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, Prague
Institute of Botany, Czech Academy of Sciences, Průhonice
https://trapa.cz/, zeisek@natur.cuni.cz

January 26 and 27, 2017

Outline I

Outline II

Outline III

Outline IV

The course information

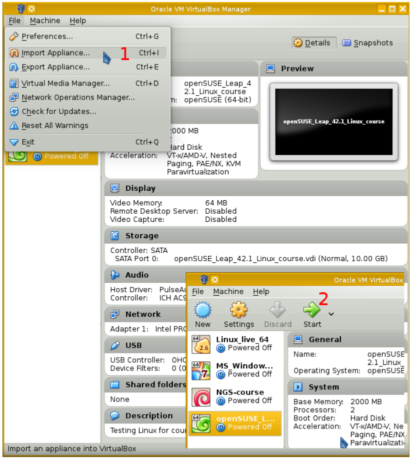
- The course page: `https://trapa.cz/en/course-linux-command-line-2017`
  - Česky: `https://trapa.cz/cs/kurz-prikazove-radky-linuxu-2017`
- Subject in SIS: `https://is.cuni.cz/studium/eng/predmety/index.php?do=predmet&kod=MB120C17`
  - Česky: `https://is.cuni.cz/studium/predmety/index.php?do=predmet&kod=MB120C17`
  - 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-linux-command-line-bash-scripting-metacentrum` – feel free to contribute, request new parts or report bugs

Materials to help you…

- Download the presentation from https://soubory.trapa.cz/ /linuxcourse/linux_bash_metacentrum_course.pdf
- Download the scripts and toy data from https://soubory.trapa.cz/linuxcourse/scripts.zip
    - Note: Open the scripts in some good text editor (slide 56) – showing syntax highlight, line numbers, etc. (NO Windows notepad); the files are in UTF-8 encoding and with UNIX end of lines (so that too silly programs like Windows notepad won't be able to open them correctly)
    - Never ever open any script file in software like MS Word – they destroy quotation marks and other things by "typographical enhancements" making the script unusable

Virtual machine for learning

- If you do not have Linux installed, download and install VirtualBox from https://www.virtualbox.org/
- Download openSUSE Leap 42.2 Linux distribution for this course from ftp://botany.natur.cuni.cz/openSUSE_Leap_Linux_course.ova (3.7 GB)
- Launch VirtualBox and go to menu **File** | **Import appliance...** to import it. When do, launch it (**Start**)



openSUSE_Leap_4 2.1_Linux_course
openSUSE (64-bit)
2000 MB
Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging, PAE/NX, KVM Paravirtualization

Video Memory: 64 MB
Remote Desktop Server: Disabled
Video Capture: Disabled

Controller: SATA
SATA Port 0: openSUSE_Leap_42.1_Linux_course.vdi (Normal, 10.00 GB)

Host Driver: PulseAu
Controller: ICH AC9

Adapter 1: Intel PRO

USB Controller: OHC
Device Filters: 0 (0

None

Testing Linux for cou

Linux_live_64 Powered Off
MS_Window... Powered Off
NGS-course Powered Off
openSUSE_L... Powered Off

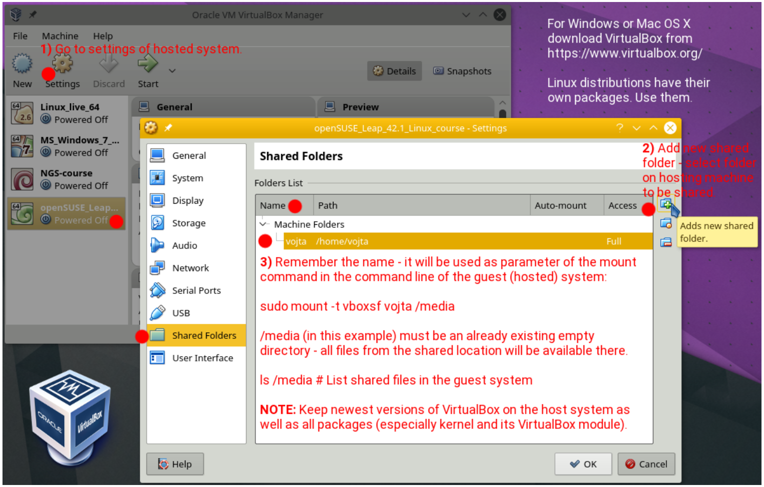
Name: openSUSE 2.1_Linux_
Operating System: openSUSE

Base Memory: 2000 MB
Processors: 2
Boot Order: Hard Disk
Acceleration: VT-x/AMD-V, Ne Paging, PAE/NX Paravirtualizati

Import an appliance into VirtualBox

VirtualBox shared folder I

VirtualBox can be configured to share folder with host operating system



For Windows or Mac OS X download VirtualBox from https://www.virtualbox.org/

Linux distributions have their own packages. Use them.

1) Go to settings of hosted system.

2) Add new shared folder - select folder on hosting machine to be shared.

Shared Folders

Name	Path	Auto-mount	Access
Machine Folders			
vojta	/home/vojta		Full

3) Remember the name - it will be used as parameter of the mount command in the command line of the guest (hosted) system:

```
sudo mount -t vboxsf vojta /media
```

/media (in this example) must be an already existing empty directory - all files from the shared location will be available there.

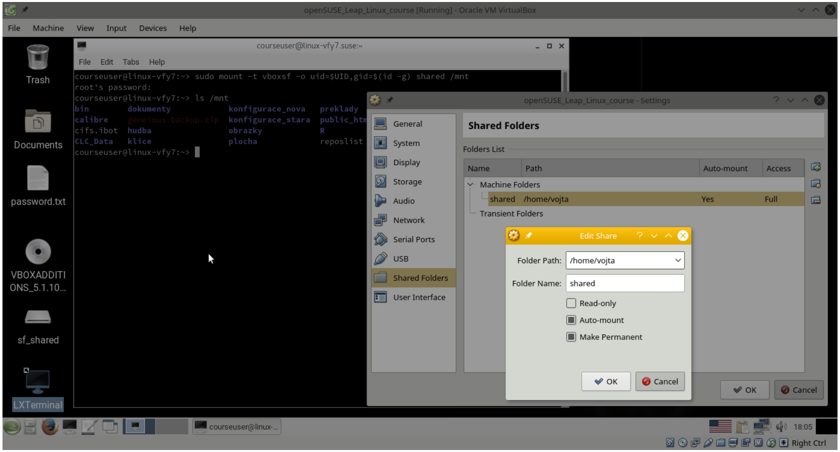
```
ls /media # List shared files in the guest system
```

NOTE: Keep newest versions of VirtualBox on the host system as well as all packages (especially kernel and its VirtualBox module).

VirtualBox shared folder II

Go to menu “Devices | Shared Folders” and set pair of folders

```
sudo mount -t vboxsf -o uid=$UID,gid=$(id -g) shared /mnt
```


openSUSE_Leap_Linux_course [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Trash
Documents
password.txt
VBOXADDITIONS_5.1.10...
sf_shared
LXTerminal
courseuser@linux-vfy7.suse:~
File Edit Tabs Help
courseuser@linux-...
18:05
Right Ctrl

```
courseuser@linux-vfy7:~> sudo mount -t vboxsf -o uid=$UID,gid=$(id -g) shared /mnt
root's password:
courseuser@linux-vfy7:~> ls /mnt
bin        dokumenty            konfigurace_nova   preklady
calibre    geneious.backup.zip  konfigurace_stara  public_htm
cifs.ibot  hudba                obrazky            R
CLC_Data   klice                plocha             reposlist
courseuser@linux-vfy7:~>
```

Name	Path	Auto-mount	Access
Machine Folders			
shared	/home/vojta	Yes	Full
Transient Folders			

Edit Share

Folder Path: /home/vojta
Folder Name: shared
☐ Read-only
▨ Auto-mount
▨ Make Permanent

OK Cancel

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, etc. (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-like (UN*X, *nix)
    - Systems compatible with UNIX (Linux, BSD and its variants, Mac OS X, ...)

What it is UNIX, Linux and GNU II

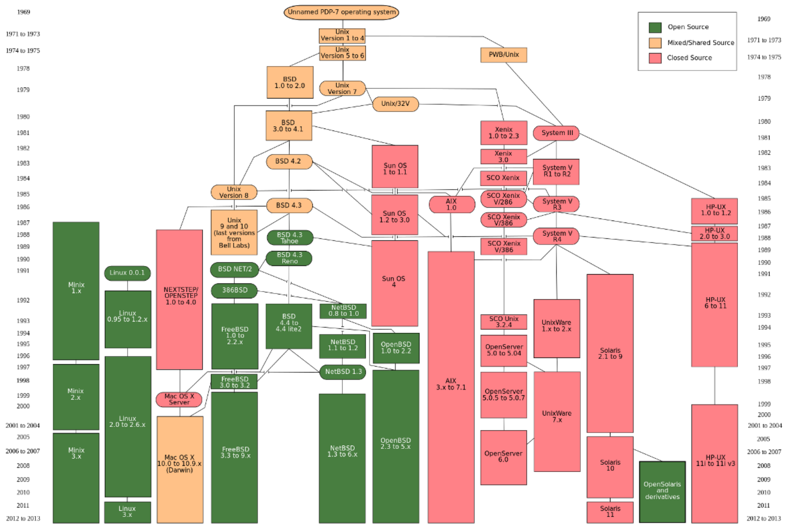
    - Mainly open-source (UNIX is commonly commercial – source code is not available for public, but its specification is)
    - Nowadays prevailing over “old” UNIX systems, used in many devices from tiny embedded toys to huge data centers
    - Try to provide same quality as paid systems, but (mostly) for free
  - Many courts about copyrights, parts of code, patents – USA allow software patents, EU not – GNU, Linux, BSD, etc. try to ensure to have only code not covered by any patents to avoid possible courts
- GNU
  - “GNU’s Not Unix!” – but it is compatible, respects its principles
  - 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

What it is UNIX, Linux and GNU III

  - Source code is free – anyone can study it (Security!), report bugs, contribute, modify, share it, ...
  - 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 – (F)OSS 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 of exact conditions
- 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 – (F)OSS II

- Apache license, Mozilla license, etc. – 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...

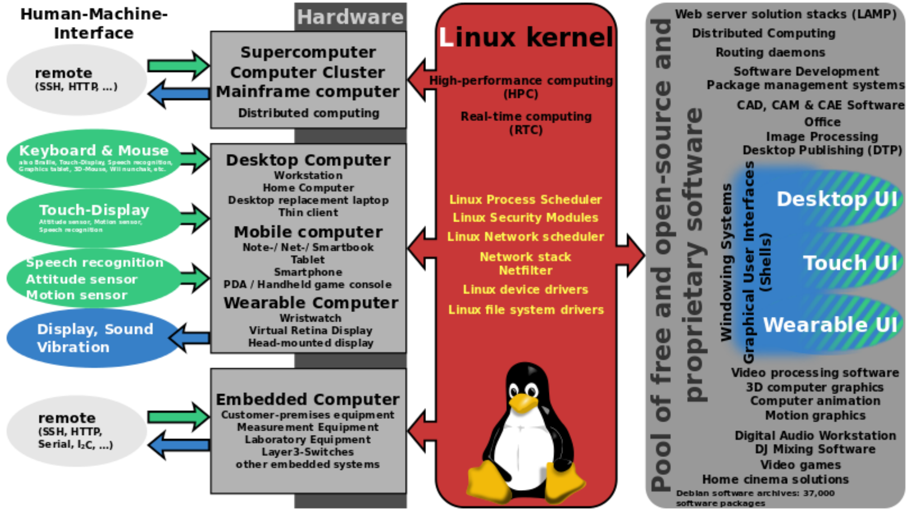
How to make money with free open-source software?

- 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 (“permission to use it”)
- Software as service
  - (F)OSS is available for free – user can use it as it is or buy a support – help of any type
  - No vendor lock-in – user has the code, so he can modify the software himself, change provider of the services, …
  - 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) asked a 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”

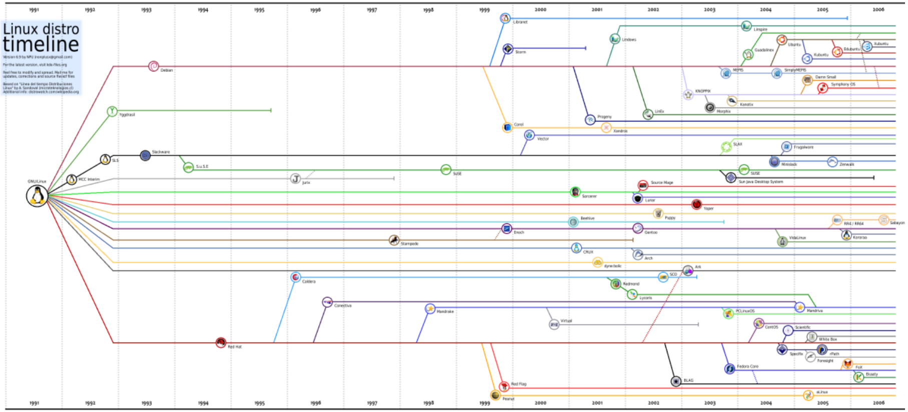
- Operating system respecting principles of UNIX
- Components
    - Linux kernel – basic part of the system responsible for hardware and very basic low-level running of the system (“Linux *sensu stricto*”)
    - GNU core utilities – basic applications
    - Graphical user environment (GUI) – many choices
    - Many other applications – according to use – whatever imaginable
- Linux distribution (“Linux *senso lato*”)
    - 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 (using e.g. web service)
    - 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/ (∼800 distributions, ∼260 active)

Most common Linux distributions

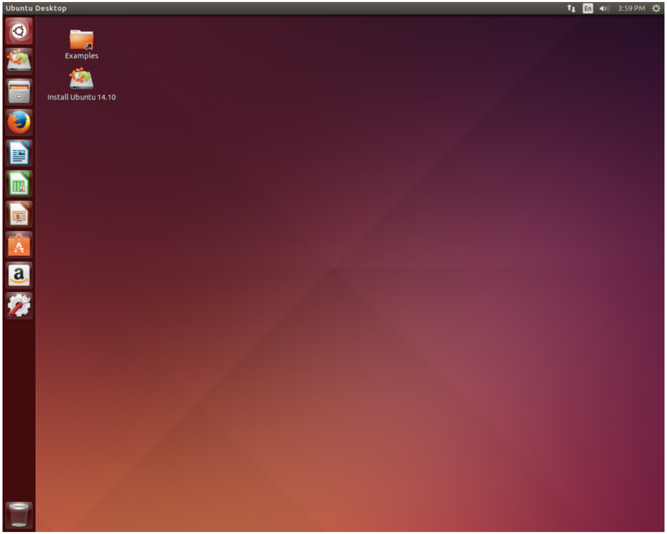
- Debian (DEB) based
  - Debian – one of oldest and most common, especially on servers
  - Ubuntu (nowadays probably the most popular on PCs and notebooks) and derivatives – Kubuntu, Xubuntu, Lubuntu, … (according to GUI used – most of the system is same)
  - Mint – Based on Ubuntu as well as Debian, very user-friendly
  - Kali, KNOPPIX, elementaryOS, …
- 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) companion of SUSE Linux Enterprise
  - Scientific Linux, Mageia, PCLinuxOS, …
- Android
- For experienced users: Arch, Slackware, Gentoo, …

Graphical User Interfaces (GUI)

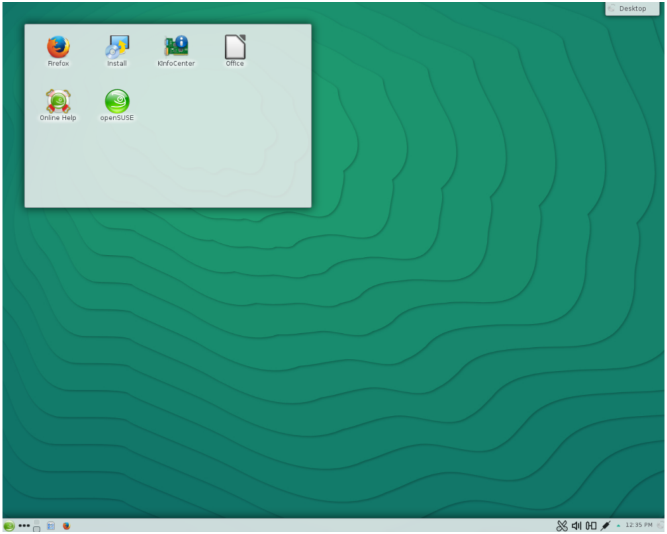
More like "Mac-style", "Windows-style" or something else? Feature rich or minimalistic?

- 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...

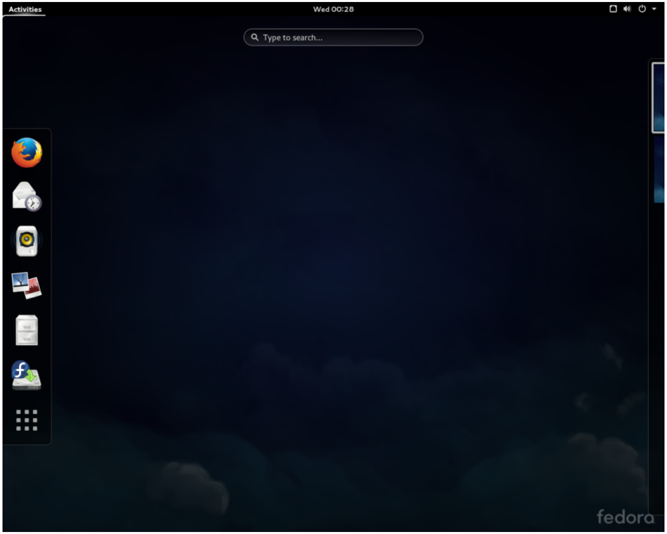
Ubuntu with Unity



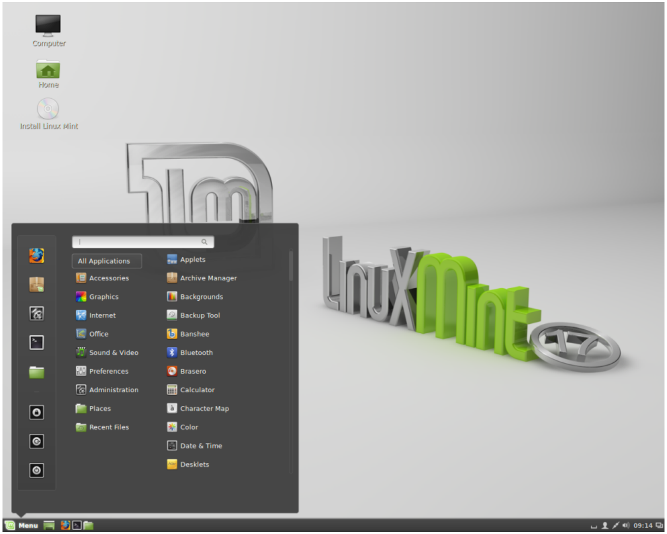
openSUSE with KDE – Kubuntu is same, but blue...



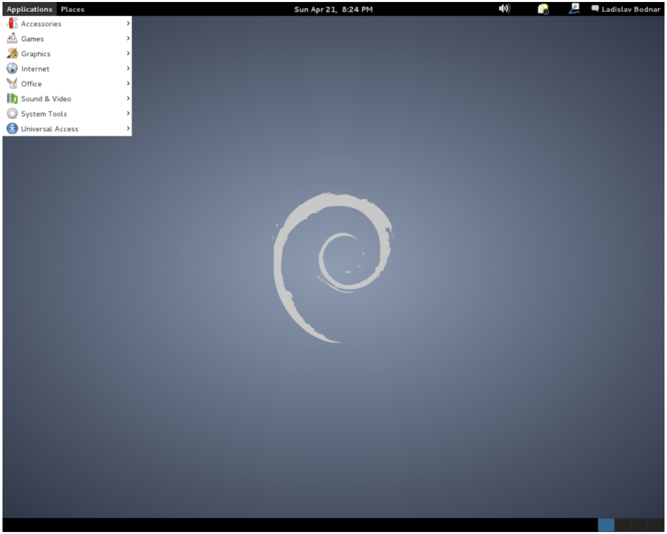
Fedora with GNOME – GNOME is always almost same



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 (slide 8)
  - 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? To have someone to ask for help...
- You can change GUI (or its design) without change of distribution (it use to be highly configurable)
- 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 servers, Parted Magic and/or SystemRescueCD to repair broken system (disk failure) and save data, ...

Differences among (common) Linux distributions

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

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
sudo somecommand # Launch somecommand with root's privileges
                 # Can be restricted for particular commands
```

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 – responsible for computer to start – in 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 (e.g. GParted)
- Always have backup before such management!

Put together more disks

Extend space and get higher data security

- 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

Creation and control of FS I

- All commands require root privileges
- `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)
  - The filesystem must be unmounted when checking it
  - XFS uses `xfs_repair /dev/sdXY`

Creation and control of FS II

- Btrfs uses `btrfs check /dev/sdXY`, if it is unmountable, `btrfs-zero-log /dev/XY` use to help, last instance is `btrfs check --repair /dev/sdXY` (dangerous operation)
- If Btrfs is mountable, but there are various FS errors and/or performance issues, `btrfs scrub start -Bdf /mount/point`, `btrfs filesystem defragment -r -v /mount/point` and `btrfs balance start -v /mount/point` – manual running can take long time and strongly slow down the computer

- `tune2fs -parameters /dev/sdXY` can set various parameters to influence behavior of disk partition
- `hdparm -parameters /dev/sdX` can set advanced hardware parameters of hard drive
- The most convenient is using graphical tools available in all distributions...

Creation and control of FS III

  - 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 filesystem df /mount/point` (`/mount/point` use to be the most commonly `/`)
- On UNIX FS, defragmentation and another maintenance tasks use to be done in background when computer is idling – unless there is at least ∼20% of free space on the device, this is not any problem and there are no performance issues

Another manipulations and information

- dd is powerful, but potentially dangerous tool used to backup or write disks or partitions (commonly to create bootable USB media)
- If writing disk image to the disk (sdX), disk's partition table is discarded and the disk is covered by whatever is the ISO image

```
dmesg # Recent entries in main system log - filter with grep, tail, ...
dmesg | grep sd | tail # Get information about recently plugged media
# 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/sdX # Used to write e.g. image of Linux live
                            # media to USB flash disk (Check sdXY!)
lnav # Comfortably browse recent logs, quit by "q"
```

- If there are encrypted partitions, they can be accessed /dev/mapper/
- If LVM (slide 34) is used, see lvscan and pvscan to find correct location in /dev/
- Disks are also accessible through /dev/disk/by-*

Mounting and unmounting disks and 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

```
eject # Open CD/DVD drive
mount # Which FS (disk partitions) are mounted
findmnt # See mounted devices in tree-like structure
mkdir /mnt/point # Directory must exist prior mounting into it
# mount usually recognize FS of mounted device, if not, us -t FS_type
mount /dev/sdXY /mount/directory # Mount disk sdXY to /mount/directory
umount /dev/sdXY # Unmount disk sdXY
umount /mount/directory # Unmount disk from /mount/directory
# Mount CD/DVD ISO image file into directory /mnt/iso
mount -t iso9660 -o loop file.iso /mnt/iso # See CD/DVD image content
```

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 * (yes, asterisk)
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 lists only "visible" files/directories)
ls -a # We will see it ("-a" to see all - also hidden - files/dirs)
```

Directory structure in Linux I

- Similar is also 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 – representations of disks, CD, RAM, USB devices, …
- `/etc` – system configuration in plain text files – edit them to change system-wide settings (read documentation and comments there)
- `/home` – users' homes
- `/lib`, `/lib64` – basic system libraries (32 and 64bits)
- `/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 FS mounted during startup
- `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 47
- Hard link – just another name for existing file, no special symbol, slide 47
- 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 78
- Socket – for communication among processes, also bidirectional, available on network

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
```

Owner and group

- Every file has an 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 (if not, `root` most change group ownership)
- Information about users and groups and their IDs are in `/etc/group` and `/etc/passwd`

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

File and directory 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 52)
- `764` – same as above — numbers for each role are summed – first one is for owner, second for group and last for others
- Executable scripts and binaries **require** executable permission (`x`)

Permissions examples

```
ls -l # Shows permissions, links, owner, group, size, date, name
# 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 (no recursive)
```

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 owning 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, `setfacl` sets
- 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
- Important especially in network environment with many users
- If intensively used, `ls -l` sometimes doesn't show correct permissions

```
getfacl FILE # get ACL for FILE
setfacl -m u/g:USER/GROUP:r/w/x FILE # Add for USER/GROUP r/w/x right
setfacl -R ... # Recursive (including subdirectories)
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 file `~/.bashrc`
  - `~` means user's home directory
  - `.bashrc` is user's configuration for BASH
- 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, ...
- `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
```

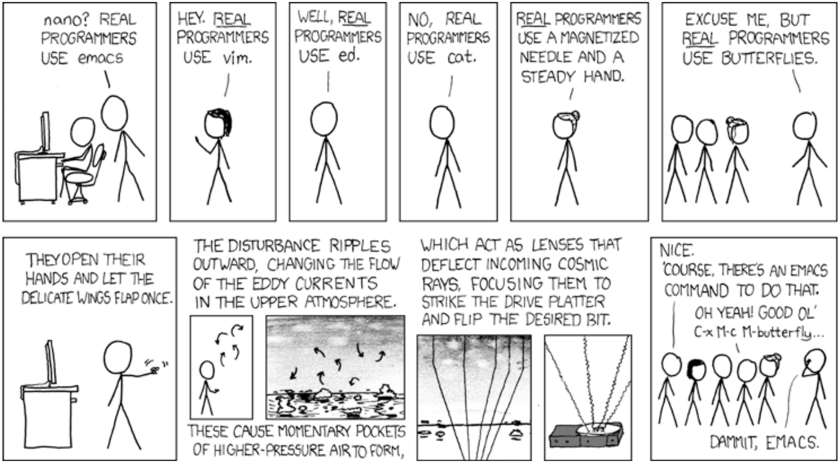
- `setgid` – application can have root permission even it was launched by normal user

```
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
- Mostly, there is no need to modify them

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

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


nano? REAL PROGRAMMERS USE emacs
HEY. REAL PROGRAMMERS USE vim.
WELL, REAL PROGRAMMERS USE ed.
NO, REAL PROGRAMMERS USE cat.
REAL PROGRAMMERS USE A MAGNETIZED NEEDLE AND A STEADY HAND.
EXCUSE ME, BUT REAL PROGRAMMERS USE BUTTERFLIES.
THEY OPEN THEIR HANDS AND LET THE DELICATE WINGS FLAP ONCE.
THE DISTURBANCE RIPPLES OUTWARD, CHANGING THE FLOW OF THE EDDY CURRENTS IN THE UPPER ATMOSPHERE.
THESE CAUSE MOMENTARY POCKETS OF HIGHER-PRESSURE AIR TO FORM,
WHICH ACT AS LENSES THAT DEFLECT INCOMING COSMIC RAYS, FOCUSING THEM TO STRIKE THE DRIVE PLATTER AND FLIP THE DESIRED BIT.
NICE.
'COURSE, THERE'S AN EMACS COMMAND TO DO THAT.
OH YEAH! GOOD OL' C-x M-c M-butterfly...
DAMMIT, EMACS.

https://xkcd.com/378/

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
- Check spelling
- Debug source code

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

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`)
  - Mac v. < 9: CR (`\r`) (Mac up to 9 wasn't UN*X, since OS X it 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

```
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
# Converts encoding of input file (ISO-8859-2) to outfile in UTF-8
iconv -f ISO-8859-2 -t UTF-8 infile.txt > outfile.txt
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
```

- Mac OS X mostly uses same encoding and EOL as Linux (and rest of UNIX world), so there are no problems with compatibility
- Launching of bash script written on Windows on Linux/Mac OS X will probably fail (because of different EOL)

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
- Names 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

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

Reading variables from command line and as output of another commands

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

```
# Reading variable from user's input from command line
# (some interactive script interacting with the user)
read X # We will read new variable from input (do not use "$" here)
10 # Type any value and press Enter
echo $X # Get value of the variable
10 # It works
# Following two commands are same and lead to same result
X=`command` # Set as variable output of command
X=$(command) # Set as variable output of command
echo $X # X will contain output of command
# "command" from previous lines can be e.g.
cat somefile.txt # Read content of somefile.txt
expr 1 + $X # Sum of 1 and variable $X
WORKDIR=`pwd`# Save current directory into variable
ls -1 | head -n 1 # First file/directory in the current directory
unset X # Destroy this variable
```

Variables and quotes and more

```
A=abcdef # Set new variable (no special characters allowed)
echo $A # See variable's content
abcdef # It works
echo '$A' # Single quotes preserve literal value
$A # We see variable's name, not its content
echo "$A" # Double quotes preserve literal value, except $, `, \
abcdef # This also works
echo `$A` # Text between back ticks is evaluated and launched
abcdef: command not found # There is no command "abcdef"...
echo "Hi, dear $USER" # Compare this and following command...
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, …)
- If variable is going to contain any special character (`?`, `.`, `*`, …), the value must be quoted – `"…"` allow escaping of special character or inclusion of another variables, `'…'` keeps absolutely literal value

How quotes influence reading of variable content

```
A=abcde # OK
echo $A # abcde
B=abcd$e # The content will be "abcde + $e" or "abcd" (if $e is missing)
echo $B # abcd
C=abcd\$e # \ escapes next character - it is loosing its special meaning
echo $C # abcd$e
D='abcd$e' # '...' keep literal value of the content
echo $D # abcd$e
# Next command breaks shell - incomplete quotes " - pres then Ctrl+C
E=ab"cde # The variable should contain incomplete quotes ", it fails
echo $E # Nothing - empty
F=ab\"cde # \ escapes next character - it is loosing its special meaning
echo $F # ab"cde
G='ab"cde' # '...' keep literal value of the content
echo $G # ab"cde
H=abc`echo $USER`de # See $USER to see what will be inserted into `...`
echo $H # abcvojtade # To add output of command into the variable
I='...' # Needed if $I should contain spaces, quotes, `, $, ...
```

Launching commands and scripts

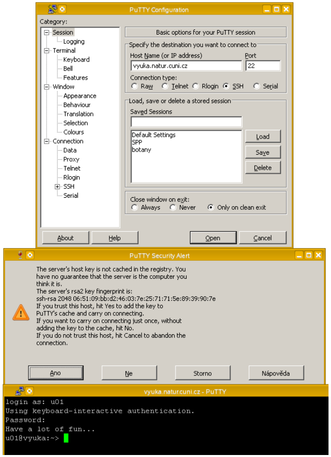
- Parameters of commands are separated by space and preceded by one or two minus(es)
- Parameter `-h` or `--help` usually gives help for particular command
- Getting help with `man` command
  - `man somecommand`
  - Arrows to list up and down, `q` to quit
  - Type `/` and type text and hit Enter to search – 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`)

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`
- `ssh uXY@vyuka.natur.cuni.cz`
- If fingerprint key changes, ssh complains a lot – possible man in the middle attack
- From Windows use Putty



Host Name (or IP address): vyuka.natur.cuni.cz
Port: 22
Connection type: Raw Telnet Rlogin [✓ SSH] Serial
Saved Sessions: Default Settings, SPP, botany
Close window on exit: Always Never [✓ Only on clean exit]

PuTTY Security Alert

The server's host key is not cached in the registry. You have no guarantee that the server is the computer you think it is.
The server's rsa2 key fingerprint is:
ssh-rsa 2048 06:51:09:bb:d2:46:03:7e:25:71:71:5e:89:39:90:7e
If you trust this host, hit Yes to add the key to PuTTY's cache and carry on connecting.
If you want to carry on connecting just once, without adding the key to the cache, hit No.
If you do not trust this host, hit Cancel to abandon the connection.

```
login as: u01
Using keyboard-interactive authentication.
Password:
Have a lot of fun...
u01@vyuka:~>
```

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`)

The shell

- Many names, many ways how to get it, still the same thing
- Fish – friendly interactive shell – 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, XFCE Terminal, LxTerminal, ...
  - Commonly allow appearance customization – font, colors, background, style of notifications, ...
  - Launch as many copies as you need (usually 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 some 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

Places to store BASH settings

- `/etc/bash.bashrc` – System wide BASH settings – can be overridden by user's configuration
- `~/.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/profile` – System wide profile file – can be overridden by user's configuration
- `~/.profile` – Settings loaded when user logs-in (mainly for language settings), sometimes used by remote connections
- Note: BASH scripts are non-interactive shells – they do not read settings above – there are no aliases, ... but they inherit some settings (PATH, language, ...) and they can read global variables

BASH settings

Write them into BASH configuration file

- In any text editor open `~/.bashrc` and edit it
- Behavior of BASH can be set to fit user's needs
- Terminal emulators allow to set custom fonts and colors, ...

```
# More colors for outputs
eval "`dircolors -b`"
# Ignore repeated entries in bash history (stored in ~/.bash_history)
HISTCONTROL='ignoreboth'
# Maximal length (number of lines) of bash history (~/.bash_history)
HISTFILESIZE='100000'
# Following two settings save history from multiple terminals
# Normally, only history from last time opened terminal is kept
shopt -s histappend # Append to history, don't overwrite it
# Save and reload the history after each command finishes
export PROMPT_COMMAND="history -a; history -c; history -r;
  $PROMPT_COMMAND" # Note recursive behaviour of $PROMPT_COMMAND
```

Aliases and BASH settings

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 write just "ll"
# To make the change above permanent, write it into ~/.profile or
# ~/.bash_profile or ~/.bashrc and reload the configuration
source ~/.bashrc # to reload BASH settings
                 # or "source" the file you modified
# If there are many aliases, they can be stored e.g. in ~/.alias
test -s ~/.alias && . ~/.alias || true # Check for extra alias file
# 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)
alias grep='grep --colour=always' # Enable color in grep
# Always human readable output of df (disk free)
alias df='df -h'
# Add aliases pointing to software installed outside PATH, ...
```

BASH globbing and wildcards

- BASH itself doesn't recognize regular expressions – it's wildcards have some of functions of regular expressions (from slide 121) and can look similarly, but behave differently! Do not confuse!
- `?` – Replaces any single character
- `*` – Replaces any number of any characters (`ls a*` lists all files starting with "a")
- `[]` – 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

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

Expressions

```
# Many operands have special meaning in BASH - must be escaped
echo `expr 1 '<' 2` # Is 1 smaller than 2? TRUE (1)
echo `expr 1 '>' 2` # Is 2 smaller than 1? FALSE (0)
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 5 characters starting at position 10 of the text
echo `expr substr "MetaCentrum and Linux" 10 5`
# Does 1st chain contain 2nd chain (how long)? 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`)

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; }`
- `|` – pipe – redirects standard output of one command into standard input of second command: compare `mount` and `mount | column -t`
- Behavior in shells other than bash might be little bit different

Redirects and pipes

- `/dev/null` – “black hole”
  - Can discard anything
  - Discard only errors (note“2”): `command 2> /dev/null`
- `/dev/stdin` – standard input
  - Typically keyboard
  - In case application reads files, not from standard input: `echo "Žluťoučký k úpěl" | iconv -f utf-8 -t cp1250 /dev/stdin`
- `/dev/stdout` – standard output
  - Typically screen, commonly redirected into file
  - We wish to see errors which would be discarded otherwise: `command 2> /dev/stdout`
- `/dev/stderr` – standard error output
  - Typically screen or log file
  - Right place to send errors to: `echo "error" > /dev/stderr`
- `|` – pipe – basic redirecting method – standard output of one command to standard input of another command

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, …)

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

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

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

Redirects of standard input and output

```
# Write directory listing into text file
# If file directory_listing.txt exists, will be overwritten
ls -la > directory_listing.txt
cat directory_listing.txt # See result (same as running "ls -la")
# If file directory_listing.txt exists, new content will be appended
ls -la >> directory_listing.txt
cat directory_listing.txt # See result
# Add error output to the end of standard output file
# Note: In the examples below command "commandX" does not exist -
# it produces error "command not found" to be recorded by the log
# and because of redirect, the error is not shown in the terminal.
command >> outputfile.log 2>&1 # Example:
{ commandX; ls; } >> outputfile 2>&1
cat outputfile # See result
# Add error output to the error log text file
command >> outputfile.log 2>error.log # Example:
{ commandX; ls; } >> outputfile.txt 2>error.log
cat outputfile.txt # See results
cat error.log # See results
```

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 # Information about mounted file systems
findmnt # Display mounted devices in tree structure
```

Processes – every running program has its own process ID

```
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 the 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

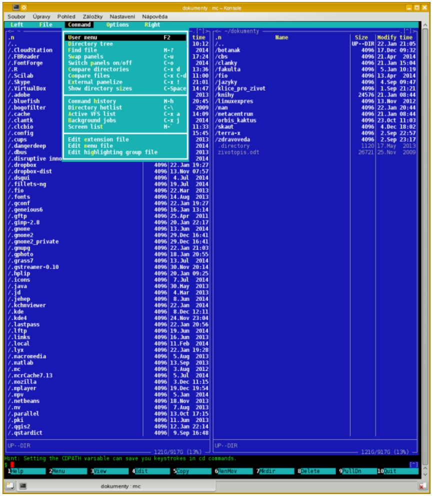
```
whoami # What is my user name
id # Information about current user (user ID and group IDs)
who # Who is logged in
w # Who is logged in, more information
users # Plain list of currently logged users
finger # Information about users on current terminals
last # Last logged-in users
passwd # Change password
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 :-)



```
  Left     File     Command     Options     Right
<- ~ -----------                          -.[^]> <- ~/dokumenty ------------------------------.[^]>
.n                 User menu             F2       time  .n               Name         Size  Modify time
/..                Directory tree                10:12  /..                        UP--DIR 22.Jan 21:05
/.CloudStation     Find file             M-?      2014  /botanak                      4096 17.Dec 09:32
/.FBReader         Swap panels           C-u     17:24  /cbs                          4096 21.Apr  2014
/.FontForge        Switch panels on/off  C-o      2014  /clanky                       4096 21.Jan 15:04
/.R                Compare directories   C-x d   13:36  /fakulta                      4096  5.Jan 10:19
/.Scilab           Compare files         C-x C-d 11:00  /fio                          4096 13.Apr  2014
/.Skype            External panelize     C-x !   21:01  /jazyky                       4096  4.Sep 09:47
/.VirtualBox       Show directory sizes  C-Space 14:47  /klice_pro_zivot              4096  1.Sep 21:21
/.adobe                                           2013  /knihy                       24576 21.Jan 08:44
/.bluefish         Command history       M-h     20:45  /linuxexpres                  4096 13.Nov  2012
/.bogofilter       Directory hotlist     C-\      2009  /man                          4096 22.Jan 20:44
/.cache            Active VFS list       C-x a   14:09  /metacentrum                  4096 21.Jan 08:44
/.clamtk           Background jobs       C-x j    2014  /orbis_kaktus                 4096 23.Oct 11:03
/.clcbio           Screen list           M-`     11:33  /skaut                        4096  4.Dec 18:02
/.config                                         15:45  /terra-x                      4096  2.Sep 22:57
/.cups             Edit extension file            2013  /zdravoveda                   4096  2.Sep 23:17
/.dangerdeep       Edit menu file                 2014   .directory                   1120 17.May  2013
/.dbus             Edit highlighting group file   2013   zivotopis.odt               26721 25.Nov  2009
/.disruptive inno                                 2014
/.dropbox                         4096 22.Jan 19:27
/.dropbox-dist                    4096 13.Nov 07:57
/.dsgui                           4096  4.Jul  2014
/.fillets-ng                      4096 19.Jul  2014
/.fio                             4096 22.Mar  2013
/.fonts                           4096 14.Aug  2013
/.gconf                           4096 22.Jan 19:27
/.geneious6                       4096 16.Jan 13:14
/.gftp                            4096 25.Apr  2011
/.gimp-2.8                        4096 20.Jan 22:17
/.gnome                           4096 13.Jun  2014
/.gnome2                          4096 29.Dec 16:41
/.gnome2_private                  4096 29.Dec 16:41
/.gnupg                           4096 22.Jan 21:03
/.gphoto                          4096 18.Jan 20:55
/.grass7                          4096 13.Jul  2014
/.gstreamer-0.10                  4096 30.Nov 20:14
/.hplip                           4096 20.Jan 09:25
/.icons                           4096  7.Jul  2014
/.java                            4096 30.May  2013
/.jd                              4096  4.Mar  2013
/.jehep                           4096  8.Jun  2014
/.kchmviewer                      4096 22.Jan  2014
/.kde                             4096  8.Dec 12:11
/.kde4                            4096 24.Nov 23:04
/.lastpass                        4096 22.Jan 20:56
/.lftp                            4096 19.Jun  2014
/.links                           4096 16.Jun  2013
/.local                           4096 11.Feb  2014
/.lyx                             4096 22.Jan 19:28
/.macromedia                      4096  5.Aug  2013
/.matlab                          4096 13.Sep  2013
/.mc                              4096  3.Aug  2012
/.mcrCache7.13                    4096  5.Jul  2014
/.mozilla                         4096  3.Dec 11:15
/.mplayer                         4096 19.Dec 19:54
/.mpv                             4096  5.Jan  2014
/.netbeans                        4096 18.Nov  2013
/.nv                              4096  7.Aug  2013
/.parallel                        4096 13.Oct 17:15
/.pki                             4096 11.Jun  2013
/.qgis2                           4096 12.Jan 22:14
/.qstardict                       4096  9.Sep 16:48
UP--DIR                                              UP--DIR
                           121G/917G (13%)                                      121G/917G (13%)
Hint: Setting the CDPATH variable can save you keystrokes in cd commands.
$                                                                                          [^]
 1Help   2Menu   3View   4Edit   5Copy   6RenMov   7Mkdir   8Delete   9PullDn  10Quit
```

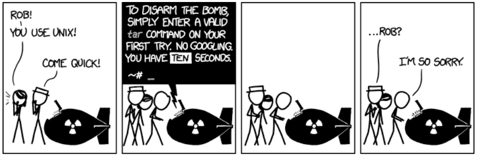
Compressing files into archives

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 can be 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


ROB!
YOU USE UNIX!
COME QUICK!
TO DISARM THE BOMB, SIMPLY ENTER A VALID tar COMMAND ON YOUR FIRST TRY. NO GOOGLING. YOU HAVE TEN SECONDS.
~# _
...ROB?
I'M SO SORRY.

https://xkcd.com/1168/

Looking for files and applications

```
apropos keyword # Searches for command descriptions containing keyword
updatedb # Must be regularly launched to get "locate" to work
         # It is usually regularly launched by cron task (see further)
locate somename # Searches for files/directories in local database
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."; exit 1; }
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."; exit 1; }
hash Application 2>/dev/null || { echo >&2 "Application is required
  but not installed. Aborting."; exit 1; }
exit 1 # Use to be added (with various numbers) after any error to
       # send term signal 1 - for better handling of various errors.
       # Every termination has exit status number - 0 is normal exit.
       # Exit status 1 and higher number is various error.
```

Find

```
find <where> <what> <what to do> # The most powerful searching tool:
find /.. -type d/f -name XXX -print # Most common usage
```

- First `find`'s parameter is location to search – absolute or relative, "`.`" means current directory (the only compulsory parameter)
- `-type` for only directories `d` or only files `f` (without this parameter, files as well as directories are looked for)
- `-name` of the searched files/directories supports wildcarts (`*`, `?` and `[…]`), see globbing (slide 74)
- `-print` is default action – prints list of results
- `-exec` runs some command with results (some operation, not just listing)
  - 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

Find examples

```
# Find in photos subdir all JPG files and resize them to 1000x1000 px
find photos/ -name *.jpg -exec mogrify -resize 1000x1000 '{}' \;
# Another possibility with xargs (it chains commands - reads input from
# stdin and execute command with given arguments, using all CPU threads)
# Note in the example below -print is not needed as it is default action
find photos/ -name *.jpg | xargs mogrify -resize 1000x1000
# Find all R scripts in ~/Documents and find in them lines with "DNA"
find ~/Documents -name *.r -print | grep -nH DNA # Two possibilities
find ~/Documents -name *.r -exec grep -nH DNA '{}' \;
# How many directories are there in the books directory
find books/ -type d -print | wc -l # wc -l calculate lines
# Find in /home/$USER/photos all JPG files containing string "trip"
find /home/$USER/photos -name *trip*.jpg -print
# Change permissions of all files within "files" directory to 777
find files/ -type f -exec chmod 777 '{}' \;
# Find all executable files within current directory and list them
find . -executable -type f -print
find doc/ -type d -empty -execdir rmdir {} \; # Delete empty directories
man find # See another options. Much more...
```

Network protocols I

- 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 20049)

Network protocols II

- webDAV – file transfer over web (using WWW server) – not so common, but good (port 80 or 443 – same as WWW)
    - Accompanied by calDav and cardDav to share calendars and address books over the network
- SAMBA – UNIX connection to Microsoft network shares (port 5445)
- web – "The Internet" for most of users (port 80 or encrypted 443)
- IMAP (port 143 or 993) and SMTP (port 25 or 465) to connect to e-mail server and send mails
- Messaging protocols like XMPP (Jabber and derived services like Google Talk or Facebook Messenger), IRC, ICQ, Skype, ...
- And much more...
- Port number can be changed in configuration of respective server service
- All firewalls on the way must allow communication on given port

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
# If using nmap at faculty, firewall disconnects you for 10 minutes!
nmap -r someserver.cz # Scan someserver.cz for opened ports
nmap botany.natur.cuni.cz --script ssh-hostkey # See SSH key
```

Connecting file systems on remote servers

- Target mount point must exist before mounting
- Servers can be accessed by IP address or hostname

```
# Mount Windows server (requires package samba)
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
```

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 connect to remote server (next slide)
- On server side, same files can be accessible (shared) by more methods

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`

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

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
```

Faculty web server

- Login requires same credentials as to CAS (login name, no ISIC number)
- Faculty information are only in Czech
- It is Linux server running Debian
- Connect with SSH/SFTP/SCP to `web.natur.cuni.cz`
- Mainly used for webhosting, user's address will have form `https://web.natur.cuni.cz/~loginname/`, user can apply for another URL
- Every department also has dedicated space there, it can be used for various web projects, address can be discussed with IT department
- Personal web can be placed in `public_html` within home folder
- Users can apply for MySQL database or special settings
- Department of Botany (and some other departments) have their own web and file servers

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 provided, `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 500x500 \
  -quality 75 '{}' '{.}-small.jpg' # or
parallel convert -resize 25% '{}' '{.}-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"'
# Convert all WAV files into OGG
parallel -X oggenc ::: *.wav # -X parse as many parameters as possible
```

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 -e "Start: `date`\n" > {}.log && mb {} |
  tee -a {}.log && echo "End: `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 typing, record.txt is used)
```

Launching of tasks at certain time

- `at` can run command at certain time (atd daemon must be running)
- Tasks are running in background, outputs are mailed (e.g. to `/var/spool/mail/$USER`)

```
# Check status of atd daemon (it must run), start/stop/enable/disable it
systemctl status/start/stop/enable/disable atd.service
man at # Check for various possibilities of time settings
at HH:MM # Run commands at certain time (hour:minutes)
at> command1 # Add as many commands as you wish (separate by Enter)
at> # When done, press Ctrl+D to cancel giving commands to at
# Instead of manual typing of tasks, run script at certain time
at HH:MM -f somescript.sh # Run somescript.sh at certain time
at -l # List of scheduled tasks (alias is atq)
at -r <number> # Cancel scheduled task (according to number from at -l)
atrm # Alias for previous command
batch # Commands will be executed when system loads drops below 0.8 or
      # other value specified in configuration or startup of atd
```

Automated launching of tasks

- `cron` runs tasks repeatedly (cron daemon must be running)
- Scripts for tasks running hourly/daily/weekly/monthly can be copied into respective `/etc/cron.*/` directories

```
# Check status of cron daemon (it must run), start/stop/enable/disable
systemctl status/start/stop/enable/disable cron.service
crontab -l # List user's cron tasks
crontab -e # Edit user's 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
  0       11    *             *      0        # Every Sunday, 12:00
  30      3     */2           *      *        # Every even day, 4:31
  */15    *     *             *      0,4   # At Sun and Fri every 15 min
# Columns are separated by any number of spaces
```

Everything is (text) file

- As UNIX configuration and outputs (logs, …) are mostly saved as relatively simple text files, manipulations of any type with text files is one of the most common tasks
  - Similar situation is for molecular data – input/output data use to be text files with simple structure
- One of the most powerful features of BASH
- Some operations are complicated (e.g. complex manipulations with columns, various calculations) it is necessary to use AWK or Perl (probably the most advanced language working with text)
- Text-manipulating tools have very rich implementation of regular expressions (slide 121)
- Most of the operations are done in stream – per line – everything is very fast

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 # Append 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 (also used by man)
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 (by lines)

```
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 121

Work with columns

- `cut` extracts columns, `paste` joins, `column` reformates
- BASH can not select column according to its name (Perl can do that)

```
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 -f 1,2 cut_awk_test_file.tsv # get columns 1 and 2 separated by TABs
# Add second file as second column
paste file1 file2 > outputfile
# Output will be two columns (from file1 and file2) separated by TAB
paste -d "|" diff_test_file_1.txt diff_test_file_2.txt # -d for delimiter
# Swapping columns is not very comfortable...
paste <(cut -f2 cut_awk_test_file.tsv) <(cut -f1 cut_awk_test_file.tsv)
ls -l | column -t # Reformate input as table (compare with ls -l)
```

Get a column with awk

- AWK is scripting language mainly for text manipulations
- Can not select column according to its name (Perl can do that)
- Can do things other BASH tools can not do (easily) – better manipulation with columns, calculations, …
- Has complicated syntax, it is hard to read, it is not similar to other tools – Perl can do more and is more common (learn it instead)…
- Supports regular expressions, slide 121

```
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)
```

More AWK examples

```
# Print on even lines ">", former column 1, new line, former column 2
awk '{print ">"$1"\n"$2}' awk_test_file.tab # 2 columns into 2 lines
# 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=1, 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 6 is numeric, select lines where field 6 is higher than 100
awk -F '\t' '$6>100' cut_awk_test_file.tsv # Separated by TABs (\t)
# Print fields 4 and 5 (fields are separated by "_" or TAB)
awk -F '[_\t]' '{print $4, $5}' awk_test_file.tab
# Precede each line by its line number for all files together, with TAB
awk '{print NR "\t" $0}' diff_test_file_*
# substitute "foo" with "bar" ONLY for lines which contain "baz"
awk '/baz/{gsub(/foo/, "bar")}; 1'
```

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")
LC_ALL=C sort ... # To force use of English locale use
sort -k 2 -n textfile # Sort according to 2nd field (numerically)
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
# Replace capital letters by small ones
tr "[A-Z]" "[a-z]" < inputtextfile > outputtextfile
# Alternative (easier reading) of previous command:
cat inputtextfile | tr "[:upper:]" "[:lower:]" > outputtextfile
# Replace all new lines (line breaks) by TABs
cat inputtextfile | tr "\n" "\t"  > outputtextfile
# Discard all new lines - output will be one line
tr -d "\n" < textfile > /dev/stdout # stdout is typically screen
tr --help # See another possibilities for pattern to find/replace
```

Replacements with sed

- `sed` supports regular expressions, see slide 121 (same as in `grep` and `vim`), with parameter `-r` can use extended regular expressions (do not confuse – the syntax is slightly different, richer)
- Output is written to standard output – use pipes, redirects or `-i` to modify the file in place (without printing of output)
- Mac OS X has old outdated versions of `grep`, `sed` and other tools (richness of regular expressions is poor) – use versions from Homebrew or search documentation how to modify the patterns…
- Option `-s` separates multiple files (otherwise lines in multiple files are calculated as one stream)
- Option `-n` use to be used when deleting lines or printing only specific lines to suppress other lines (see examples)
- Various parameters, modificators, operators can be combined…

```
sed 'operator/FindToReplace/Replace/modificator' textfile > newtextfile
```

Sed examples I

```
# 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/Replace/3' # 's/.../.../' replace only first occurrence
sed '1,7s/...' # To work only on particular line, place single number or
sed '5s/...'   # range (e.g. 1,7) right before "s" ("$" for last line)
sed '1~2n;s/F/R/g' # Work on every second line, starting by line 1
sed -n '2~10p' # Print every 10th line, starting with line 2
seq 1 100 | sed -n '2~10p' # Example of above pattern (see "seq 1 100")
# 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
# Convert all capital letters into lower
sed 's/[A-Z]/\L&/g' inputtextfile > outputtextfile # And vice versa:
sed 's/[a-z]/\U&/g' inputtextfile > outputtextfile
# Groups to remember work in same way in sed, grep as well as vim
\(ToRemember\) # Remember expression in brackets (example at next slide)
\Number # Use remembered expression (numbered from one: \1, \2, \3, ...)
```

Sed examples II

```
# Take output of ls -l and replace value of $USER by "$USER-RULEZZZ"
ls -l | sed "s/\($USER\)/\1-RULEZZZ/g" # Note " to use variable
# Replace size column (2nd numeric) by "size:TAB<file size>b"
# Second sed replaces any white spaces by single TAB
ls -l | sed 's/\([0-9]\+\)/size:\t\1b/2' | sed 's/[[:blank:]]\+/\t/g'
sed '/^$/d' # Delete blank lines ('8d' line 8, 's/ *$//' spaces on EOL)
sed '/GUI/d' # Delete all lines containing "GUI" ('s/GUI//g' only "GUI")
sed '5 i\Linux is great.' # Insert to 5th line ('4 a\' after 4th line)
# Insert text to the beginning of the 3rd line (compare with previous)
sed '3s/^/INSERT/' # ^ is beginning of line, $ end ('$/...' last line)
# From ls -l keep number of links (1st numeric column after permissions)
# and then flip user and group and print it as "group:user"
ls -l | sed 's/ \([[:digit:]]\+\) \([[:alnum:]]\+\) \([[:alnum:]]\+\) /
  \1 \3:\2 /g' # Note separating spaces (previous line ends with space)
# Escaping - replace dot by comma (dot means any single character)
sed 's/\./,/g' # \ escapes following character (compare with 's/./,/g')
# Replace any of characters within [...] by some pattern
sed 's/[abcd]/X/g' # Compare with reverse case 's/[^abcd]/X/g'
```

Joining

- Generally, most of tools work per-line, `paste` appends columns (slide 107)
- Join compares every matching lines (by default 1st field) and creates all combinations - ensure to have sorted input files with unique text
    - E.g. if 1st file contains `A B` and `A C` and 2nd file `A D` and `A E`, the result will be `A B D`, `A B E`, `A C D` and `A C E`

```
# Add file to the end of another text file
cat file1 >> file2 # file2 will contain both files, file1 is unchanged
# Compare two sorted text files and write shared lines
# (duplicitous lines are shown just once)
join textfile1 textfile2 > outputfile
# If used on wrong files, it can create huge file
seq -f "1 %g" 100 > aaa && less aaa
seq -f "1 %g" 100 > bbb && less bbb
join aaa bbb | wc -l
join --help # See more options...
```

Comparisons

```
cat diff_test_file_1.txt diff_test_file_2.txt # See and use examples
# 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)
```

Command line text 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 56)
- Emacs and Vim are extremely rich, but having completely different approach – when you get use to one, you can't use the another

```
nano textfile # Enhanced clone of pico, basic simple text editor
pico textfile # Basic simple text editor
mc # Use its internal editor, just very basic (press F4 on the file)
emacs textfile # Extremely feature rich (including file browser
               # and many tools), Exit by Ctrl+X and Ctrl+C
vim textfile # Probably the most common, as rich as Emacs (see further)
vimtutor # Launch tutorial to learn Vim (in various languages)
```

The editors and their usage

- In `nano` and `pico` see bottom line for commands
  - `Ctrl+O` to write the file, `Ctrl+X` to quit the editor, `Ctrl+G` for help (`^` stands for `Ctrl` key)
- In `mc` highlight the file to edit and press `F4`
  - `F2` to save, `F10` to quit, `F1` to help, `F9` for top menu (navigate with arrows, cancel with double `Esc`), it is possible to use mouse
- **Emacs** use huge number of commands with `Ctrl` key
  - See `https://www.gnu.org/software/emacs/tour/`, `http://tuhdo.github.io/emacs-tutor.html` and `http://www.jesshamrick.com/2012/09/10/absolute-beginners-guide-to-emacs/`
- The most common is **Vim** (next slide)
  - See `http://vim-adventures.com/` to play a game and learn Vim
- **Emacs** and **Vim** have huge number of possibilities and support for plugins and scripts, but completely different usage style – one person can really learn only one...

Vim

Vim has three different working modes

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, `/ToSearch` to search (`Enter`, `n` for next occurrence, quit with `Esc`), … Exit to command mode by `Backspace` key (delete “`:`”).

- See `http://vim.wikia.com/`, `http://www.vim.org/docs.php`; česky `http://www.nti.tul.cz/~satrapa/docs/vim/`

Regular expressions are useful...

WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.
OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!
BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!
IT'S HOPELESS!
EVERYBODY STAND BACK.
I KNOW REGULAR EXPRESSIONS.
PERL!
TAP TAP

- Find text according to a pattern
- Manipulate the text – flip, reformat, replace, ...
- Syntax is variable among programming languages and applications
- There are commonly more solutions for one task
- 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 boundaries
- `[[: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 `)` by backslash `\` – **egrep** (extended version, launched as `grep -E` or `egrep`), **sed** with extended reg exp (`sed -r`) and **perl** not
- Read http://www.regular-expressions.info/; česky 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, …
- See sed examples, slide 113
- 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 +x noninteractive.sh`)
- Launch with `./noninteractive.sh`
- The most simple script:

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

Special variables available in the script (selection)

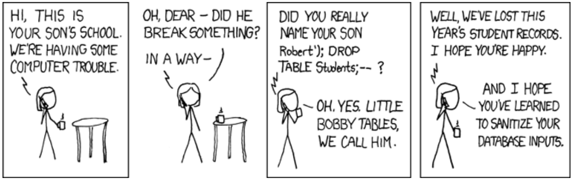
- `$1`, … (number from `1` up to number of parameters) – individual positional parameters (see next slide for example)
- `$0` – path of the starting script
- `$#` – number of command-line arguments
- `$*` – all of the positional parameters, seen as a single word, must be quoted
- `$@` – same as `$*`, but each parameter is a quoted string – the parameters are passed on intact, without interpretation or expansion, each parameter in the argument list is seen as a separate word, should be quoted (i.e. something like `"$@"`)
- `$$` – process ID (PID) of the script itself
- `$?` – Exit status of a command, function, or the script itself
- See more variables…

Functions in BASH

Pieces of code, which can be used repeatedly

```bash
# Declare new function
function MyNewFunction1 {
  echo "Hello, $USER on $HOSTNAME!"
  }
# Use it in a script as any other command
...
MyNewFunction1
...
# Use with variables - provide parameters for the function
# See following script examples for another input - same as in scripts
function MyNewFunction2 {
  echo "The sum is `expr $1 + $2`."
  }
# Use it in the script
...
MyNewFunction2 5 8 # For example
...
```

It is important to check user input...


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

https://xkcd.com/327/

- By accident or purpose (attack), user can enter unexpected value
  - In the "best" case, the script "just" crashes
  - Script can behave unexpectedly, returning very weird results
  - Internal functions/commands can return error messages, which are hard to understand
  - Attacker can e.g. modify web content (XSS, ...), obtain private data, root privileges, ...
- Programmer should always check if user input is correct, filter it

Script reading two variables

```bash
#!/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:

```bash
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
./interactive2.sh # Values will be provided when script asks
```

- There is no checking of input values, nothing advanced, ...
- See next slide to read the variable in `while` cycles to ensure it is correctly entered

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

Ensuring user interactively provides correct input

- Detailed explanations of all features used here are in various following slides... See scripts `interactive2{whiles,functions}.sh`

```
... # Following code replace lines 3 and 4 from previous script
NUMBER='^[0-9]+$'
echo "Please, provide a number as input value:"
while : # Start of while cycles - run until correct input is provided
  do # Star of the body of the cycles
  read INPUT # Here the input from keyboard is received
  if [[ $INPUT =~ $NUMBER ]]; then # Test if INPUT is a number
    echo "OK, input value is $INPUT."
    break # We have correct value, we can break the cyclus and continue
    else # What to do if the user did not provided correct value
      echo "Error! You provided wrong value!" # Tell the user
      echo "Try again (the number):" # Ask user for new input value
    fi # End of the conditional evaluation
  done # End of the while cycles
... # The code continues...
```

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*`, ...
- This is the most simple usage, more complex possibilities are ahead

Provide parameters, verify them and behave accordingly I

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

Multiple switches in classical UNIX form (no positional) I

- Following code use to be near beginning of the script to evaluate input
- See `interactive5.sh` for complete example
- `getopts` reads short (one-letter) parameters, they can have input value (marked by `:`)

```
... # All provided values are evaluated in while cycles...
while getopts "hvi:o:a:" INITARGS; do
  case "$INITARGS" in # $INITARGS contains the parameters to evaluate
    h|v) # Accept parameters "-h" or "-v" for help
      echo "Usage options..." # What will be done it this is selected...
      exit # Terminate after providing help
      ;; # End of this option
    i) # Parameter "-i" accepts some value (e.g. "-i inputfile.txt")
      ... # Do some checking etc...
      INPUTFILE="$OPTARG" # $OPTARG always contains value of parameter
      ;;
    # Continues on following slide...
```

Multiple switches in classical UNIX form (no positional) II

```
# ...starts on previous slide...
    o) # Parameter "-o" accepts some value (e.g. "-o outputfile.txt")
      ... # Do some checking etc...
      OUTPUTFILE="$OPTARG" # $OPTARG always contains value of parameter
      ;;
    a) # Parameter "-a" accepts some value (e.g. "-a X" for number)
      # Check if provided value makes sense (integer between 10 and 300)
      if [[ "$OPTARG" =~ ^[0-9]+$ ]] && [ "$OPTARG" -ge 10 -a
        "$OPTARG" -le 300 ]; then # The condition is long...
        VALUE="$OPTARG" # $OPTARG always contains value of parameter
        echo "Value is OK: $VALUE"
        else
          echo "Error! For parameter \"-a\" you did not provide an
            integer ranging from 10 to 300!"
          exit 1
        fi
      ;;
# ...ends on following slide...
```

Multiple switches in classical UNIX form (no positional) III

```
# ...continuing from previous slide...
    ?)
     echo "Invalid option(s)!"
     echo "See \"$0 -h\" for usage options."
     exit 1
     ;;
  esac
done # ...the end.
... # Following code...
```

- Script `interactive5.sh` contains complete example

```
# Start with displaying help
./interactive5.sh -h # Or ./interactive5.sh -v
# Try it as common command line tool
./interactive5.sh -i input.txt -o output.txt -a 50
./interactive5.sh -o output.txt -a 50 -i input.txt # Order doesn't matter
```

If branching

```
# Basic variant - commands are done only if condition is met
if expression; then
    commands
  fi
# Two branches - when condition is met and when not
if expression; then
    commands1 # expression is TRUE
  else
    commands2 # expression is FALSE - all other cases
  fi
# Join together two (or more) if branches
if expression1; then
    commands1
  elif expression2
    then
      commands2
    else
      commands3
    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

Selecting version of RAxML according the CPU type I

- RAxML can take advantage of modern CPUs to highly speed up calculations, with or without parallelization – every version has separated binary, user must select, see README
- After compilation, the script can select e.g. on remote server
- RAxML binaries must be in `$PATH`
- See `raxml_if.sh` for whole script

```
if grep -iq avx2 /proc/cpuinfo; then # Does the CPU support AVX2?
  RAXML='raxmlHPC-AVX2' # Select appropriate binary
  elif grep -iq avx /proc/cpuinfo; then # Does the CPU support AVX?
    RAXML='raxmlHPC-AVX' # Select appropriate binary
    elif grep -iq sse3 /proc/cpuinfo; then # Does the CPU support SSE3?
      RAXML='raxmlHPC-SSE3' # Select appropriate binary
      else # The very last option
        RAXML='raxmlHPC' # Slowest oldest CPU...
  fi # End of branching
$RAXML -s $INPUT ... # All the parameters as usually...
```

Selecting version of RAxML according the CPU type II

Multiple branching in one step

- Same task as on previous slide, but instead of it-then branching it is using `case`
- See `raxml_case.sh` for whole script

```bash
# Determine which CPU is available and which binary use then
CPUFLAGS=$(grep -i flags /proc/cpuinfo | uniq)
case "$CPUFLAGS" in
  *avx2*|*AVX2*) # Does the CPU support AVX2?
    RAXML='raxmlHPC-AVX2';; # Select appropriate binary
  *avx*|*AVX*) # Does the CPU support AVX?
    RAXML='raxmlHPC-AVX';; # Select appropriate binary
  *sse3*|*SSE3*) # Does the CPU support SSE3?
    RAXML='raxmlHPC-SSE3';; # Select appropriate binary
  *) # The very last option
    RAXML='raxmlHPC';; # Slowest oldest CPU...
  esac # End of branching
$RAXML -s $INPUT # All the parameters as usually...
```

For cycles

```
# Ways how to declare number of repetitions
for I in `seq 1 10`; do echo $I; done # "seq" is outdated
for I in 1 2 3 4 5 6 7 8 9 10; do echo $I; done
for I in {1..10}; do echo $I; done
for (( I=1; I<=10; I++ )); do echo $I; done
# One line for cycle for resizing of images (another option, as above)
for JPGF in *.jpg; do convert $JPGF -resize 100x100 thumbs-$JPGF; done
# More commands in a block
for JPGF in `ls -1 *.jpg`; do
  echo "Processing JPGF $JPGF"
  convert $JPGF -resize 100x100 thumbs-$JPGF
  echo "File thumbs-$file created"
  done
for ...; do # Start cyclus as you need
  command1 # command1 will be executed in any cas
  if (condition); then # Set some condition to skip command2
    continue; fi # Go to next iteration of the loop and skip command2
  command2
  done
```

While and until cycles

```
# 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
while ...; do # Start cyclus as you need
  commands...
  if [condition]; then # If something happens
    break; fi # End up the cyclus and continue by following commands
while read TEXTLINE; do # Run cyclus on text file
  commands...
  done < text_file_to_process.txt
while :; do echo "Press CTRL+C to exit..."; done # Infinite loop
for (( ; ; )) ; do echo "Press CTRL+C to exit..."; done # Infinite loop
```

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 `/usr/lib` and `/sr/lib64`, data 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 `/usr/local/bin` and so on)

Package management III

Installation of software

- Common distributions use to provide convenient graphical tool to manage software
    - Ubuntu Software Center
    - Aptitude – feature rich, advanced, for any DEB distribution
    - YaST Software for openSUSE
    - And many more...
- Distributions use to provide convenient simple update applet notifying about awaiting updates
- There use to be web services to look for packages, also from other sources – openSUSE, Debian, Ubuntu (+ Launchpad and PPAs), Fedora, ...
- The task is always same, the exact work-flow and commands more or less differ among distributions...
- Tools like Android Google Play and Apple Store inspired from Linux...

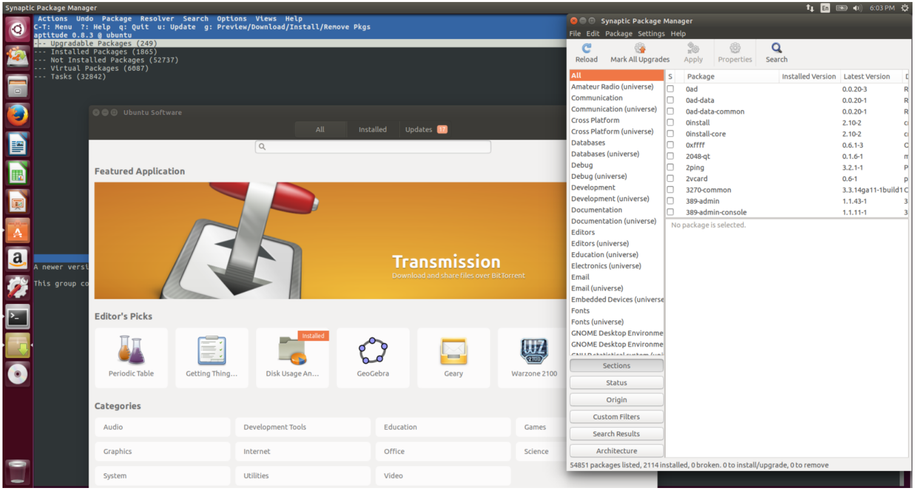
Package management in command line in openSUSE and Debian/Ubuntu (basic commands)

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 upgrade`
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`
List repositories	`zypper lr`	`cat /etc/apt/sources.list`
Add repository	`zypper ar` *repository*	`nano /etc/apt/sources.list`
Remove repository	`zypper rm` *repository*	`nano /etc/apt/sources.list`
For other tasks...	`rpm*` commands	`dpkg-*`, `apt-*` commands

Graphical package managers I

Ubuntu Software, and Synaptic and text-based Aptitude for all DEB-based distributions



```
 Actions  Undo  Package  Resolver  Search  Options  Views  Help
C-T: Menu  ?: Help  q: Quit  u: Update  g: Preview/Download/Install/Remove Pkgs
aptitude 0.8.3 @ ubuntu
--- Upgradable Packages (249)
--- Installed Packages (1865)
--- Not Installed Packages (52737)
--- Virtual Packages (6087)
--- Tasks (32842)

A newer versi
This group co
```

All
Amateur Radio (universe)
Communication
Communication (universe)
Cross Platform
Cross Platform (universe)
Databases
Databases (universe)
Debug
Debug (universe)
Development
Development (universe)
Documentation
Documentation (universe)
Editors
Editors (universe)
Education (universe)
Electronics (universe)
Email
Email (universe)
Embedded Devices (universe
Fonts
Fonts (universe)
GNOME Desktop Environme
GNOME Desktop Environme

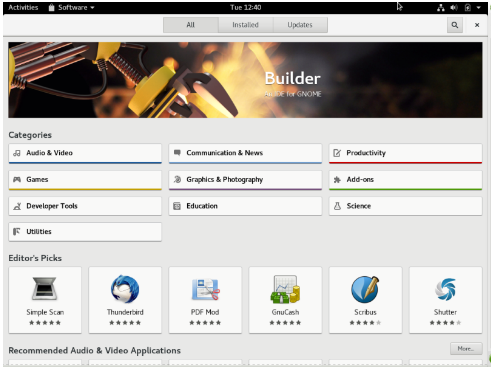
S	Package	Installed Version	Latest Version	D
☐	0ad		0.0.20-3	R
☐	0ad-data		0.0.20-1	R
☐	0ad-data-common		0.0.20-1	R
☐	0install		2.10-2	c
☐	0install-core		2.10-2	c
☐	0xffff		0.6.1-3	C
☐	2048-qt		0.1.6-1	m
☐	2ping		3.2.1-1	P
☐	2vcard		0.6-1	p
☐	3270-common		3.3.14ga11-1build1	C
☐	389-admin		1.1.43-1	3
☐	389-admin-console		1.1.11-1	3

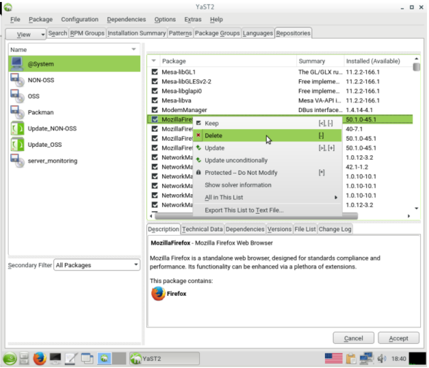
No package is selected.

54851 packages listed, 2114 installed, 0 broken. 0 to install/upgrade, 0 to remove

Graphical package managers II

GNOME Software in Fedora and YaST in openSUSE





Package	Summary	Installed (Available)
Mesa-libGL1	The GL/GLX ru…	11.2.2-166.1
Mesa-libGLESv2-2	Free impleme…	11.2.2-166.1
Mesa-libglapi0	Free impleme…	11.2.2-166.1
Mesa-libva	Mesa VA-API i…	11.2.2-166.1
ModemManager	DBus interface…	1.4.14-4.1
MozillaFire		50.1.0-45.1
MozillaFire		40-7.1
MozillaFire		50.1.0-45.1
MozillaFire		50.1.0-45.1
NetworkMa		1.0.12-3.2
NetworkMa		42.1-1.2
NetworkMa		1.0.10-10.1
NetworkMa		1.0.10-10.1
NetworkMa		1.0.10-10.1
NetworkMa		1.0.12-3.2

MozillaFirefox - Mozilla Firefox Web Browser

Mozilla Firefox is a standalone web browser, designed for standards compliance and performance. Its functionality can be enhanced via a plethora of extensions.

This package contains:

Firefox

- Practically every common general distribution has some graphical tool…

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
- 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...

```
# General schema within application directory with the source code:
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
```

Install tools needed for compilation

- You need to install compilation tools for your distribution and programming languages you are going to use
- Commonly, extra dependencies are required to compile the application
  - Packages for compilation use to end with `-dev` or `-devel` (e.g. if the software requires package `zlib` to run, install also its developmental version `zlib-dev(el)` to be able to compile it)
  - All requirements should be listed in README or INSTALL documents of particular package – user must install them manually...

```
# openSUSE and SUSE Linux Enterprise
zypper in -t pattern devel_basis devel_C_C++
# Debian, Ubuntu and derivatives like Linux Mint and others
apt-get install build-essential # Or "aptitude install build-essential"
# Red Hat, CENTOS, Scientific Linux, Older Fedora and derivatives
yum groupinstall "Development Tools" "C Development Tools and Libraries"
# Fedora since version 22
dnf group install "Development Tools" "C Development Tools and Libraries"
```

Compilation of RAxML

- Available from https://github.com/stamatak/standard-RAxML (cite Stamakis 2014)
- Before compilation check README
- This example does not require to run `make install`, it does not have extra dependencies for compilation, it requires specifying of particular source file by `make -f` (there are multiple `GCC` files, no one main)

```
mkdir raxml # Create working directory
cd raxml/ # Go there
# Get source code from GitHub (svn downloads only changed files)
svn co https://github.com/stamatak/standard-RAxML/tags/v8.2.9
cd v8.2.9/ # Go to newly created directory
ls # List files
rm -rf Windows* # No need of Windows version - delete it
# Compile standard version (other versions are available for better CPU)
make -f Makefile.gcc
rm *.o # Remove unneeded files (temporal for compilation)
./raxmlHPC -h # Launch it - see RAxML help
```

Compilation of SAMtools

- See http://www.htslib.org/download/
- Ensure packages `zlib` and `zlib-dev(el)` are installed – required for running and compilation, see INSTALL and README

```
wget https://github.com/samtools/samtools/releases/download/1.3.1/
  samtools-1.3.1.tar.bz2 # Download SAMtools
tar xjvf samtools-1.3.1.tar.bz2 # Unpack the archive
cd samtools-1.3.1/ # Go to the unpacked directory
./configure # Configure settings for compilation
./configure --help # See various configuring options
./configure --without-curses # Compile without ncurses support
make # Compile the software - check if there is error
     # Ensure developmental files for zlib (and ncurses) are available
make install # Copy products into final location - default /usr/local
make prefix=/where/to/install install # Install into custom location
make prefix=/home/$USER/bin install # Binary is in /home/$USER/bin/bin
make clean # Cleanup - final files are already in the destination
```

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
```

Windows applications on Linux I

- Applications written for one operating system do not work on the other systems...
  - They must be written in portable language like Java or script like Perl, Python or BASH
  - Otherwise we need an emulator – not everything works
- Windows 10 has possibility to run Linux applications, other option (for more Windows versions) is Cygwin (application must be specially compiled to support Cygwin)
- To run Windows applications on Linux use Wine
  - Search for packages named `wine` and install it
  - Sometimes, extra functionality is in extra packages – check `wine-*`
- To run DOS application on Linux use dosbox (package `dosbox`)
- As soon as Wine is installed, just click to Windows `*.exe` file...

Windows applications on Linux II

- Windows applications are installed into `~/.wine/` where Windows directory structure is created, launchers use to be placed to standard application menu into **Wine** section
- Use `winecfg` to change settings (e.g. version of Windows – can be different for each application, custom DLL library, …)
- `winefile` starts Windows file browser, `notepad` Notepad, `winemine` Mines
- To install some extra parts required by some applications use `winetricks`
    - Usage use to differ according to distribution and GUI
    - Browsing and selecting items to install can be bit messy…
    - It can be hard to check application requirements – if it fails, check if it is listed at https://appdb.winehq.org/ and/or run it from command line like `wine application.exe` and inspect errors in output

Windows applications on Linux III

- Before installing Windows application under Wine, check if there is some native Linux application to fit your needs...
  - Plenty of applications are available for more operating systems
  - Linux distributions use to have external contributor's sites to provide more packages
  - For many Windows-only applications there are fully comparable alternatives
- Some applications do not work under Wine (from various reasons), some complex packages are supported commercially (I have no experience with it)
- Wine is well compatible with rest of the Linux hosting system, it is considerable to install Windows in e.g. WirtualBox (or another virtualization platform), if needed

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
  - MetaCloud – computing (HPC) cloud similar to e.g. Amazon Elastic Compute Cloud (EC2) or Google Compute Engine
  - ownCloud to backup and/or sync data across devices (default capacity is 100 GB, user may ask for more)
    - It is possible to connect by webDAV to ownCloud (slide 95) – many applications support it
    - It is possible to share calendars and/or address books via claDAV and cardDAV among devices and/or people

CESNET and MetaCentrum II

- Services accessible without registration
  - ownCloud `https://owncloud.cesnet.cz/`
  - FileSender `https://filesender.cesnet.cz/`
  - 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`
  - After registration for MetaCentrum, user can join MetaCloud via `https://perun.metacentrum.cz/fed/registrar/?vo=meta&group=metacloud`
  - Users not having access to EduID have to register first at HostelID `https://hostel.eduid.cz/en/index.html`

CESNET and MetaCentrum III

  - 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/`
- Information about MetaCloud `https://wiki.metacentrum.cz/wiki/Kategorie:Clouds`
- 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

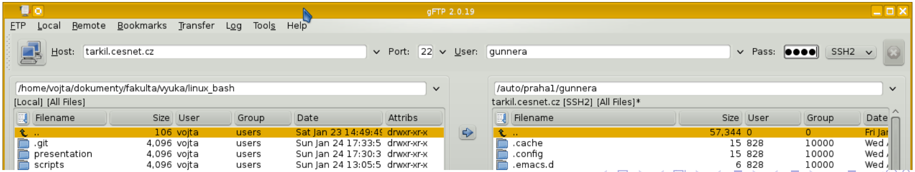
- User can transfer data on one of frontends by e.g. `scp` or WinSCP from Windows
- Same credentials are used for all front ends, for SSH login as well as file transmissions

```
# Login to selected server (tarkil is located in Prague)
ssh USER@tarkil.metacentrum.cz
module add torque-client # Switch job scheduler to PBS Torque on tarkil
# Continue as in any other command line...
qsub ... # Submit the job (see later)
```

- 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.metacentrum.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), and `tarkil` (in Prague) – so that e.g. `tarkil.grid.cesnet.cz` is synonymous to `tarkil.metacentrum.cz`
- See slide 95 and following to command-line transfers of files



Host: tarkil.cesnet.cz Port: 22 User: gunnera Pass: ●●●● SSH2

/home/vojta/dokumenty/fakulta/vyuka/linux_bash
[Local] [All Files]

Filename	Size	User	Group	Date	Attribs
..	106	vojta	users	Sat Jan 23 14:49:4	drwxr-xr-x
.git	4,096	vojta	users	Sun Jan 24 17:33:5	drwxr-xr-x
presentation	4,096	vojta	users	Sun Jan 24 17:30:3	drwxr-xr-x
scripts	4,096	vojta	users	Sun Jan 24 13:05:5	drwxr-xr-x

/auto/praha1/gunnera
tarkil.cesnet.cz [SSH2] [All Files]*

Filename	Size	User	Group	Date
..	57,344	0	0	Fri Jar
.cache	15	828	10000	Wed
.config	15	828	10000	Wed
.emacs.d	6	828	10000	Wed

Basic skeleton of script running tasks I

```bash
#!/bin/bash
# Modify the script according to your needs!
# Set data directories
WORKDIR="bayes_batch" # Or something else
DATADIR="/storage/praha1/home/$LOGNAME" # Or other storage
# 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-3.2.6
# 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
```

Scheduling details I

- `tarkil` is currently testing new scheduler PBS Professional
  - It is using only limited number of machines, the syntax is slightly different from version described here
  - Rather use for now another frontend or switch to old scheduler by `module add torque-client` and continue as usual (`qsub`)
- Specify needed time (up to 4 weeks; day, hours): `-l walltime=3w:4d:6h`, `-l walltime=6d:18h`, …
  - User can ask to prolonge the walltime
- Ask for as much RAM as you need (e.g. `-l mem=8gb`)
  - If the task is going to require more, than allowed, system kills it…
  - If user doesn't use all required RAM, the system temporarily lowers priority for future tasks
  - It can be hard to estimate…

Scheduling details II

- Disk space is relatively free resource, user can ask more to have some reserve (e.g. `-l scratch=10gb`)
- Specify how many physical computer you are going to use (e.g. `-l nodes=1` for one machine) and number of CPU threads on each machine (e.g. `-l nodes=1:ppn=8` for 1 machine with 8 cores or `-l nodes=2:ppn=4` for 2 machines, each with 4 CPU threads)
  - It use to be necessary to specify correct number of threads for the application (e.g. `parallel -j 4`) – the application sees all CPUs on the machine, but can't use them
  - If the application consumes less than required, the system temporarily lowers priority for future tasks, if it try to use more, it will be very slowed down
  - If using more physical machines, ensure correct settings of e.g. MPI

Scheduling details III

- If requesting e-mails (e.g. `-m abe` to get mail about submission, starting and termination of the task) and submitting plenty of tasks by some script, it can result in hundreds of mails – mail servers don't like it...
- Every user has certain priority highered by acknowledgments in publications to MetaCentrum and lowered by intensive usage of the service (the usage is calculated from past month)
- After submission of the task, check in the queue in which state it is – sometimes it can't start because of impossible combination of resources or so
- User can check load of machines

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

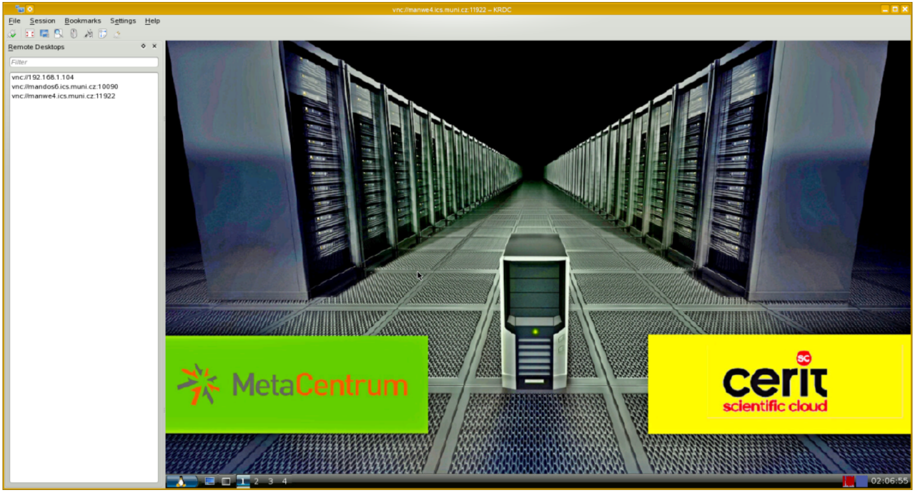
Graphical interactive task

- 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
MetaCentrum
cerit scientific cloud
02:06:55

vnc://192.168.1.104
vnc://mandos6.ics.muni.cz:10090
vnc://manwe4.ics.muni.cz:11922

Managing system services

- Different among distributions – several main methods
- In Linux, most common is SystemD, less common older init scripts and RC scripts
- Mac OS X and various BSD and other systems have different methods
- Used to manage system services like networking, cron, web server, database, ...
- Read documentation for your distribution!
- Most of 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

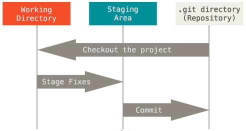
```
# List installed services and their status
systemctl list-units --type service
# Enable/disable/see status/start/stop/restart/... service
systemctl enable/disable/status/start/stop/restart servicename # TAB...
# Show overriden config files
systemd-delta
# Analyze boot time (how long does each service 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
```

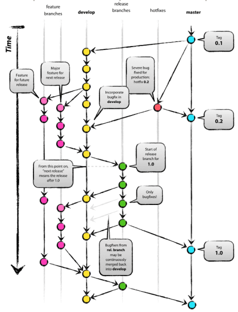
Git and GitHub

- Git is version controlling system – it traces changes among all versions – absolutely crucial for any software development
- GitHub is currently probably the most popular platform to host development of open-source projects, see documentation
- Older (nowadays not so common) version controlling systems is Subversion (SVN), there are many more
- Probably the best textbook for Git is Chacon's Pro Git
  - Dostupná i česky (včetně prvního vydání)
- Changes and their history is stored in repository (local or network, shared or private) – it is possible to view any historical state and differences between any versions
- It is possible to trace who and when did what
- Branching and merging of branches helps with making of big changes

Git principles

- Three main areas
  1. Working directory
  2. Staging (changes awaiting to be pushed to the repository)
  3. Git repository (remote/local)
- Everyone has whole repository and history – very robust
- Flexible branches
  - Very convenient
  - Keeping work structured
  - Separation of tasks
  - Keeping more versions of the project in parallel




feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

https://git-scm.com/, http://nvie.com/

Working with Git – start and sending changes

```
# Create a new repository for new project
git init # No need when cloning from existing repository
# Or checkout (make a copy) for another local or remote repository
git clone /path/to/local/repository # Locally mounted repository
git clone username@host:/path/to/remote/repository # Over SSH
git clone https://github.com/V-Z/course-linux-command-line-bash-
  scripting-metacentrum.git # Clone from web, e.g. GitHub
# Add files to trace with Git
# Ignored files (or patterns) can be listed in .gitignore file
git add <files> # Or "git add *"
# Commit changes to prepare then to send to repository
git commit -m "Message..."
# If you did not start by cloning, add connection to server
git remote add origin <location> # Do only once on the beginning
# <location> can be remote server or local path
git remote add origin . # For repository within working directory
# Push changes into the (regardless where it is) repository
git push origin master # See further for selection of branches
```

Working with Git – branching and getting changes

```
# Making new branch and switching to it
git checkout -b NewFeature # Now we are in branch NewFeature
# Switch back to branch master
git checkout master # Generally, "git checkout <branch>"
# Delete the branch (changes there are lost, must be in another branch)
git branch -d NewFeature # Delete local branch, to delete remote do:
git push origin --delete <branch>
# Branch must be also pushed to the remote server
git push origin <branch>
# List branches (current is marked by asterisk on the beginning)
git branch
# Update local repository to the newest version from central repository
git pull # Fetch and merge remote changes
# Merge another branch into the current one
git merge <branch>
# In case of conflict, git shows editor and user must fix it manually
git add <file with conflict> # Needed to re-add conflicting file
# To see changes before merging
git diff <source_branch> <target_branch>
```

Working with Git – tags, logs and settings

```
# Tagging e.g. milestones, released versions of software, etc.
git tag <name> <commit id> # <name> can be custom, <commit id> from log:
git log # Newest is on top, see also "git log --help"
git log --graph --oneline --decorate --all # Full long log
# Discard local changes for particular file
git checkout -- <file>
# Discard all local changes
git fetch origin # Overwrite local changes
git reset --hard origin/master # If local repository is broken...
gitk # Graphical interface
git config color.ui true # Set output to be colored
git config format.pretty oneline # Nicer log output
~/.git # Contains user's settings, .git in every repository contains
       # data and settings for particular repository
```

Resources to learn to work in the terminal I

- openSUSE (general handbook) https://doc.opensuse.org/documentation/leap/startup/html/book.opensuse.startup/part.bash.html
- Ubuntu (general handbook) https://help.ubuntu.com/community/UsingTheTerminal
- BASH full reference manual https://www.gnu.org/software/bash/manual/ (advanced)
- Debian (general handbook) https://www.debian.org/doc/manuals/debian-reference/ch01.en.html
- Guide to Unix https://en.wikibooks.org/wiki/Guide_to_Unix
- BASH for beginners http://tldp.org/LDP/Bash-Beginners-Guide/html/Bash-Beginners-Guide.html (the site has plenty of good resources)

Resources to learn to work in the terminal II

- BASH Guide for Beginners http://linuxcourse.rutgers.edu/documents/Bash-Beginners-Guide/
- Linux tutorial http://ryanstutorials.net/linuxtutorial/
- Getting Started with BASH http://www.hypexr.org/bash_tutorial.php
- Bash Guide http://mywiki.wooledge.org/BashGuide
- Učebnice Linuxu https://www.abclinuxu.cz/ucebnice

Manuals for package management

- openSUSE AND SUSE Linux Enterprise: https://doc.opensuse.org/documentation/leap/startup/html/book.opensuse.startup/part.reference.software.html and https://en.opensuse.org/Zypper
- Red Hat, Fedora, CENTOS, Scientific Linux and others: http://yum.baseurl.org/
- Debian (and derivatives): https://www.debian.org/doc/manuals/debian-reference/ch02.en.html and https://wiki.debian.org/PackageManagement
- Ubuntu (and derivatives): https://help.ubuntu.com/stable/ubuntu-help/addremove.html and https://help.ubuntu.com/community/AptGet/Howto
- And another distributions...

How to ask for help I

- Never ever ask simple silly lazy questions you can quickly find in manual or web
- People on mailing lists and forums respond volunteerly in their spare free time – do not waste it – be polite, brief and informative
- Be as specific and exact as possible
  - Write exactly what you did ("It doesn't work!" is useless...)
  - Copy/paste your commands and their output, especially error messages – they are keys to solve the problem
  - Try to search web for the error messages (or their parts)
  - Try to provide minimal working example – add at least part of your data (if applicable) so that the problem is reproducible
  - Specify name version(s) of distribution ans problematic software
- **OSS is free as freedom of speech – not as free beer!**

How to ask for help II

    - As soon as you don't pay for support, you can't blame anyone for lack of responses
    - There are plenty of reasons some software doesn't work – usage/data author didn't expect, unsupported version of operating system, author's mistake, user's mistake, ...
    - Authors wish their software to be useful – constructive feedback, reporting bugs and wishes is welcomed, but it must be provided in the way useful for the developer
- Imagine you should answer – which information do you need?
- Try to find the best place to ask your question – specific forum for particular distribution or software use to be the best option
- Learning command line is like learning foreign language...

Main general fora

- Probably the best are fora from StackExchange https://stackexchange.com/sites
- General forum for programmers https://stackoverflow.com/
- UNIX forum https://unix.stackexchange.com/
- Forum for administrators https://superuser.com/

openSUSE

- Homepage https://www.opensuse.org/
- Wiki https://en.opensuse.org/Main_Page, documentation https://doc.opensuse.org/
- News https://news.opensuse.org/, developers' blogs https://lizards.opensuse.org/, blogs http://planet.opensuse.org/
- Forums https://forums.opensuse.org/, mailing lists https://lists.opensuse.org/
- Community http://opensuse-community.org/, guide http://opensuse-guide.org/

Debian, Ubuntu, Linux Mint and derivatives

- Debian `https://www.debian.org/`
- Ubuntu `https://www.ubuntu.com/`, support `https://www.ubuntu.com/support`, Ask ubuntu `https://askubuntu.com/`
- Kubuntu `http://www.kubuntu.org/`, forum `https://www.kubuntuforums.net/`
- Xubuntu `http://xubuntu.org/`, Lubuntu `http://lubuntu.net/`
- Linux Mint `https://www.linuxmint.com/`

Fedora

- Homepage https://getfedora.org/
- Official forum https://ask.fedoraproject.org/
- Documentation https://docs.fedoraproject.org/
- Community forum http://fedoraforum.org/

KDE and LibreOffice

- LibreOffice https://www.libreoffice.org/, Document Foundation https://www.documentfoundation.org/
- KDE https://www.kde.org/, forum https://forum.kde.org/, application store https://store.kde.org/, KDE for education https://edu.kde.org/, blogs https://planetkde.org/

České weby – zdroje informací a fóra

- ABC Linuxu https://www.abclinuxu.cz/
- Root https://www.root.cz/
- LinuxExpres https://www.linuxexpres.cz/
- LinuxDays (největší konference) https://www.linuxdays.cz/
- OpenOffice/LibreOffice https://www.openoffice.cz/ a https://www.knihaoffice.cz/
- Ubuntu http://www.ubuntu.cz/, wiki http://wiki.ubuntu.cz/, fórum http://forum.ubuntu.cz/, Kubuntu http://www.kubuntu.cz/, Lubuntu http://www.lubuntu.cz/, Xubuntu http://www.xubuntu.cz/
- Fedora https://mojefedora.cz/, příručka https://wiki.mojefedora.cz/doku.php?id=navody:prirucka:obsah
- openSUSE http://www.opensuse.cz/
- Linux Mint https://www.linux-mint-czech.cz/

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 XƎLaTeX on openSUSE GNU/Linux January 28, 2017