables (same as in previous cases)
function MyNewFunction2 {
  echo "The sum is `expr $1 + $2`."
  }
# Use it in the script
...
MyNewFunction2 5 8 # For example
...
```

Script reading two variables

```
#!/bin/bash
# Arguments are read from command line as parameters of the script
# Order has to be kept (well, not in this case, but generally yes)
echo "Sum of two numbers $1 and $2 is `expr $1 + $2`."
# "$#" is available every time and contains number of parameters
# (variables) given to the script
echo "Number of parameters is $#."
# "$*" is available every time and contains all supplied parameters
echo "Those parameters were supplied: $*."
#  "$0" is available every time and contains script path
echo "Path to the scrip is: \"$0\"."
echo
exit
```

When done, do:

```
chmod +x interactive1.sh
./interactive1.sh 8 9 # Or select any other two numbers
```

- There is no checking of input values, nothing advanced, ...

Variables will be interactively provided by the user

```
#!/bin/bash
# Arguments are read from user input (script asks for them)
echo "Please, input first value to sum and press Enter"
read V1
echo "Please, input second value to sum and press Enter"
read V2
echo "Sum of two numbers $V1 and $V2 is `expr $V1 + $V2` ."
echo
exit
```

When done, do:

```
chmod +x interactive2.sh
./interactive1.sh # Values will be provided when script asks
```

- There is no checking of input values, nothing advanced, ...

```
$(expr $1 + $2) # Alternative - $(...) is same as `...`
```

Provide named parameters

```
#!/bin/bash
# Script has only one parameter ($1) provided as its parameter
case "$1" in # evaluating provided parameter and behaving accordingly
  -d|--disk) # "|" means alternatives - more possible inputs
    echo "Your disk usage is:"
    df -h
    ;;
  -u|--uptime)
    echo "Your computer is running:"
    uptime
    ;;
  # This should be every time last possibility - any other input
  *) # User is then notified he entered nonsense and gets some help
    echo "Wrong option!"
    echo "Usage: -d or --disk for available disk space or"
    echo "-u or --uptime for computer uptime"
    exit 1;; # In this case, exit with error code 1
esac
exit
```

Notes to previous script

- First make `interactive3.sh` executable and launch it via e.g. `./interactive3.sh -d` or `./interactive3.sh --uptime` or so
- Function `case` has basic checking of input available – as last parameter use `*)` – any other input except those defined above will produce some warning message, error or so
- In same way can be added more parameters (by multiple use of `case`), but order of parameters must be kept and all parameters are compulsory
- `case` can evaluate simple regular expressions, e.g. `--[Uu]ptime)`, `-d*`, ...

Provide parameters, verify them and behave accordingly I

```
1  #!/bin/bash
2  NUMBER='^[0-9]+$' # From beginning (^) to the end ($) only number(s)
3  function usagehelp { # Function to print help
4    echo "Usage: number1 plus/minus/product/quotient number2"
5    echo "Use plus for sum, minus for difference, product"
6    echo "for multiplication or quotient for quotient."
7    exit 1
8    }
9  if [ "$#" -ne "3" ]; then # Do we have 3 parameters provided?
10     echo "Error! Requiring 3 parameters! Received $# ($*)."
11     usagehelp # The function to print help
12   fi
13 if [[ ! $1 =~ $NUMBER ]]; then # Is parameter 1 number?
14     echo "Parameter 1 is not an integer!"
15     usagehelp # The function to print help
16   fi
```

Continues on next slide...

Provide parameters, verify them and behave accordingly II

Remaining part from previous slide...

```
if [[ ! $3 =~ $NUMBER ]]; then # Is parameter 3 number?
    echo "Parameter 3 is not an integer!"
    usagehelp # The function to print help
  fi
case "$2" in
  plus) expr $1 '+' $3;;
  minus) expr $1 '-' $3;;
  product) expr $1 '*' $3;;
  quotient) expr $1 '/' $3;;
  *) echo "Wrong option!"
     usagehelp # The function to print help
     ;;
esac
exit
```

`chmod +x interactive4.sh && ./interactive4.sh 7 plus 5` (for example)

If branching

```
1  # Basic variant - commands are done only if condition is met
2  if expression; then
3      commands
4    fi
5  # Two branches - when condition is met and when not
6  if expression; then
7      commands1 # expression is TRUE
8    else
9      commands2 # expression is FALSE - all other cases
10   fi
11 # Join together two (or more) if branches
12 if expression1; then
13     commands1
14   elif expression2
15     then
16       commands2
17     else
18       commands3
19     fi
```

Evaluation of conditions I

- "`[ ... ]`" (always keep space around it – inside) is function to evaluate expressions (alternatively use command `test`)
  - `if [ "$VAR" -eq 25 ]` or `test $VAR -eq 25`
  - `if [ "$VAR" == "value" ];`
    - Escaping variables and values by double quotes (`"..."`) is recommended (to be sure), but not strictly required all the time
  - `if [ ! -f regularfile ];` – reverted condition (`!`)
  - Single-bracket conditions – file, string, or arithmetic conditions
  - Double-bracket syntax – enhanced
    - Allow usage of regular expressions and globbing patterns
    - Word splitting is prevented – `$STRINGVAR` can contain spaces
    - Expanding file names – `if [[ -a *.sh ]]` (variant with only one bracket doesn't work when there are multiple sh files)
    - Allows more detailed test, e.g. `if [[ $num -eq 3 && "$STRINGVAR" == XXX ]]`

Evaluation of conditions II

- `-eq` – Equal to
- `-lt` – Less than
- `-gt` – Greater than
- `-ge` – Greater than or equal to
- `-le` – Less than or equal to
- `-f $FILE` – True if `$FILE` exists and is a regular file
- `-r $FILE` – True if `$FILE` exists and is readable
- `-w $FILE` – True if `$FILE` exists and is writable
- `-x $FILE` – True if `$FILE` exists and is executable
- `-d $FILE` – True if `$FILE` exists and is a directory
- `-s $FILE` – True if `$FILE` exists and has a size greater than zero
- `-n $STR` – True if string `$STR` is not a null string

Evaluation of conditions III

- `-z $STR` – True if string `$STR` is a null string
- `$STR1 == $STR2` – True if both strings are equal
- `$STR` – True if string `$STR` is assigned a value and is not null
- `$STR1 != $STR2` – True if both strings are unequal
- `-a` – Performs the `AND` function
- `-o` – Performs the `OR` function
- Do not confuse globbing patterns and regular expressions when using `[[ ]]`
  - Shell globbing: `if [[ "$STRINGVAR" == ?[sS]tring* ]]; then` – `?` represents single character `[]` any character inside and `*` zero or more characters
  - Regular expressions: `if [[ "$STRINGVAR" =~ .[sS]tring.* ]]; then` – `.` represents single character (`?` would be zero or one occurrence of preceding expression), `[]` any character inside and `.*` zero or more occurrences of any single characters

For, while and until cycles

```
# One line for cycle for resizing of images
for file in *.jpg; do convert $file -resize 100x100 thumbs-$file; done
# More commands in a block
for file in `ls -1 *.jpg`; do
  echo "Processing file $file"
  convert $file -resize 100x100 thumbs-$file
  echo thumbs-$file created
  done
# while cycle is evaluating expression and if it is equal to 0
# the cycle body is launched, repeatedly while the condition is met
while expression
  do
    commands
  done
# Like while cycle, but until expression is not equal to zero
until expression; do
  commands
  done
```

Package management I

Installation of software

- Package – an application or its part (documentation, plug-ins, translations, ...)
- Packages are available in repositories (directories) on the internet
  - System has list of applications available
  - Updates and bug fixes are installed for all applications using one interface (GUI or command line) – very reliable
  - Packages are digitally signed – security
  - User can set custom repositories to get new packages
- The most different task among distributions
- Packages have dependencies – required shared libraries and so on – use package manager and try to avoid downloading packages from the internet
- Read manual for your distribution!

Package management II

Installation of software

- Package is basically an archive and system has configured directories where to unpack it – binaries are commonly in `/usr/bin/`, shared libraries in `/lib` and `/lib64` and rest in `/var`
- User should not care where parts of packages go to – system is taking the care – user can only damage it
- Shared libraries are installed automatically whenever some application needs them
- As all files are placed in standard defined directories, it is very simple to use them also for another applications
- Applications not available in repositories, neither as distributional package should be installed into `~/bin` for current user or `/usr/local` for all users (binaries then go into `bin` and so on)

Package management in command line in openSUSE and Debian/Ubuntu

Root password is required: use `sudo` or `su -`

Task	**openSUSE**	**Debian/Ubuntu**
Package name	*.rpm	*.deb
Install	zypper in *package*	apt-get install *package*
Remove	zypper rm *package*	apt-get remove *package*
Refresh repositories	zypper ref	apt-get update
Update	zypper up	apt-get update
Upgrade	zypper dup	apt-get dist-upgrade
Search	zypper se *term*	apt-cache search *term*
Clear packages	rpmorphan	apt-get autoremove
Interactive manager	yast sw_single	aptitude
Add repository	zypper ar *repository*	nano /etc/apt/sources.list
Remove repository	zypper rm *repository*	nano /etc/apt/sources.list

Basics of compilation

- Some software is distributed only as source code written in languages like C or C++ – user has to compile it to get binary executable
- Compilation creates binary specific for particular operating system and hardware platform – can be tuned for optimal performance
- Interpreted languages like Bash, Perl, Python or Java don't have to be compiled (but it is possible) – they need their interpreter to run, relative easily portable among hardware platforms and OS
- Applications requiring compilation usually have good instructions

```
# General schema:
configure # Many possible parameters, settings for compilation
          # Not required in every time
make # Basic building command, sometimes only this is required
make install # Final creation of binary, sometimes required
```

- If you don't have to do it, don't do it. Solving problems can be complicated – contact someone skilled or author of the application...

Compilation of RAxML

Available from https://github.com/stamatak/standard-RAxML (cite Stamakis 2014).

```
# Create working directory
mkdir raxml
# Go there
cd raxml/
# Get source code from GitHub (svn downloads only changed files)
svn co https://github.com/stamatak/standard-RAxML/tags/v8.2.4
# Go to newly created directory
cd v8.2.4/
ls # List files
# No need of Windows version - delete it
rm -rf Windows*
# Compile standard version (other versions are available for better CPU)
make -f Makefile.gcc
# Remove unneeded files
rm *.o
# Launch it - see RAxML help
./raxmlHPC -h
```

Launching Java applications

- Java is probably the most portable language working on any operating system – the only condition is to install Java virtual machine (JVM)
- Linux usually use OpenJDK – search for packages named `*openjdk*`
- Let's download e.g. FigTree from http://tree.bio.ed.ac.uk/download.php?id=90&num=3

```
# Go to directory where you downloaded it
cd directory/with/downloaded/figtree
# Decompress downloaded archive
tar zxvf FigTree_v1.4.2.tgz
# Go to created directory
cd FigTree_v1.4.2/
ls * # List files, also in subdirectories
# Launch it (command java launches *.jar files)
java -jar lib/figtree.jar
# Limit its memory usage to 512 MB
java -Xmx512m -jar lib/figtree.jar
```

CESNET and MetaCentrum I

- CESNET is organization of Czech universities, Academy of Science and other organizations taking care about Czech backbone Internet, one of world leading institutions of this type
- CESNET provides various services
  - Massive computations – MetaCentrum
  - Practically unlimited data storage
  - FileSender to be able to send up to 500 GB file
  - ownCloud to backup and/or sync data across devices (default capacity is 100 GB)
    - It is possible to connect by webDAV to ownCloud (slide 84) – many applications support it
  - And more...
- Services accessible without registration
  - ownCloud `https://owncloud.cesnet.cz/`
  - FileSender `https://filesender.cesnet.cz/`

CESNET and MetaCentrum II

- Go to web and log in with your institutional password

- Services requiring registration (and approval)
  - To use MetaCentrum fill registration form https://metavo.metacentrum.cz/en/application/form
  - To use data storage fill registration form https://einfra.cesnet.cz/perun-registrar-fed/?vo=storage&locale=en
  - Users not having access to EduID have to register first at HostelID https://hostel.eduid.cz/en/index.html
  - Note some browser do not have required certificate and registration pages do not work correctly. Mozilla Firefox should be safe choice every time.

- Information about data storage https://du.cesnet.cz/en/start contains detailed usage instructions

- Information about MetaCentrum https://www.metacentrum.cz/en/

CESNET and MetaCentrum III

- Most of practical information for users are at wiki
  https://wiki.metacentrum.cz/w/index.php?&setlang=en

MetaCentrum

- Also available is Galaxy https://galaxy.metacentrum.cz/galaxy/ (same login as to MetaCentrum) – web based bioinformatics framework (more information at wiki)
- Current state and usage as available at https://metavo.metacentrum.cz/en/
- Manage your user account at http://metavo.metacentrum.cz/en/myaccount/index.html
- Personal view on actual resources and running tasks is at https://metavo.metacentrum.cz/pbsmon2/person
- List of available applications https://wiki.metacentrum.cz/wiki/Kategorie:Applications
- It has 8 front ends where users log and thousands of computers doing the calculations – they are not accessed directly
- Most of computers are running Debian GNU/Linux

MetaCentrum usage

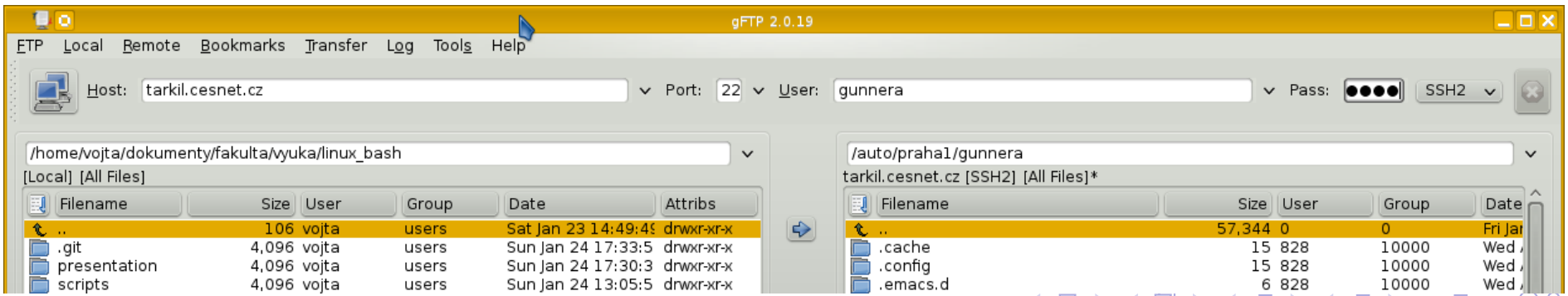
- User can transfer data on one of frontends by `scp` or for example WinSCP from Windows
- Same credentials are used for all front ends, for SSH login as well as file transmissions

```
# Login to selected server (tarkil is located in Prague)
ssh -X USER@tarkil.cesnet.cz
# Continue as in any other command line...
```

- In home directory on the server prepare all needed data and non-interactive script (interactive are more complicated) which will do the calculations
- Tasks are not launched immediately, but using `qsub` – task is submitted into queue and system decides when it will be launched

File transfers to MetaCentrum

- Graphical applications: gFTP, FileZilla or from most of file managers
- Protocol is SSH/SSH2/SFTP/SCP, port 22, server is selected frontend's address (e.g. `tarkil.cesnet.cz`) – it is recommendable to use all the time same frontend
- All servers are accessible under domain `*.metacentrum.cz`: `skirit`, `perian`, `onyx`, `zuphux` (located in Brno), `nympha`, `minos` (in Pilsen), `hermes` (in České Budějovice) and `tarkil` (in Prague) – so that e.g. `tarkil.cesnet.cz` is synonymous to `tarkil.metacentrum.cz`
- See slide 84 and following to command-line transfers of files



Basic skeleton of script running tasks I

```
#!/bin/bash
# Modify the script according to your needs!
# Set data directories
WORKDIR="bayes_batch"
DATADIR="/storage/praha1/home/$LOGNAME"
# So there is directory /storage/praha1/home/gunnera/bayes_batch
# (in this case) containing all the data needed for calculations
# Clean-up of SCRATCH (it is temporal directory created by server) -
# the commands will be launched on the end when the job is done
trap 'clean_scratch' TERM EXIT
trap 'cp -ar $SCRATCHDIR $DATADIR/ && clean_scratch' TERM
# Prepare the task - copy all needed files from working directory
# into particular computer which will finally do the calculations
cp -ar $DATADIR/$WORKDIR/* $SCRATCHDIR/  || exit 1
# Change working directory - script goes to the directory where
# calculations are done
cd $SCRATCHDIR/ || exit 2 # If it fails, exit script
```

Ends on following slide...

Basic skeleton of script running tasks II

...begins on previous slide.

```
# Prepare calculations - load required application modules
# See https://wiki.metacentrum.cz/wiki/Kategorie:Applications
# Every application module is loaded by "module add XXX"
. /packages/run/modules-2.0/init/sh
module add parallel # In this case GNU Parallel and MrBayes
module add mrbayes-3.2.4
# Launch the analysis - calculate MrBayes for multiple files
# Note Parallel will distribute task among 8 CPU threads (-j 8),
# so that qsub must in this case contain nodes=1:ppn=8 (see further)
ls -1 *.nexus | parallel -j 8 'mb {} | tee -a {}.log'
# Copy results back to home directory
cp -ar $SCRATCHDIR $DATADIR/$WORKDIR || export CLEAN_SCRATCH=false
# This is all needed, the script is ready to be launched...
exit
```

- Make `metacentrum.sh` executable and modify it to fit your needs...
- If it was written on Windows, convert EOL (and encoding)...

Launching of tasks

- https://wiki.metacentrum.cz/wiki/Running_jobs_in_scheduler
- Personal view https://metavo.metacentrum.cz/pbsmon2/person has nice overview of available resources and tasks and allows comfortable construction of submission command

```
# We will run up to 5 days, require 4 GB of RAM, 5 GB of disk space, one
# physical computer with 8 CPU threads and we get all information mails
qsub -l walltime=5d -l mem=4gb -l scratch=5gb -l nodes=1:ppn=8 -m abe \
  bayes_batch.sh
# Check how the task is running (above web) and
qstat | grep $USER
qstat -u $USER
qstat 123456789 # The task ID is available from commands above or mail
qstat -f 123456789 # Print a lot of details
qdel 123456789 # Terminate scheduled or running task
qfree # List available resources
```

Common problems with launching the tasks

- Script fails because of wrong PATH or missing file – ensure all needed files are transferred and applications receive correct paths
  - Do not use absolute paths (starting with `/`) – only relative
- Not all required applications are correctly loaded
  - Check wiki and load all needed applications
  - Names of binaries are commonly little bit different – contain names of versions etc
- Estimation of time needed to run the task
  - No really good solution...
  - Make some trials and try to estimate...
  - There are very different CPUs available (with different speeds) – it is possible to require particular CPU type (but it reduces number of available nodes...)

Get to task's working directory

- Go to https://metavo.metacentrum.cz/pbsmon2/person and click to list of your tasks and click to selected task
- Search for information **exec_host** (address of node doing the task) and **SCRATCHDIR** (temporal directory for all data and results)
- Sometimes one needs to monitor task progress or influence it
- It is not possible to directly modify running task, but at least check (and possibly modify) input data and see outputs

```
# From MetaCentrum frontend login to respective node running the task
ssh exec_host # No need to specify user name
# Go to SCRATCH directory
cd SCRATCHDIR
# There are working data of currently running task...
```

Interactive tasks

- See information at https://wiki.metacentrum.cz/wiki/Interactive_job

```
# Again launch qsub according to actual needs
# Note "-I" for interactive session and missing script name
qsub -I -l walltime=2h -l nodes=1:ppn=1 -l mem=2gb
# Wait for job to start..
# After we get the interactive task, we are on new server
hostname # See where we are - we can connect to that server directly
ssh USER@given.server.cz # User name and password are the same
                         # Server address is output from "hostname"
screen # Secure we can log off in the meantime
```

- Work as on normal Linux server...
- With `screen` we can disconnect as usually and let tasks run in background

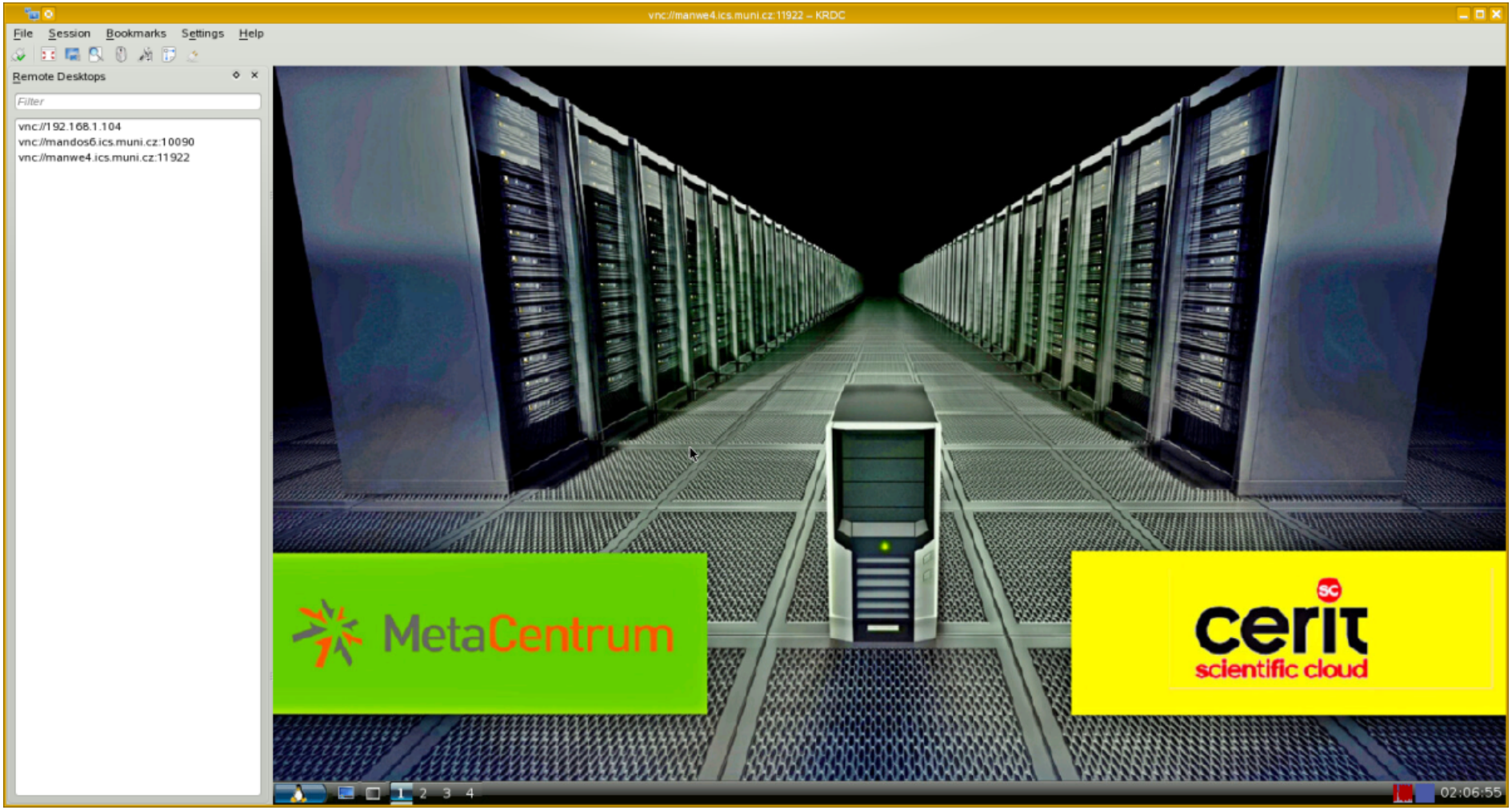
Graphical interactive task

- See information at https://wiki.metacentrum.cz/wiki/Remote_desktop

```
# Again launch qsub according to actual needs
# Note "-I" for interactive session and missing script name
qsub -I -l walltime=2h -l nodes=1:ppn=1 -l mem=2gb
# Wait for job to start..
# After we get the interactive task, we are on new server
screen # Secure we can log off in the meantime
module add gui # We need to add GUI module
gui start # Start GUI (see above link for details)
gui info -p # Print information about running VNC sessions
            # Including address, port and password to connect
```

- Launch your favorite VNC client (KRDC, TightVNC, ...) and use credentials from above output to connect
- Work as on normal Linux desktop...
- With `screen` we can disconnect as usually

Running VNC


vnc://manwe4.ics.muni.cz:11922 – KRDC
File Session Bookmarks Settings Help
Remote Desktops
Filter
vnc://192.168.1.104
vnc://mandos6.ics.muni.cz:10090
vnc://manwe4.ics.muni.cz:11922
MetaCentrum
cerit
scientific cloud
02:06:55

Creation and control of FS I

- `fdisk -l` lists disks and partitions
- To manage disk partitioning use `fdisk /dev/sdX` (doesn't support GPT very well yet) or `gdisk /dev/sdX`
- When hard drive is partitioned, partitions must be formatted
- Commands `mkfs.*` create various FS, common syntax is `mkfs.XXX -parameters /dev/sdXY`, where sdXY is particular disk partition
- Parameters can set label and various settings of behavior of the disk partition, check `man mkfs.XXX`
- To check FS for errors use `fsck.XXX /dev/sdXY` (according to respective FS)
- `tune2fs -parameters /dev/sdXY` can set various parameters to influence behavior of disk partition

Creation and control of FS II

- `hdparm -parameters /dev/sdX` can set advanced hardware parameters of hard drive
- The most convenient is using graphical tools available in all distributions...
  - in openSUSE there is YaST administrative module – from command line launch `yast --qt disk` for graphical or `yast disk` for text-based version
  - All distributions have graphical tools like GParted where it is possible to comfortable manage disks
- `df -h` shows available/occupied space on disks/partitions, but because of special features of Btrfs it doesn't show every time correct values for this FS – it is better to use `btrfs fi df /mount/point` (`/mount/point` use to be the most commonly `/`)

Managing system services

- Different among distributions – several main methods
- Most common is SystemD, less common older init scripts and RC scripts
- Used to manage services like web server, database, ...
- Read documentation for your distribution!
- All actions require root authentication

```
# SystemD - huge amount of possibilities
systemctl enable/disable/status/start/stop servicename # TAB helps
# RC scripts
rcservicename status/start/stop # TAB helps to select service
# Init scripts
/etc/init.d/servicename status/start/stop # TAB helps to select service
```

SystemD usage

```
# List installed services and their status
systemctl list-units --type service
# Enable/disable/see status/start/stop service
systemctl enable/disable/status/start/stop servicename # TAB helps
# Show overriden config files
systemd-delta
# Analyze boot time (how long does each service take to start)
systemd-analyze blame # Text output
systemd-analyze plot > filename.svg # Same in graphics
# Log for particular service
journalctl -u servicename
# Last logged messages (press Ctrl+C to exit)
journalctl -f
# Log records since last boot
journalctl -b
# Time and date information and management
timedatectl
```

The end

Our course is over...

...I hope it was helpful for You...

...any feedback is welcomed...

...happy Linux hacking...

...any final questions?

Typesetting using XeLaTeX on openSUSE GNU/Linux January 28, 2016