Linux, command line & MetaCentrum

Use of Linux command line not only for CESNET's MetaCentrum

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 14 to 16, 2020

Outline I

Outline II

Outline III

Outline IV

Outline V

Introduction

First steps in the world of Linux and open-source software

The course information

- The course page: https://trapa.cz/en/course-linux-command-line-2020
  - Česky: https://trapa.cz/cs/kurz-prikazove-radky-linuxu-2020
- Subject in SIS: https://is.cuni.cz/studium/eng/predmety/index.php?do=predmet&kod=MB120C23
  - Česky: https://is.cuni.cz/studium/predmety/index.php?do=predmet&kod=MB120C23
  - For students having subscribed the subject, requirements are on next slide
- Working version of the presentation 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

Requirements to exam ("zápočet")

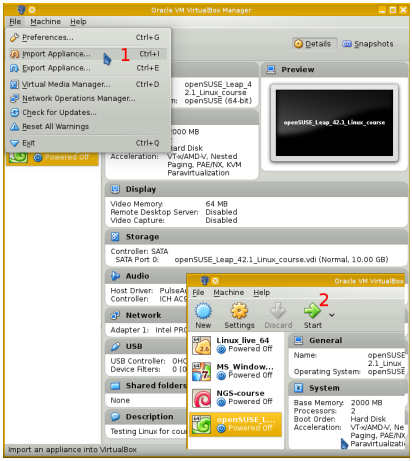
1. Be present whole course.
2. Be active — ask and answer questions.
3. Write short script solving any task the student is (going to be) solving. E.g. prepare script to process student's data on MetaCentrum. Or short script to do anything the student needs to do. This will be very individual. According to topics and interests of every student. Students can of course discuss with anyone, use Internet, manuals, etc. The aim is learn how to solve real problem the student has/is going to have.
4. Write at least one page (can be split into multiple articles) on Wikipedia about any topic discussed during the course. Again, this is very open, students can write about any topic they like. I prefer native language of the student (typically to make larger non-English Wikipedia).

Materials to help you...

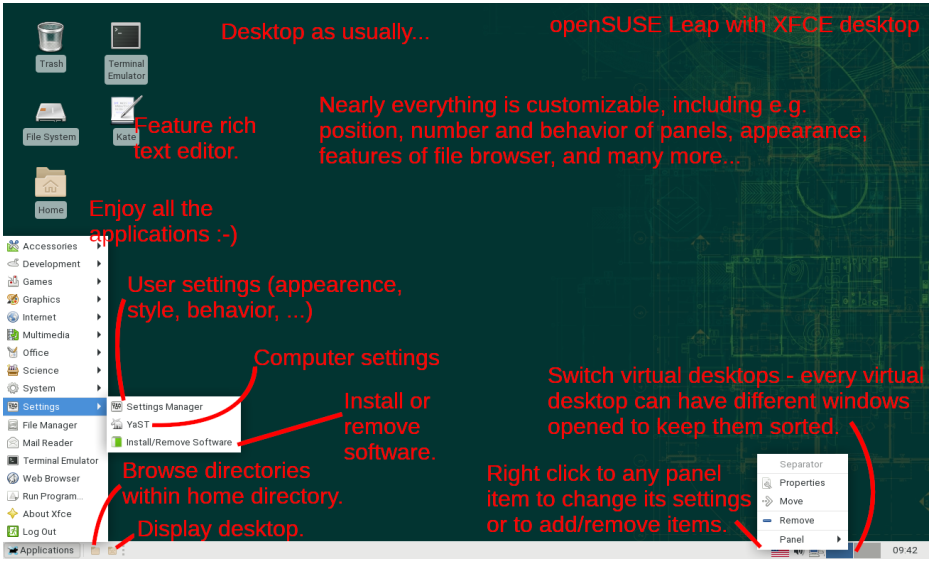
- Download the presentation from `https://soubory.trapa.cz/linuxcourse/linux_bash_metacentrum_course.pdf`
- Download the scripts and toy data from `https://soubory.trapa.cz/linuxcourse/scripts_data.zip`
  - Note: Open the scripts in some good text editor (slide 138) — showing syntax highlight, line numbers, 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 15.0 Linux distribution for this course from `ftp://botany.natur.cuni.cz/openSUSE_Leap_courses.ova` (∼5 GB)
- Launch VirtualBox and go to menu **File | Import appliance...** to import it. When done, launch it (**Start**)

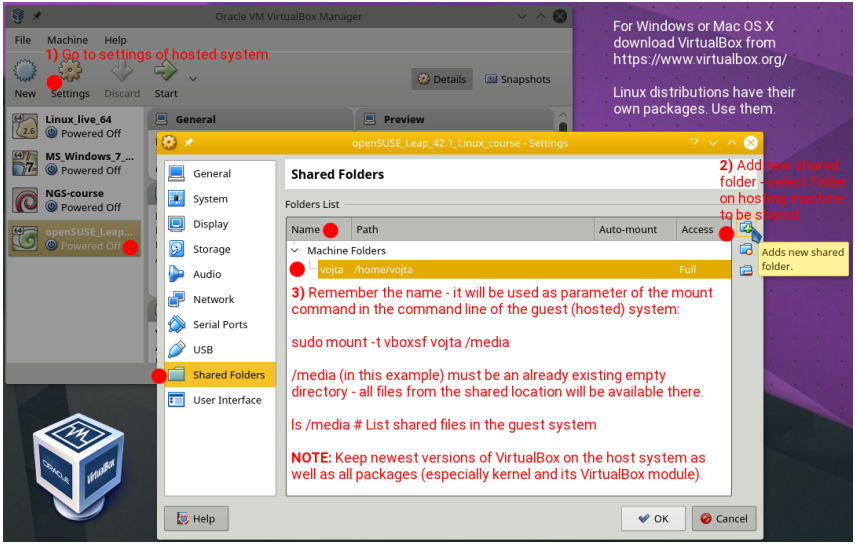


Enjoy learning virtual machine for the course



VirtualBox shared folder I

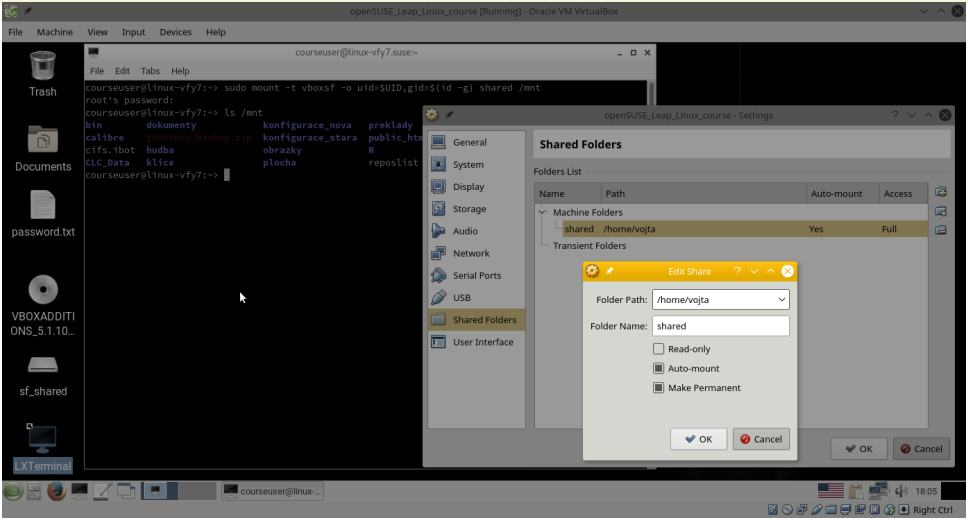
VirtualBox can be configured to share folder with host operating system



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
```



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, macOS, ...)
    - Mainly open-source (UNIX is commonly commercial — source code is not publicaly available, 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

What it is UNIX, Linux and GNU II

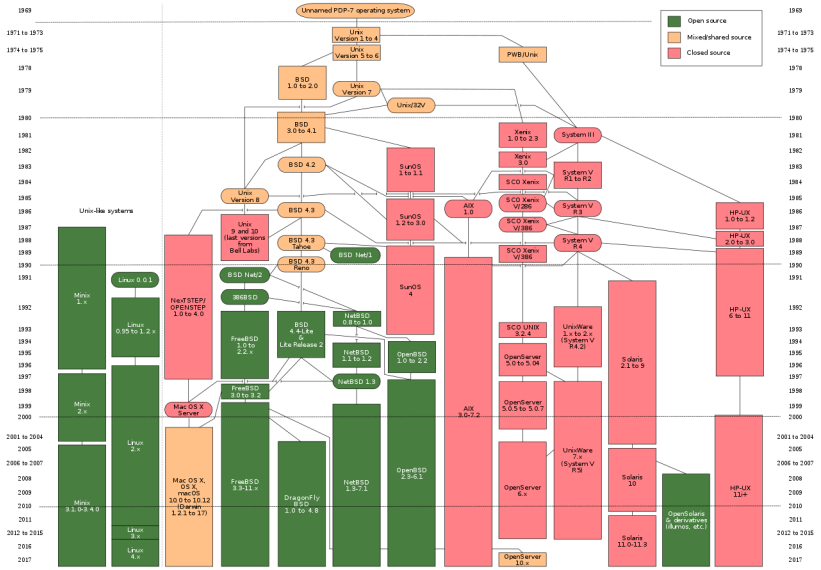
- Many courts about copyrights, parts of code, patents — USA allow software patents, EU not — GNU, Linux, BSD, etc. try to ensure to have only code not covered by any patents to avoid possible courts

- GNU
  - "GNU's Not Unix!" — but it is compatible, respects its principles
  - System written from scratch, following ideas of UNIX
  - 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*, macOS, ...), also present in many commercial paid UNIX systems
  - Source code is free and open — 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

What it is UNIX, Linux and GNU III

- Kernel was in principle inspired by various UNIX systems and using GNU tools for standard work
- Quickly became popular — anyone can take it and use for any needs, adopt (modify it), etc.
- 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, ...
- Most of people working with UNIX are using Linux (or macOS)

Extremely simplified UNIX phylogeny



Most common UNIX-based systems (except Linux)

- macOS (previously Mac OS X) — system kernel is based on older BSD and uses plenty of GNU tools (although mostly older outdated versions)
- BSD — one of the oldest operating systems, still developed in many independent variants
  - Still very popular especially on servers, for special purposes, etc.
  - License allows closing of the code — used by Apple macOS kernel, PlayStation firmware, ...
  - Installation and management is for beginners usually harder than Linux, everything must be done manually, not so common as Linux anymore
  - E.g. FreeBSD, DragonFly BSD, OpenBSD, FreeNAS (for storage servers), ...
- Solaris — commercial, not very common
  - Mainly special servers, paid
  - Several community-based variants freely available
- Command line usage is nearly same across UN*X systems (all follow same standards)

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...
- According to book by Eric S. Raymond

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
- 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 e.g. Apple macOS, Safari browser, small electronics, ...)
- Apache or Mozilla licenses etc. — specific use in particular software
- Creative Commons (CC) — software licenses are not suitable for multimedia, text, etc. — CC has many options (including denial of reuse of the product), see `https://creativecommons.org/`

Free and open-source software — (F)OSS II

Spirit of FOSS

- Orientation might be tricky, but practical output for users is same — **the software can be independently checked for bugs, backdoor, 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...

- And many more licenses...
- Practical output of using open-source licenses for the user
  - Software can be used anywhere
  - Software can be modified
  - User can learn from the software (from the code), new software can be developed on top of it
  - Easier to find and trace bugs
  - Security — no back doors

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" with many limitations)
- 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

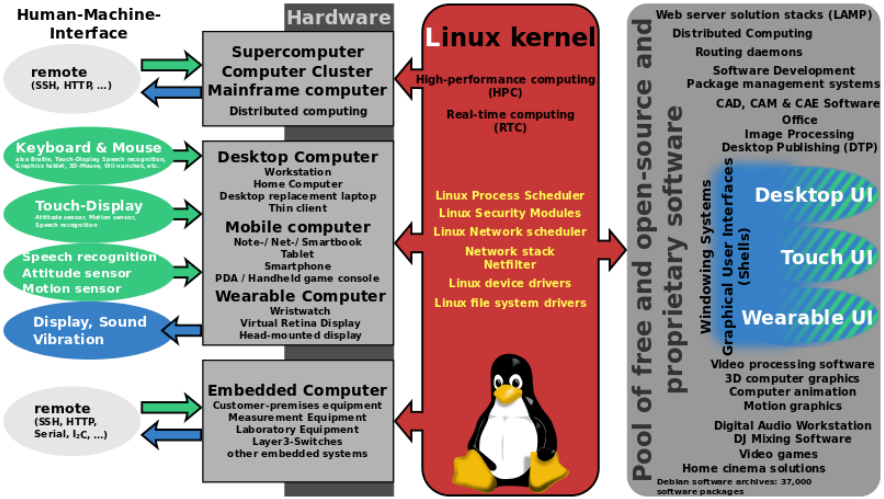
Linux

Generally about Linux

What it is a “Linux”

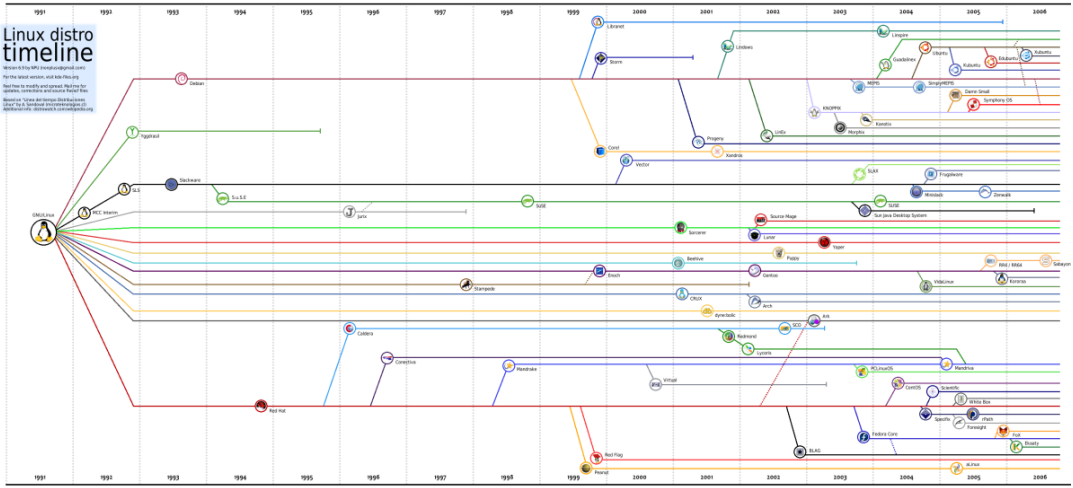
- Operating system respecting principles of UNIX
- Components
  - Linux **kernel** — basic part of the system responsible for hardware and very basic low-level running of the system (“**Linux *sensu stricto***”)
  - GNU core utilities — basic applications
  - Graphical user environment (GUI) — many choices
  - Many other applications — according to use — whatever imaginable
- Linux **distribution (“Linux *sensu 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 parts 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/ (∼850 distributions, ∼250 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
  - Mageia, PCLinuxOS, ...
- Manjaro, Solus, ...
- Android (practically Linux kernel, touch interface & Java)
- For experienced users: Arch, Slackware, Gentoo, ...

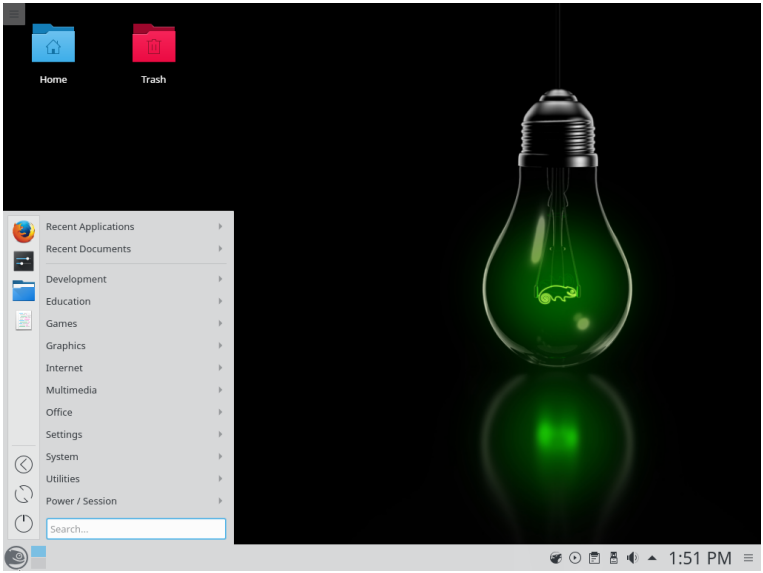
Graphical User Interfaces (GUI)

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

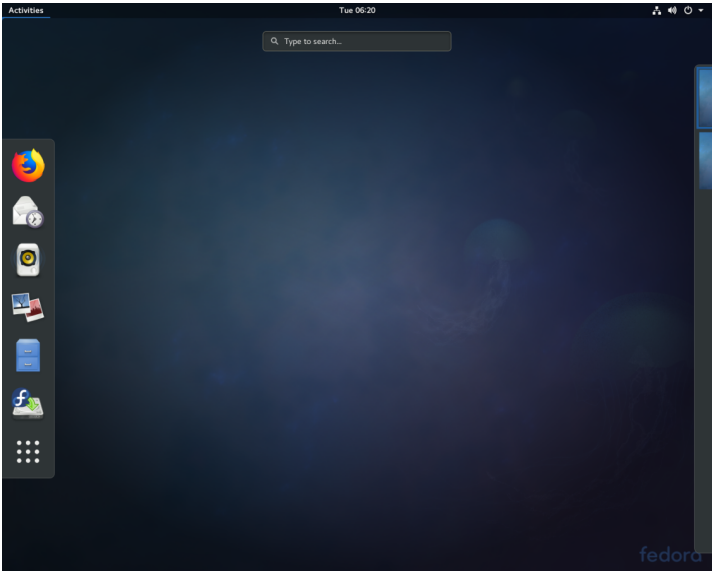
- Most of GUIs are available for most of common distributions — one is picked as default and “only” color style is different
- Unity — originally developed by Ubuntu (discontinued, now community based), “Mac-style”
- KDE — one of the most common, feature extremely rich, basically “Windows-like” (can 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 GNOME

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

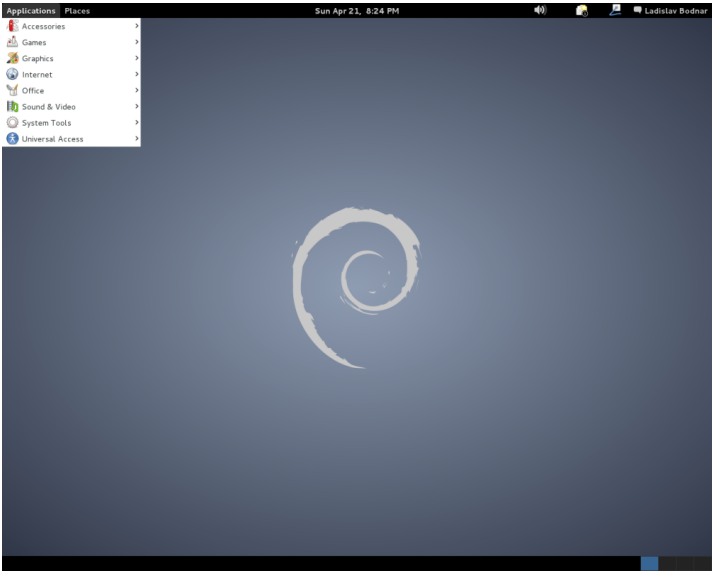


Fedora with GNOME — GNOME is always almost same

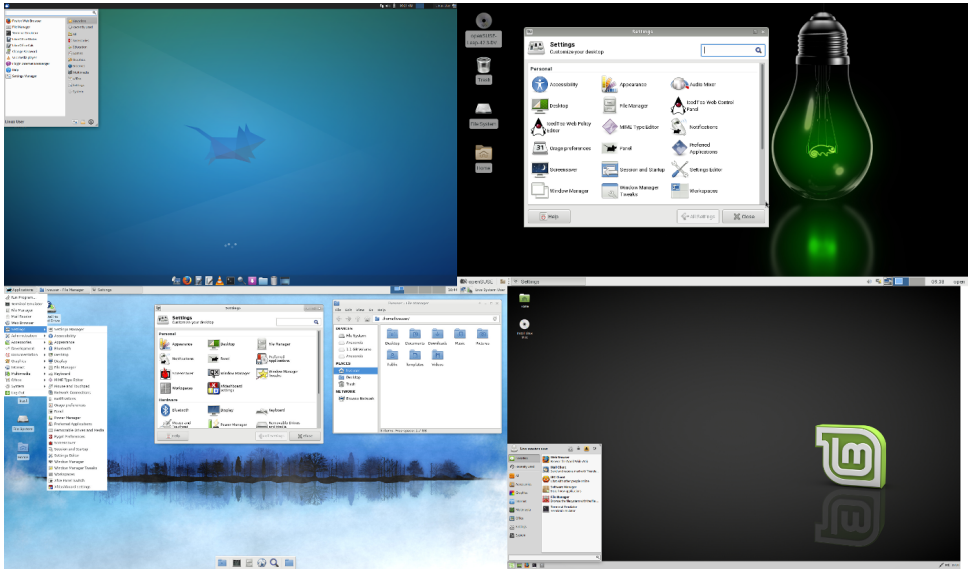


Linux Mint with Cinnamon

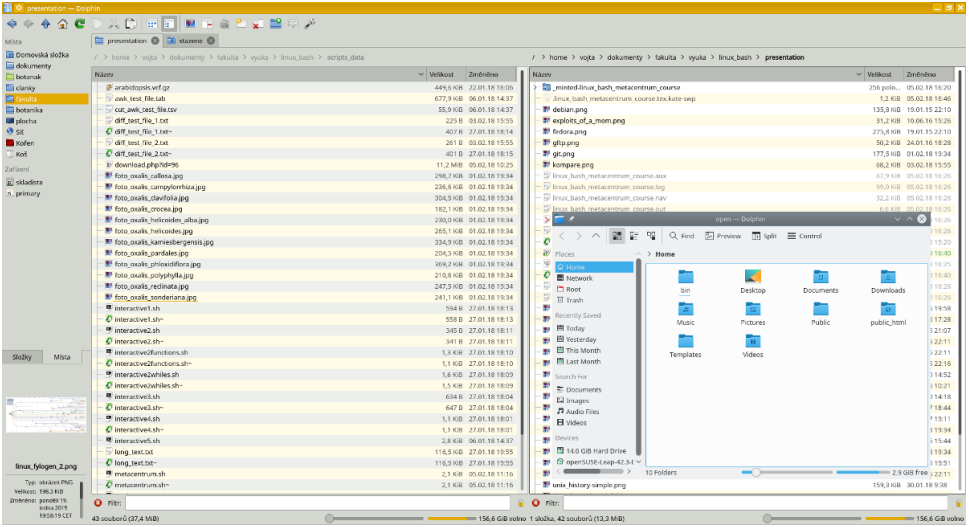
Debian with XFCE — Xubuntu has more “modern” design



XFCE in Xubuntu, openSUSE, Fedora and Linux Mint



Dolphin (KDE file manager) — default for openSUSE (inset) and after tuning in the same distribution...



How to try Linux? I

- 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...
  - Major updates of Windows 10 or Windows reainstalls use to destroy bootloader and it is not possible to start Linux anymore (although it can be usually easily fixed)
- 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 11)
  - Requires relatively powerful computer (preferable Intel i5 or i7 or AMD Ryzen and over 4–8 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
- Linux subsystem in MS Windows 10 Store

How to try Linux? II

- To install follow https://docs.microsoft.com/windows/wsl/about
- Only for command-line applications, does not aim to GUI
- Has some problems with paths, text files, so sometimes it is not usable very well...

- Cygwin
  - Download and install from https://www.cygwin.com/
  - It is not native Linux, it is collection of GNU and open-source utilities compiled to work on Windows
  - Follows POSIX standards (i.e. it works like normal UNIX command line, with all features)
  - Every application must be specially compiled to be able to work under Cygwin (it is sometimes complicated)
  - Collection is large, include also GUI and DE, but not everything is easy working
  - Installation of custom software might be very hard...

The Linux diversity...

- Try several distributions and just choose one you like...
- If 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 — install some terminal emulator there :-)
- 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 (others 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 server) — 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 (macOS, ...)

UN*X

Theory and principles of UNIX-based operating systems

Short overview of hard disk layout

- Physical disk (piece of hardware) has at least 1 partition — division seen in Windows as "disks" (C, D, ...) and mounted directory in UNIX
- MBR — older description of disk division, up to 4 primary partitions (OS typically requires at leas one to run), one can be extended and contain more partitions, disks up to 2 TB
- GPT — newer, no relevant limits, requires UEFI (replacement of BIOS — 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!

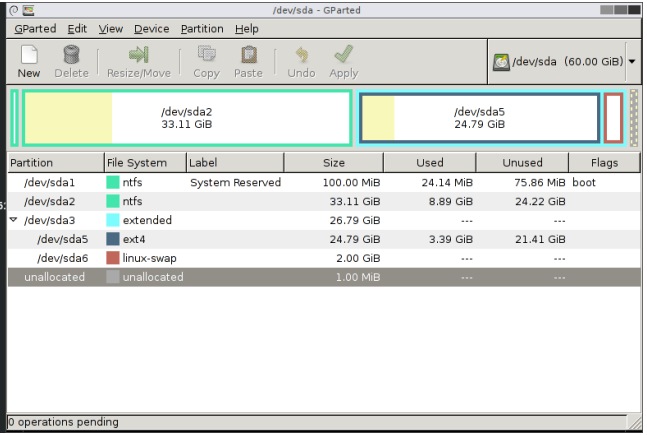
Comparison of file systems (limits and compatibility)

FS name	Name length	Signs in name	Path length	File size	Partition size	Supported systems
FAT32	255	Unicode	No limit defined	4 GiB	2 TiB	Any
exFAT	255	?	No limit defined	16 EiB	64 ZiB	Any
NTFS	255	Variable	Variable	16 TiB	16 EiB	Windows (UN*X)
HFS+	255	Unicode	Unlimited	8 EiB	8 EiB	macOS
ext4	255	Any, not /	No limit defined	16 TiB	1 EiB	Linux (UN*X)
XFS	255	Any	No limit defined	9 EiB	9 EiB	Linux (UN*X)
Btrfs	255	Any	No limit defined	16 EiB	16 EiB	Linux (only?)
ZFS	255	Unicode	No limit defined	16 EiB	256 ZiB	UN*X

- FAT32 (including extensions) is old-fashioned and not reliable FS, but still common in various flash disks and memory cards
- NTFS (basic Windows FS) and FAT do not support UNIX permissions, so they can't be used as system partition in Linux; see also full comparison
- Btrfs, ext4, XFS and ZFS are not accessible from Windows at all
- Btrfs, XFS and ZFS are the most advanced FS in common use
- Linux users use mainly ext4, less Btrfs or others

Creation and control of FS I

```
$ sudo fdisk -l
[sudo] password for root:
Disk /dev/sda: 894.26 GiB, 960197124096 bytes, 1875385008 sectors
Disk model: Corsair Force LE
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: gpt
Disk identifier: 55C22C41-DBE7-4E05-8139-FC47E5E342F4

Device       Start        End    Sectors  Size Type
/dev/sda1     2048     819199     817152  399M Microsoft basic data
/dev/sda2   819200    1124351     305152  149M EFI System
/dev/sda3  1124352 1875384319 1874259968 893.7G Linux LVM


Disk /dev/mapper/cr_ata-Corsair_Force_LE_SSD_16298021000105550200-part3: 893.73 GiB, 9596
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes


Disk /dev/mapper/pocitac-odkladiste: 8.72 GiB, 9349103616 bytes, 18259968 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes


Disk /dev/mapper/pocitac-korinek: 885 GiB, 950261514240 bytes, 1855979520 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes


Disk /dev/sdb: 931.53 GiB, 1000204886016 bytes, 1953525168 sectors
```



- Work in command line or use graphical tool like GParted...
- All commands require root privileges (slide 50)

Creation and control of FS II

- `fdisk -l` lists disks and partitions
- To manage disk partitioning use `fdisk /dev/sdX` or `gdisk /dev/sdX`
- When hard drive is partitioned, partitions must be formatted in next step
- 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`
  - Btrfs uses `btrfs check /dev/sdXY`, if it is unmountable, `btrfs-zero-l /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

Creation and control of FS III

- `tune2fs -parameters /dev/sdXY` can set various parameters to influence behavior of disk (labels and more) partition
- `hdparm -parameters /dev/sdX` can set advanced hardware parameters of hard drive
- The most convenient is using graphical tools available in all distributions...
  - In openSUSE there is YaST administrative module — from command line launch `yast --qt partitioner` 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
- Users must be sure what they are doing, otherwise system can be damaged

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 in 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/sdX 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 sdX!)
lnav # Comfortably browse recent logs, quit by "q"
```

- If there are encrypted partitions, they are in /dev/mapper/
- If LVM (slide 49) is used, see lvscan and pvscan to find correct location in /dev/
- Disks are also accessible through /dev/disk/by-<TAB><TAB

Mounting and unmounting disks and removable media

- Mounting and unmounting of devices require root privileges
- In modern Linux distributions, mounting is done automatically and media are visible in `/media` or `/run/media`
- 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, add '-t FS_type'
mount /dev/sdXY /mount/directory # Mount disk sdXY to /mount/directory
umount /dev/sdXY # Unmount disk sdXY, alternatively use below command
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
```

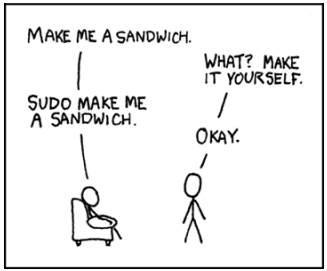
Put together more disks

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 (in RAID 6 two disks are used for parity control), little bit slower, popular in cheaper storage servers (NAS)
- 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 Btrfs and ZFS

Root vs. “normal” user

- Root is administrator — more than God (of the server) — 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



https://xkcd.com/149/

Becoming root

- Root privileges are required for any administrative task (install of software package, change of system settings, ...)
- Word “root” is used as name for system administrator user, and also for top of filesystem directory hierarchy (/)

```
# 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 (USER's 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
                 # /etc/sudoers can contain special settings for
                 # particular users and/or groups
cat /etc/passwd # See all users (including system users)
```

Directory structure in Linux I

- It is similar 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
- `/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

Directory structure in Linux II

- `/mnt` — usually manually mounted file systems (but they can be mounted elsewhere according to needs)
- `/opt` — optional, usually locally compiled software
- `/proc` — dynamic information about system processes
- `/root` — root's (admin's) home
- `/run` — temporal ID files (locks) of running processes
- `/sbin` — basic system utilities
- `/selinux` — SELinux is security framework
- `/srv` — FTP and WWW server data (can be in `/var/srv`)
- `/sys` — basic system
- `/tmp` — temporary files — users have private dynamically created spaces there, used automatically by applications according to need
- `/usr` — binaries (executable applications) and libraries of installed applications

Directory structure in Linux III

- `/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
- E.g. MetaCentrum has storage servers in various locations accessible from front and calculation nodes in `/storage`
- 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, etc.)
- 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 for new users (home directory, default shell, ...)
- `resolv.conf` — DNS settings (part of basic networking)
- `shadow` — users passwords in encrypted form
- `skel` — basic directories and configuration for new users
- Much more according to software installed...

Example of configuration in /etc/ssh/sshd_config

```
# This is the ssh client system-wide configuration file.  See
# ssh_config(5) for more information.  This file provides defaults for
# users, and the values can be changed in per-user configuration files
# or on the command line.
# Minimum accepted size of the DH parameter p. By default this is set to
# 1024 to maintain compatibility with RFC4419, but should be set higher.
# Upstream default is identical to setting this to 2048.
#KexDHMin 1024
Host *
# If you do not trust your remote host (or its administrator), you
# should not forward X11 connections to your local X11-display for
# security reasons: Someone stealing the authentification data on the
# remote side (the "spoofed" X-server by the remote sshd) can read your
# keystrokes as you type, just like any other X11 client could do.
# Set this to "no" here for global effect or in your own ~/.ssh/config
# file if you want to have the remote X11 authentification data to
# expire after twenty minutes after remote login.
ForwardX11Trusted yes
```

Everything is (text) file

- In UNIX specifications, everything is (text) file
  - Technically, directory is "just" a file listing its content
  - Text files are easy to read, parse, manipulate
    - Very easily editable (easy to change configuration)
    - Easy to transfer to another system
    - Easily comparable among users/versions/systems
  - UNIX command line tools are the most powerful when processing text files (of any sort, e.g. genetic data)
- When transferring to and from Windows, be aware of EOL and encoding (slide 70)!
- FAT32 (commonly used for USB flash disks) has limits to maximal file size (~ 4 GB) and range of characters allowed in file name is limited (slide 43), it doesn't support UNIX permissions (breaks executability of scripts)
  - Avoid large single files (or use archives to split large files)
  - In file names keep only alphanumerical characters, dots and underscores (omit spaces and accented characters)
  - Use archives to keep needed permissions (e.g. executability of scripts), see further

File names

- Space serves as separator of parameters
- Linux allows 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 single 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... Or rm '*'
```

- 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)
```

Task: Try everything on this slide, also with different file names and characters.

Types of files

```
ls -la ~
-rw-r--r-- 1 vojta users 1685 Feb 25  2019 .bashrc
drwxr-xr-x 1 vojta users 1040 Jan  8 09:08 bin
lrwxrwxrwx 1 vojta users   28 Jan 11 18:00 .lyxpipe.in -> /tmp/kile-...
...
```

- Regular file — ordinary file, marked by dash ()
- Directory — in UNIX special type of file, marked by `d`
- Symbolic link (symlink, “soft link”) — points to another place, marked by `l`, slide 60
- Hard link — just another name for existing file, no special symbol, slide 60
- 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 108
- Socket — for communication among processes, also bidirectional, available on network

Links

- Soft links — like links on the web — short-cut to another place
  - When we delete link, nothing happens, when target, non-working link remains

```
ln -s source target # Source will point to target
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`
  - If any one of the two files is deleted, the second remains to be fully working

```
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 7298 21. jan 16.43 .bash_history # One link
-rw-r--r-- 2 vojta users 2707 29. nov 16.21 .bashrc # Same file as below
-rw-r--r-- 2 vojta users 2707 29. nov 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 (numbers): GID and UID
- Commands `chown` to change owner requires root privileges
- Commands `chgrp` to change group commonly requires root privileges — user has to be member of particular group to be able to change ownership to it (if not, `root` must do it)
- Information about users and groups and their IDs are in `/etc/group` and `/etc/passwd`
- Ownership (and permissions, slide 63) are important especially on servers with plenty of users
- It is not possible to add particular permissions for particular user on one file — there must be special group or ACL must be used (slide 66)

Change owner and/or group

```
ls -l # Shows also owner and group (columns 3 and 4):
drwxr-xr-x 1 vojta users    80  5. jan 16.12 linuxcourse
drwxr-xr-x 1 vojta users  1648 31. jan 10.15 presentation
-rw-r--r-- 1 vojta users  1944  5. jan 15.18 README
drwxr-xr-x 1 vojta users   822 29. jan 10.12 scripts_data
-rwxr-xr-x 1 vojta users  1126  5. jan 15.22 web_update.sh
l # Common alias for 'ls -l' or 'ls -la' (according to distribution)
ll # Common alias for 'ls -l' or 'ls -la' (according to distribution)
id # Display UID and GIDs of current user
# New owner or group can be defined as name or ID
chown newowner:newgroup files # Change owner and group
chown -R newowner files # Recursively (-R) change owner
chgrp -R newgroup files # Recursively (-R) change group
chown --help # Or 'man chown' for more options
chgrp --help # Or 'man chgrp' for more options
```

- Equally important is to have correct permissions (especially on server) — next slides

File and directory permissions

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

- `rwxrw-r--` — 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 66)
- `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`, e.g. `chmod +x` or `chmod u+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. jan 09.23 .bash_history
# Owner can write read and write the file, others read; 644:
-rw-r--r--   1 vojta users    2707 29. nov 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. jan 09.33 .lyxpipe.in ->
  /tmp/kde-vojta/kilemj7d3E/.lyxpipe.in
# Executable (application) - everyone can launch it, but only owner can
# write into the file (change or delete); 755:
-rwxr-xr-x   1 vojta users    2187 27. nov 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 ## Colored 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 other action...
chmod 777 * # All permissions for everyone on everything (no recursive)
chmod --help # Or 'man chmod' for more options
```

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 Btrfs, XFS and ZFS
- 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 + after permissions — next slide) sometimes do not show correct result and permissions may work unexpectedly
- Important especially in network environment with many users
- If intensively used, `ls -l` sometimes doesn't show correct permissions
- If not in use on server (like e.g. on CESNET data storage), relatively high number of groups is required to be able to correctly setup sharing permissions

ACL examples

```
getfacl FILE # get ACL for FILE:
# file: dokumenty
# owner: zeisek # Correct
# group: zeisek # Correct
user::rwx        # Correct
user:nasik:r-x   # This is not seen from 'ls -l' output below!
group::r-x       # This group has only one memebr, this is fine
mask::r-x
other::---       # Correct
ls -l FILE # Compare this and previous output (This might be wrong!):
drwxr-x---+  2 zeisek zeisek     6 17. zář 20.40 dokumenty/
setfacl -m u/g:USER/GROUP:r/w/x FILE # Add for USER/GROUP r/w/x right
# E.g. recursively add read permission to user 'arabidopsis_data' to
# folder 'dokumenty/arabidopsis' (no extra group is required)
setfacl -mR u:arabidopsis_data:r dokumenty/arabidopsis
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
- ACL can be used to set default permissions/ownership for new files in 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. jan 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
```

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), UTF-16 for Asian languages
  - Windows: win-cp-125X (variants according to region)
  - Older UNIX: ISO-8859-X (variants according to region)
  - Other much less common (historical) types
  - Important mainly for accented characters
- Text editors can usually open any encoding, but automatic detection commonly fails — set it manually

Command line

Launching commands and scripts

- Parameters of commands are separated by space and preceded by one or two dash(es)
- Parameter -h or --`help` usually gives help for particular command
- Getting help with `man` command
  - `man somecommand`
  - Arrows to move up and down, q to quit
  - Type / and then type text to search 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 (slide 102) — actual directory isn't
  - If the script is is current directory, use `./script.sh` or full path
- Custom scripts must have execute permission (`chmod +x script.sh`)

macOS and Homebrew

- macOS contains outdated versions of many command line utilities with limited functionality comparing to what we are going to use (what is available in modern Linux distributions)
- Several projects provide Linux style way of installation and update of various (not only) command line tools, probably the best is Homebrew
- Homebrew contains also plenty of scientific packages, there is also specialized similar source for bioinformatics
- Tools installed via Homebrew are installed into `/usr/local` not to interact with system packages
- Derived project is Linuxbrew (works also on Windows subsystem for Linux) useful especially for installation of special (scientific) software unavailable in main Linux repositories (software resources)

Working with Homebrew

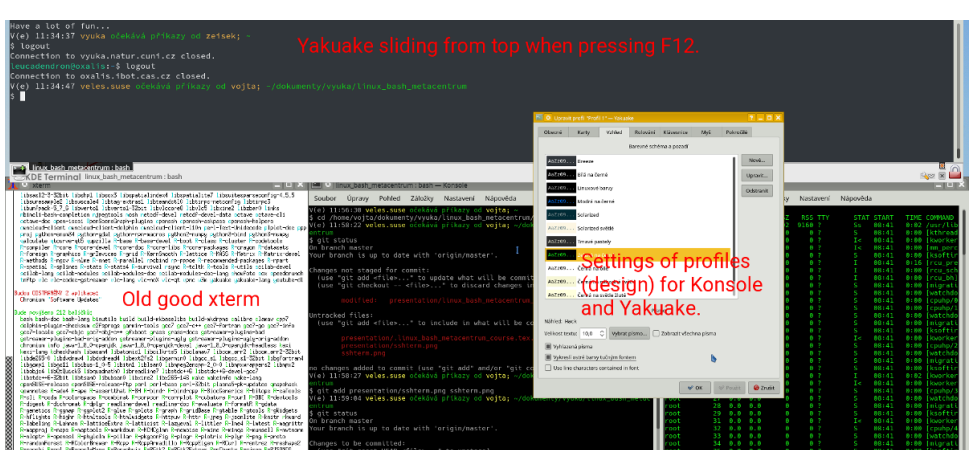
```
# Install compilation tools
xcode-select --install
# Install Homebrew
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/
  Homebrew/install/master/install)"
brew help # Basic help
# Install updated basic UNIX tools
brew install coreutils gnu-sed gawk grep bash gcc make wget dos2unix
brew list # List of installed packages (brew formulae)
brew info FORMULA # Information about particular formula
brew search KEYWORD # Search for applications
brew update # Update Homebrew
brew upgrade # Update all packages installed by Homebrew
brew uninstall FORMULA # Remove Homebrew package (formula)
brew cleanup # Cleaning after uninstallation
# Completely remove Homebrew (after uninstallation of all formulae)
ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/
  install/master/uninstall)"
```

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)

The command line can has various look and feel...

Change colors, font size, etc. for your terminal to like it more and work comfortably



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 crashes
- Sometimes number of connections to the server 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`)

Tmux

- More advanced (but not so common) alternative to `screen`

```
# Start new tmux session
tmux
# Or name the new session (useful if there are more sessions)
tmux new -s SomeName
# Detach from the session by Ctrl+B, D
# List sessions on the server
tmux ls
# Attach to existing session by name
tmux attach-session -t SomeName
# Attach to existing session by its number
tmux attach-session -t 0
# Cancel running session by Ctrl+D or exit
```

- Get help by `Ctrl+B, ?` (Q to quit)
- Split window by `Ctrl+B, %` and navigate between them by `Ctrl+B, left/right arrow`

Login to remote server

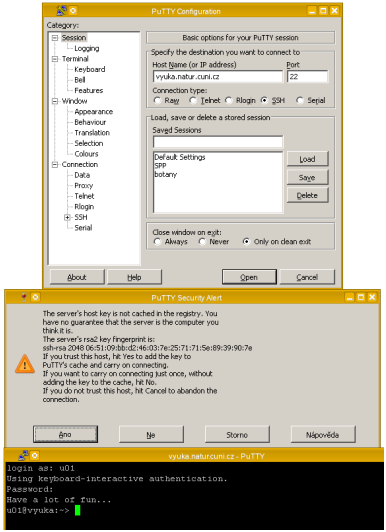
SSH — secure shell — encrypted connection

```
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
```

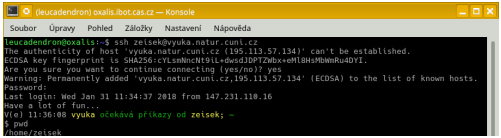
- Our toy server: user names from `cu01` to `cu20`

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

- If fingerprint key changes, ssh complains a lot — possible man in the middle attack
- From Windows use Putty



SSH and screen practice



Tasks

1. Login via SSH to `vyuka.natur.cuni.cz` and launch `screen`.
2. Run commands `pwd`, `whoami` and `ls -la`. What do they show?
3. Detach from screen and logout from the server.
4. Login again to the server. Is it same or different process? Why?
5. Reattach to the screen — same state as before logout from server.
6. Open new screen and practice tasks with file names (slide 58).
7. Detach and re-attach to both sreens.
8. Close all screen sessions and logout from the server.

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 systems)
- **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 (e.g. SSH) — user can have different settings for local and remote work
- `/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 (popular examples)

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 its recursive behavior
```

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 by e.g.
source ~/.bashrc # to reload BASH settings
# 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 --color=auto' # Enable color in grep
# Always human readable output of df (disk free)
alias df='df -h'
# Easier history listing
alias his="history | grep" # Use e.g. 'his ls' to list last 'ls' usage
# Add aliases pointing to software installed outside PATH, ...
```

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
```

Tasks on the remote server I

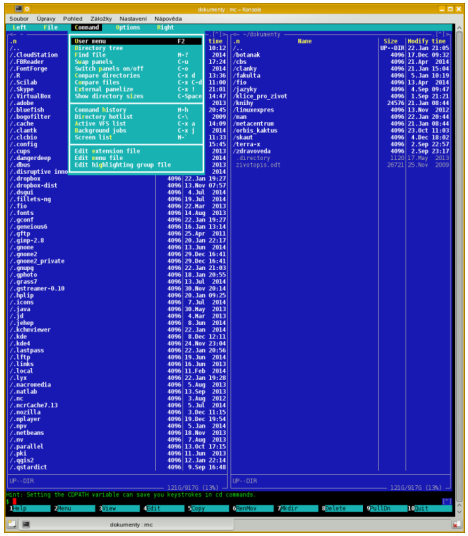
1. Login via SSH to `vyuka.natur.cuni.cz`.
2. Get your path by `pwd`.
3. Go to `/home/scripts_data` (with `cd`) and explore its content (`ls`).
4. List permissions in `/home/scripts_data` (slide 63). What do they show? What can you do with the content?
5. How much space does `/home/scripts_data` consume?
6. Go back to home directory (by `cd`).
7. Create new directory in your home directory (`mkdir`).
8. Copy content of `/home/scripts_data` into your newly created directory (`cp`).
9. Rename that directory with scripts and data using `mv` to any custom name. Who is owner of the files in origin location and in new location? Why?
10. Explore your home directory and its content by command `ls` and `tree` and some files by command `file`. Which hidden files and directories are there? What could it be?

Tasks on the remote server II

11. Change permissions of the files so that only you can read, write and execute them (`chmod`).
12. Create other directory, see it and then remove (`rmdir`).
13. Can you access directories of another users? Why? If yes, what are your permissions there? Explain it.
14. What are some permissions in `/`? Why?
15. Create directory in your home directory and share it with another user so she/he can write there anything (using e.g. `touch somefile` or `mkdir somedirectory`) (work e.g. in pairs). Use everywhere as restricted permissions as possible. Can you figure out solution with or without ACL (slide 66)?
16. Practice moving between `/home/scripts_data` and your home directory. Use `cd` and TAB.
17. Within `/home/scripts_data` list by single command only `jpg` and `txt` files.

Midnight Commander

- mc to launch MC
- Move (F6), copy (F5), delete (F8), files/directories
- Connect to SSH/(S)FTP, ...
- Can be used with mouse
- Edit (F4) or view (F3) text files
- F2 for quick menu
- F9 for top menu
- And much more...
- Impossible to live without it :-)
- Task: Which of the previous tasks can you solve with it? Try it.



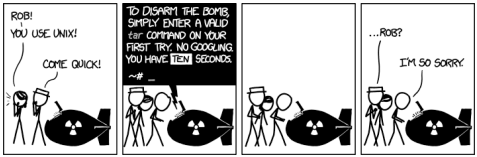
Compressing files into archives

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

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



https://xkcd.com/1168/

Tasks with archives

1. Compress (and decompress) text file `Oxalis_HybSeq_nrDNA_selection_alignment.fasta` (target enrichment sequencing data of South African *Oxalis*) from `scripts_data` with various compressing tools.
2. Compare sizes of original file and compressed outputs.
3. Compress (and decompress) all `foto_oxalis_*.jpg` (*Oxalis* photos) from `scripts_data` with various compressing tools.
4. Compare sizes of original files and compressed outputs.
5. Which compression tool seems to be the best? In terms of compressing ratio and time needed for compression.
6. Is more effective compression of text files or images? Why?
7. Why is even plain `tar` (without compression, it requires `gzip`, `bzip2` or `lzma` to add compression) useful with FAT disks?
8. Search the Internet to find out how to unpack `rar` or `arj` archives from command line.

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 "locate" 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 # "exit" 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 wildcards (`*`, `?` and `[...]`), see globing (slide 97)
- `-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 (apply some of them to the toy data)

```
# Find in /home/$USER/ all JPG files containing string "oxalis"
find /home/$USER/ -name "*oxalis*.jpg" -print
# Find in scripts_data all JPG files and resize them to 1000x1000 px
find scripts_data -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 | xargs grep -nH DNA # Or
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
# Change permissions of all files within "files" directory to 640
find files/ -type f -exec chmod 640 '{}' \;
# Find all executable files within current directory and list them
find . -executable -type f -print
find doc/ -type d -empty -execdir rmdir '{}' \; # Delete empty dirs
man find # See another options. Much more...
```

Searching tasks

1. Use `locate` to find file `long_text.txt` on the server. Is the output absolutely correct? If not, why?
2. Where is executable of `mc`? Why can be such information useful? Where are its libraries, data, configuration and manual?
3. Which software is related to keywords `permission` and `compress` (use e.g. `apropos`)? Use `man` to explore some of them.

Tasks with `find`

1. Find all `*.vcf.gz` files in whole `/home`. Why do you get errors for some directories?
2. Compress, see and decompress all shell scripts in your home directory.
3. Change permissions of all content of your home directory so that no one else can access it. Consider hidden files, directories and scripts.
4. List all directories in `/etc` up to second level.

BASH globing and wildcards

- BASH itself doesn't recognize regular expressions — its wildcards have some of functions of regular expressions (from slide 164) 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 literally asterisk * and not "any number of any characters" as usually)
- For details see `man 7 glob` and `man 7 regex`
- This is useful e.g. to list only subset of files or to provide file selection as input for some command

Brace expansion and quotes

```
1 echo a{p,c,d,b}e # ape ace ade abe - all combinations
2 echo {a,b,c}{d,e,f} # ad ae af bd be bf cd ce cf - all combinations
3 ls *.{jpg,jpeg,png} # expansion to *.jpg *.jpeg *.png, same as
4 ls *.jpg *.jpeg *.png
5 ls scripts_data/*_R[12].fastq.bz2
```

- Text in single quotes (`'...'`) preserves the literal value of each character within the quotes
- Text in double quotes (`"..."`) preserves the literal value of all characters within the quotes, with the exception of dollar (`$`), back tick (`` ` ``) and back slash (`\`)
- A double quote may be quoted within double quotes by preceding it with a backslash (`\"` means literally "double quote")
- Text between back ticks (`` `...` ``) or within `$(...)` will be evaluated and then used as command or argument (see next slide for examples)
  - Syntax with back ticks is deprecated, keep using `$(...)` instead
- This is important when handling file names with non-Latin characters, when working with variables (from slide 100), printing various information within scripts (e.g. slide 104), etc.

Variables in BASH

- Variables contain various information (where to look for the executable programs, name of the computer, various settings, input files, ...)
- Can be local (within a script for some temporal purpose) or global — available for all processes (and users)
- 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 (slide 102)
  - SHELL — shell in use (bash or something else)
  - USER — user name
  - And many more, commonly specific for particular server
- Names start with $, e.g. $HOME, but declaration is without $, e.g. MYVAR='XXX'

BASH expressions

```
# Many operands have special meaning in BASH - must be escaped
echo $((5 < 7)) # Is 5 smaller than 7? TRUE (1)
echo $((3 > 4)) # Is 3 greater than 4? FALSE (0)
echo $((16 / 3)) # Division (without decimal part)
echo $((12 % 5)) # What remains after aritmetic division
x=$((1 + 6)) # Result will be in 'x' - save output into variable $x
echo $x # See content of $x
x=1 # Set x to 1
y=$x+1 # Will this add 1? Why?
echo $y # See result
y=$(($x + 1)) # Will this work? Why?
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`)

Work with variables

```
printenv # Get all environmental variables and their values
export -p # Get all exported variables and their values
declare -p # Get all declared 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
LISTFILES="$(ls -1)" # Get output of command 'ls -1' into the variable
echo "$LISTFILES" # See content of the variable LISTFILES
export GREP_OPTIONS='--color=auto' # Colored grep (see further)
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
- Computing clusters (like MetaCentrum) use to have special command (e.g. `module`) to load particular software (including particular version) by extending user's `$PATH`

```
# 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 $X # Get value of the variable
10 # It works
echo "$((1 + $X))" # Sum of 1 and variable $X
# Output some command into variable
Y=$(command) # Set as variable output of command
echo $Y # Y will contain output of command
# "command" from previous lines can be e.g.
cat somefile.txt # Read content of somefile.txt; or
find . -name "*.txt" # Save list of matching files
WORKDIR=$(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 $($A) # Same as `...`, this is now recommended way, `...` is legacy
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, ... — next slide)
- 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 in $(...)
echo $H # abcvojtade # To add output of command into the variable
I='...' # Needed if $I should contain spaces, quotes, `, $, ...
```

Chaining commands

- Key feature of BASH — work with individual commands as with Lego to get new functionality
- & — 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

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
```

- Task: Practice chaining and redirects from previous and this slide. What is it good for?

Redirects and pipes

- `/dev/null` — "black hole" — can discard anything

```
command 2> /dev/null # Discard only errors (note "2" for errors only)
command > /dev/null # Discard all output (no logging or onscreen output)
```

- `/dev/stdin` — standard input
  - Typically keyboard
  - In case application reads files, not from standard input:

```
echo "Žluťoučký kůň úpěl ďábelské ó" | 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 # Errors go to screen (typically), not to log
```

- `/dev/stderr` — standard error output
  - Typically screen or log file, right place to send errors to:

```
echo "error" > /dev/stderr
command 2> /dev/stderr # Errors go to standard error log
```

- `|` — pipe — basic redirecting method — standard output of one command to standard input of another command

Redirects of standard input and output I

Tasks with redirects of standard input/output

- Try everything on this and following slide. Is it clear what happens?
- Be sure to understand the redirects and differences among the variants.
- Try to figure out some other example regarding your practical needs.

```
# 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 # See result 'cat directory_listing.txt'
# We are outputting 'ls -lh' to 'awk' and not to screen, printing only
# selected columns, and parsing with 'column -t' for tabular display
ls -lh # Compare outputs of the 3 commands starting with 'ls -lh'
ls -lh | awk '{ print $NF, " ", $5}' # See further more info about awk
ls -lh | awk '{ print $NF, " ", $5}' | column -t
```

Redirects of standard input and output II

Try everything and be sure to understand

```
# 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.log 2>&1
cat outputfile # See result
# Compare with following. What is the difference?
{ commandX; ls; } > outputfile.log # Where does error and output go?
# Compare two following commands. What is the difference?
{ commandX; ls; } > outputfile.log 2>&1 # Inspect outputfile.log
{ commandX; ls; } >> outputfile.log 2>&1 # Inspect outputfile.log
# 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
```

Examples of redirects and pipes when working with molecular data

```
# Extract and sort depth of coverage (how many times was each position
# sequenced) in genomic VCF with multiple individuals
zcat arabidopsis.vcf.gz | grep -o "DP=[0-9]\+" | sort | less
# Convert DNA sequence from FASTQ to FASTA (two of many options)
# (discard FASTQ quality scores and keep only the sequence and its name)
bzcat Oxalis_hirta_R1.fastq.bz2 | sed -n '1~4s/^@/>/p;2~4p' > \
  Oxalis_hirta_R1.fasta
bzcat Oxalis_hirta_R2.fastq.bz2 | awk '{if(NR%4==1)
  {printf(">%s\n",substr($0,2));} else if(NR%4==2) print;}' > \
  Oxalis_hirta_R2.fasta
# See both sequences in one view
less Oxalis_hirta_R{1,2}.fasta
# Mapping of trimmed Illumina FASTQ reads to reference FASTA sequence
# and creation of BAM containing the alignment (reference and mapped
# reads, one BAM for each sample)
bwa mem reference.fasta input_R1.fastq input_R2.fastq | samtools view \
  -bu | samtools sort -l 9 -o mapped.bam
# How many FASTQ reads are there in the file
cat file.fastq | echo "$(("$(wc -l)" / 4))"
```

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, ...)
lspci | grep -i vga # Get information about graphical 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 ux # All user's processes
ps aux # All processes
# kill (terminate) process by name or process ID (PID)
# Find which PID has application to terminate
pgrep geany
14639
# Kill (stop by SIGTERM) selected application according to above PID
kill -SIGTERM 14639 # SIGTERM is "nice" termination, SIGKILL "brutal"
kill -SIGTERM $(pgrep geany) # Two above commands in one step; note $()
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
```

Tasks regarding hardware, information and processes

1. Change your password on the remote server and/or virtual machine.
2. Who is and was recently logged in to the remote server?
3. Which information about hardware can you get from your local computer (if you are running Linux) and from course server? Why could such information be useful?
   1. Which disks are mounted?
   2. How many memory (RAM) slots are occupied and what is memory size?
4. Find your user ID on local computer and remote server. Do you think this has any implications when copying files from your local computer to server and back?
5. Terminate from command line some running graphical application. What is difference between `kill` and `killall`?
6. Change priority of some running (graphical) application.
7. Which applications are consuming the most resources (CPU, RAM)?

Network protocols I

- Every network communication protocol (e.g. browsing web or Skype) has its own distinct port ("door to access target computer") — it must be opened (by firewall)
- There are plenty of network services
  - Especially file servers and various data storages support more protocols to access the data — according to operating system and/or use case
  - Sometimes it can be tricky to pick up (and configure) the right protocol
- SSH — secure shell — command-line connection to remote server to work there (port 22), slide 79
  - Plenty of other protocols can run over SSH (SFTP, SCP, SSHFS, rsync, ...)
- Telnet — old deprecated insecure predecessor of SSH, never ever use it (port 23), only low-quality electronics still sometimes use it
- FTP — file transfer protocol — outdated, no encryption (port 21), slowly replaced by FTPS or SFTP/SCP
  - Sometimes used only for download, e.g. `ftp://ftp.ebi.ac.uk/pub/` or `ftp://botany.natur.cuni.cz/`
- FTPS — FTP with added connection encryption for higher security (port 21), common

Network protocols II

- SFTP — FTP over SSH — common, secure (port 22), slide 121
- SCP — secure copy — uses SSH, but has restricted possibilities, common, secure (port 22), slide 121
- NFS — network file share/server — very common in UNIX world (commonly used e.g. by CESNET MetaCentrum and data storage — česky), commonly used to permanently connect to network server, share directories, etc. (port 20049), slide 120
- webDAV — file transfer over web (using WWW server) — not so common, but good (port 80 or 443 — same as WWW), slide 120
  - Accompanied by calDav and cardDav to share calendars and address books over the network
  - Used e.g. by ownCloud/nextCloud (also provided by CESNET — česky) — tool to synchronize and share files etc, similar to Dropbox or Google Drive
- SAMBA — UNIX connection to Microsoft network shares (port 5445), slide 120
- 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

Network protocols III

- Messaging protocols like XMPP (Jabber and derived services like Google Talk or Facebook Messenger), IRC, ICQ, Skype, ...
- Databases (if accessible over the Internet), and another special services, use to be available on dedicated ports
- And much more (see `/etc/services`)...
- Port number can be changed in configuration of respective server service
- All firewalls on the way must allow communication on given port — some ports are commonly filtered in restrictive Wi-Fi networks or totalitarian countries

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 (path) 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 wireless 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 packages samba and cifs)
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
# Mounting remote server over SSH (sshfs package must be installed)
sshfs USER@vyuka.natur.cuni.cz:/some/dir /local/mount/point
fusermount -u /mount/point # Disconnect SSHF
# 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/remote.php/webdav/ /local/dir
umount /mount/point # Disconnect CIFS/SAMBA, NFS, webDAV, ...
```

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 e.g. `duplicity`

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

SSH keys

Secure way how to connect to multiple servers with single key and more

- Secure way how to connect to multiple servers via SSH with single (or no) password using asymmetric encryption
- Command `ssh-keyge` generates pair of keys
  - Private key is typically in `~/.ssh/id_*` file(s) (according to cipher)
  - Public key is typically in `~/.ssh/id_*.pub` file(s) (according to selected cipher) and are copied to target servers
  - Private key is unlocked by password and it then allows to login to any server having the public key
  - Having only public key is not enough to login, private key without password is still not enough, but user must be sure private key is kept securely and not lost or stolen
- File `~/.ssh/known_hosts` contain servers you have ever connect to, and their SSH fingerprints (unique IDs) — if this changes, SSH complains a lot as it could be Man-in-the-middle attack
- File `~/.ssh/authorized_keys` contains public keys allowing logging to the machine
- `/.ssh/config` can store various settings for particular servers

Connect to SSH with key

No need to remember password for every server...

```
# Create the key (several options)
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
# 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
```

- SSH keys work in all applications using SSH (SFTP, SSHFS, rsync, many graphical file managers, ...)
- User can have various local passwords on servers, but use single SSH key to connect to them

Network tasks I

1. Create SSH key (if you don't have any), copy the public part to `vyuka.natur.cuni.cz` and connect there with SSH.
   1. What are advantages and disadvantages of having the key with or without password?
   2. Is there any reason for having multiple keys? Can you find any examples?
   3. Which services, protocols can use it? Find some examples.
2. Make in your notebook new empty directory and mount there via SSHFS your home directory on `vyuka.natur.cuni.cz` (or on some other server).
   1. Explore it. What is then path to files on the remote server?
   2. Disconnect the server when done.
3. Use `wget` and `curl` to download this presentation to your notebook (link at slide 8).
   - For `curl` be aware of setting of output — what is default behavior?
4. Use `scp` to copy directory with toy data and scripts from your home directory on `vyuka.natur.cuni.cz` into your notebook.
5. Use `scp` to copy any file from your notebook into your home directory on `vyuka.natur.cuni.cz`.

Network tasks II

6. Use `rsync` to update toy data and scripts in your notebook according to version on `vyuka.natur.cuni.cz` in `/home/scripts_data`.
7. Use `rsync` to update directory with toy data files and scripts in your home directory on `vyuka.natur.cuni.cz` according to version in your notebook.
   - For `rsync` try to use various options, run it several times.
   - Explore help of `rsync`. Can you find there some useful parameters?
8. Get MAC and IP address of your notebook. Why can be such information useful?
9. Ping some web server. Is it alive and well reachable?
10. Why can be output of `traceroute` useful when you have problems with network?
11. Mount (explore and then unmount) your CESNET ownCloud into your notebook. If unsure, consult help (česky).
    - It is available for all academics in the Czech Republic, just go to `https://owncloud.cesnet.cz` and login with your institutional credentials
    - Alternatively connect to some other ownCloud/nextCloud/generic webDAV server

Network tasks III

12. What are advantages and disadvantages of mounting of remote servers (SSHFS, etc.) into local directories?
13. Connect to `vyuka.natur.cuni.cz` (or any other server) using `mc` and copy there and back some files. What are advantages and disadvantages of `mc` when compared with usage of `scp` or `rsync`?
14. Find in graphical interface of your computer how to connect via SSH/SFTP/SCP to transfer files. Connect to `vyuka.natur.cuni.cz` or some other server and transfer there and back some files.

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
  - See examples on next slides

GNU Parallel examples I

```
# Convert all images from JPG to PNG
find . -name '*.jpg' -print | 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
find . -name '*.txt' -print | parallel 'cat block_to_be_added.txt >> {}'
# Replace particular text in multiple files with sed and GNU Parallel
find . -name '*.txt' -print | parallel 'sed -i "s/XXX/YYY/g" {}'
# Launch MrBayes for multiple nexus files and create log file (see also
# next slide) with starting and ending date and time
find . -name '*.nexus' -print | parallel 'echo -e "Start: $(date)\n" >
  {}.log && mb {} | tee -a {}.log && echo "End: $(date)" >> {}.log'
```

Tasks

1. Resize using `parallel` photos `foto_oxalis_*.jpg` to 1000x1000 px.
2. Convert all above JPG files to PNG.
3. Use several options how to run `parallel`.

Recording output of commands

- Alternative (commonly more convenient) to redirects of outputs to log file (slide 108) is command `tee`
  - Very useful if redirects would be harder to code
- `tee` can record
  - All output of the software (standard as well as error output)
  - Commands (keys) typed by user — it can be later reused to rerun the application — unique feature useful for certain software

```
# tee (-a for append to existing file) records output of any application
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
- Can be replaced by SystemD timers (more complex, but more versatile)

```
# 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 second day, 4:31
  */15    *     *             *      0,4  # At Sun and Fri every 15 min
# Columns are separated by any number of spaces
```

Text

Various aspects of working with text files

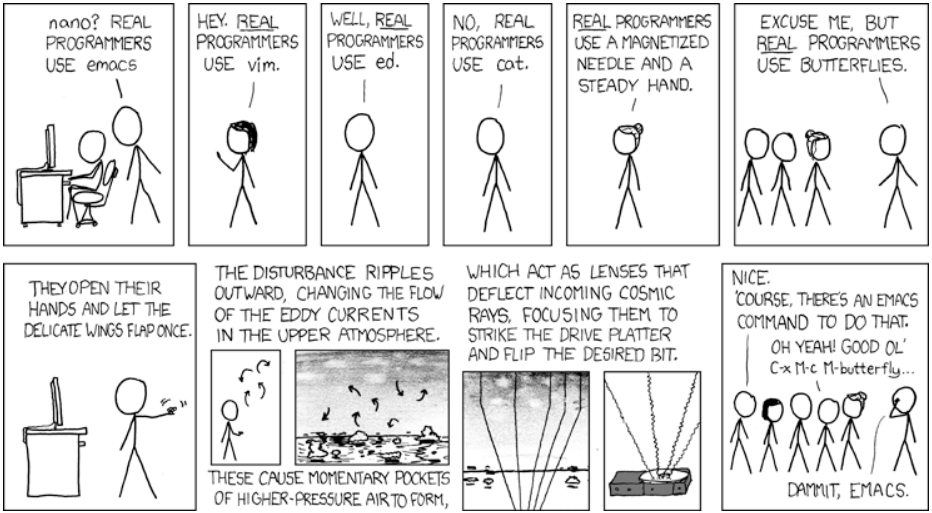
Everything is (text) file

- 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 164)
- Most of the operations are done in stream — per line — everything is very fast and memory efficient

Text tasks

Try all the commands from this chapter. It is one of key BASH features.

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



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
- And more...

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

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 encoding
# 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 encoding to convert
iconv --help # More information about usage
enca -h # See usage of another encoding tool
recode CP1250..UTF-8 textfile # Convert encoding from CP-1250 to UTF-8
recode ../CR-LF textfile # Convert EOL from UNIX to Windows
recode --help # More information about usage; recode is deprecated
```

- macOS 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/macOS will probably fail (because of different EOL)

Read text file

```
cat # Read or join (using redirects) files
    # -n adds line numbers, -v prints non-printable characters like EOL
cat long_text.txt # Print content of text file
cat textfile1 >> textfile2 # Append textfile1 to the end of textfile2
nl long_text.txt # Like cat -n, prints textfile with line numbers
tac textfile # Like cat, but prints lines in reverse order
more long_text.txt # When textfile is long, prints screen by screen
                   # (space for next screen, q to quit)
less long_text.txt # 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)
fmt long_text.txt # 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 long_text.txt # lines, words and bytes in text file
                 # wc -l for only lines, -m for characters, -w for words
mc # It has text viewer (F3) and editor (F4)
```

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 for case insensitive
grep -q ... # quiet - no output (only T/F) - good for testing in scripts
grep -ls user /etc/* # -l print files with pattern, -s suppress errors
grep "longer text" textfile # Extract whole phrase
```

- Grep supports regular expressions, slide 164

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 -f 2 cut_awk_test_file.tsv) <(cut -f 1 cut_awk_test_file.tsv)
ls -l | column -t # Reformate input as table (compare with 'ls -l')
```

Examples of usage of head, grep and cut with our data

```
head long_text.txt # Beginning of the file
# Extract names of FASTA sequences
grep "^>" Oxalis_HybSeq_nrDNA_selection_alignment.fasta
# Extract every FASTA sequence containing "AAAAAA" and name of
# respective FASTA sequence
grep -n -B 1 AAAAAA Oxalis_HybSeq_nrDNA_selection_alignment.fasta
# Get column of pairwise identities of sequences exported as TSV from
# Geneious - discard header (1st line, print from 2nd line) and extract
# second column (separated by TAB)
tail -n+2 cut_awk_test_file.tsv | cut -f 2 | less
# Number of occurrences of word "Gregor" in file long_text.txt
grep -o Gregor long_text.txt | wc -l # Or just
grep -c Gregor long_text.txt
ls -1 *.sh | wc -l # How many BASH script are in current directory
# Extract from FASTA file only sequence (discard sequence names) and
# save result in seq.txt
grep -v "^>" Oxalis_HybSeq_nrDNA_selection_alignment.fasta > seq.txt
# Save names of oxalis* JPG files into list_oxalis_photo.txt and see it
ls -1 *oxalis*.jpg > list_oxalis_photo.txt && cat list_oxalis_photo.txt
```

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 (easily) do — 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 164
- For more information see manuals
  https://www.gnu.org/software/gawk/manual/,
  https://en.wikibooks.org/wiki/An_Awk_Primer and
  https://www.grymoire.com/Unix/Awk.html

```
awk 'regexp { commands parameters }' file # General syntax
# Print last column (separated by tab, built-in variable $NF)
awk '{print $NF}' cut_awk_test_file.tsv
```

AWK examples I

```
# Select 2nd column ($2; separated by tab)
awk '{print $2}' cut_awk_test_file.tsv
# Print columns 3 and 2 (in this order)
awk '{print $3, $2}' cut_awk_test_file.tsv
# Get column 5 and 1 (in this order, separated by ":") from /etc/passwd
# (only lines containing "home") print ", username:" between the columns.
# Note usage of commas and consequences to output.
grep home /etc/passwd | awk -F ':' '{print $5 ", username:", $1}'
# Separate columns by TAB, /^d/ for lines starting with "d" (only dirs)
ls -l | awk '/^d/ { print $8 "\t" $3 }'
# Print on even lines ">", former column 1, new line, former column 2
# 2 columns into 2 lines (create FASTA from tabular record)
awk '{print ">"$1"\n"$2}' awk_test_file.tab | less -S
# Print field 1, TAB (\t), length of field 2, TAB and field 2
awk '{print $1"\t"length($2)"\t"$2}' awk_test_file.tab
# If field (column) 2 contains exactly 100.0%, print whole line ($0)
awk '{if($2=="100.0%"){print $0}}' cut_awk_test_file.tsv
# Field 1 is numeric (less then 5 digits) - add leading zeroes
awk '{printf "%05d\n", $1;}' awk_test_file.tab
```

AWK examples II

```
# As previous, but add leading zeroes to field 1 and print whole line
awk '{$1=sprintf("%05d", $1); print $0}' awk_test_file.tab
# Field 6 is numeric, select lines where field 6 is higher than 200
awk -F '\t' '$6>200' 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}' cut_awk_test_file.tsv
# Precede each line by its line number for all files together, with TAB
# (i.e. print line number (NR) and then whole original line ($0))
awk '{print NR "\t" $0}' diff_test_file_*
# substitute "a" with "XXX" ONLY for lines which contain "The"
awk '/The/{gsub(/a/, "XXX")}; 1' diff_test_file_1.txt
# For every 4th line starting from line 2 of FASTQ file (from line 2
# every 4th line contains the DNA sequence) print its length (bzcat
# prints content of file compressed by bzip2)
bzcat Oxalis_hirta_R1.fastq.bz2 | awk 'NR%4==2{ print length($0) }'
# If sequence is longer that 500 bp (length of field 2), print its name
# (field 1) like this "Seq. name: TAB the sequence name (ID)"
awk '{if(length($2)>500){print "Seq. name:\t" $1}}' awk_test_file.tab
```

Sorting

```
sort file # Sorting is influenced by locale setting (e.g. Czech "ch")
LC_ALL=C sort ... # To force use of English locale use
sort -d textfile # Take into account only spaces and alphanumerical
                 # characters (ignore any other)
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 (repeated) entries
# Extract only unique sequences from FASTQ
bzcat Oxalis_hirta_R1.fastq.bz2 | awk 'NR%4==2{print $0}' | sort -u
sort -b textfile # Ignore leading blanks (space on beginning of line)
sort -k 2 -n cut_awk_test_file.tsv # Sort according to 2nd field
uniq textfile # Filters following identical lines - only unique are
              # printed (to get unique lines from whole file, sort it)
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]" < diff_test_file_1.txt > outputtextfile.txt
# Alternative (easier reading) of previous command:
cat diff_test_file_1.txt | tr "[:upper:]" "[:lower:]" > outputtextfile
# Replace all new lines (line breaks) by TABs
cat diff_test_file_1 | 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 164 (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)
- macOS has old outdated versions of `grep`, `sed` and other tools (richness of regular expressions is poor) — use versions from Homebrew (slide 73) or search Internet 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)
- See manuals `https://www.gnu.org/software/sed/manual/` and `https://www.grymoire.com/Unix/sed.html`
- Various parameters, modificators, operators can be combined...

Sed examples I

```
sed 'operator/FindToReplace/Replace/modificator' textfile > newtextfile
# Search and replace ("s") all occurrences ("g") of "find" by "replace"
sed 's/find/replace/g' textfile
# Replace third occurrence of pattern on every line
sed 's/pattern/Replace/3' # 's/.../.../' replace only third 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
# Convert sequences in tabular format into FASTA (place ">" to the
# beginning of the line, replace TAB "\t" by newline "\n")
sed 's/^/>/' awk_test_file.tab | sed 's/\t/\n/' > seq.fasta
# Convert FASTQ sequences into FASTA (on every 4th line, starting with
# line 1 replace "@" by ">", print every 4th line, starting by line 2)
bzcat Oxalis_hirta_R1.fastq.bz2 | sed -n '1~4s/^@/>/p;2~4p' > seq.fasta
sed -i 's/find/replace/g' directory/* # Process all files in directory
```

Sed examples II

```
# 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
\Number # Use remembered expression (numbered from one: \1, \2, \3, ...)
# Take output of ls -l and replace value of $USER by "$USER-RULEZZZ"
ls -l | sed "s/\($USER\)/\1-RULEZZZ/g" # Note " to use the 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'
head long_text.txt | sed '/^$/d' # Delete blank (empty) lines
head long_text.txt | sed '6d' # Delete 6th line
sed 's/ *$//' # Delete extra spaces on the end of lines
sed '/GUI/d' # Delete all whole lines containing "GUI"
sed 's/GUI//g' # Delete all occurrences of "GUI" (not whole lines)
sed '4 i\Linux is great.' diff_test_file_1.txt # Insert to 4th line
sed '3 a\Linux is great.' diff_test_file_1.txt # Insert after 3rd line
```

Sed examples III

```
# Insert text to the beginning of the 3rd line (compare with previous)
# "^" is beginning of line, $ end ('$/...' last line)
sed '3s/^/INSERT/' diff_test_file_1.txt
# 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)
ls -l # Compare to the previous command, explain behavior of sed pattern
# Escaping - replace dot by comma (dot means any single character)
sed 's/\./,/g' diff_test_file_1.txt # \ escapes following character
sed 's/./,/g' diff_test_file_1.txt # Compare with the previous example
# Replace any of characters within [...] by some pattern
sed 's/[abcd]/X/g' diff_test_file_1.txt # Compare with reverse case:
sed 's/[^abcd]/X/g' diff_test_file_1.txt # "[^...]" means anything else
sed -i ... file.txt # In-place editing - file is edited, no output to
                    # standard output (no need for redirects and pipes)
ls -l | sed 's/[0-9]\{4,\}/BIG!/' # Replace 4 or more digits by "BIG!"
sed -E ... ; sed -r ... # Use extended regular expressions (see further)
```

Joining

- Generally, most of tools work per-line, `paste` appends columns (slide 142)
- 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...
```

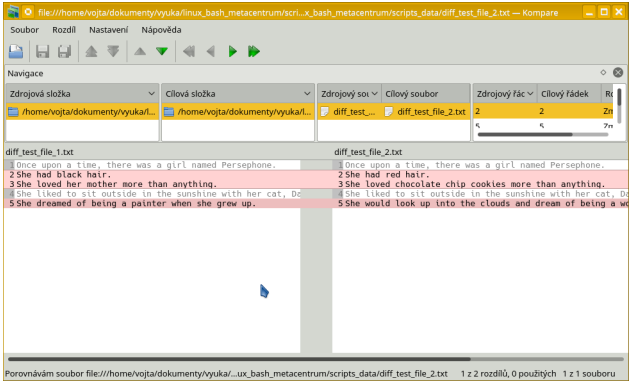
Variants of basic commands for processing of compressed text files

- Many tools are able to directly handle files (i.e. single text files) compressed by `gzip`
- For files compressed by `gzip` use `zcat` (instead of `cat`), `zdiff` (instead of `diff`), `zegrep` (instead of `egrep`), `zfgrep` (instead of `fgrep`), `zless` (instead of `less`), `zmore` (instead of `more`), ...
- For files compressed by `bzip2` use `bzcat`, `bzdiff`, `bzegrep`, `bzfgrep`, `bzless`, `bzmore`, ...
- For files compressed by `lzma` use `lzcat`, `lzdiff`, `lzegrep`, `lzfgrep`, `lzgrep`, `lzless`, `lzmore`, ...
- Sometimes these variants are used automatically when user works with compressed file
- `mc` can also do the job

```
# Extract lines with grep
zgrep FORMAT arabidopsis.vcf.gz
# Display file compressed by bzip2
bzless Oxalis_hirta_R2.fastq.bz2
# Get end of the file
bzcat Oxalis_hirta_R2.fastq.bz2 | tail
```

Comparisons

- Graphically compare two files by e.g. Kompare, DiffMerge, Meld, ...
- Most common is usage of GNU diffutils (next slide), see manual
  `https://www.gnu.org/software/diffutils/manual/`
- See also guide with examples, česky příklady na diff a porovnání dvou textových souborů



comm and diff

```
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
# Don't show 2nd column (similarly -1, -3)
comm -2 diff_test_file_1.txt diff_test_file_2.txt
# Show differences between text files
diff diff_test_file_1.txt diff_test_file_2.txt
  # 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 diff_test_file_1.txt diff_test_file_2.txt # More simple output
diff -c diff_test_file_1.txt diff_test_file_2.txt # Show context
diff -u diff_test_file_1.txt diff_test_file_2.txt # Better, most common
diff -y diff_test_file_1.txt diff_test_file_2.txt # In two columns
colordiff # Same usage and parameters as previous, colored output
colordiff -u diff_test_file_1.txt diff_test_file_2.txt # Most common
```

diff and patch

- Users of `vim` (slide 161) can display the diff in `vim`, or use `vimdiff`
- In vimdiff swap between panes by `Ctrl+W+W`
  - Can open more files, works like standard `vim` (from slide 161), individual files can be saved

```
# Display diff in vim
diff -u diff_test_file_1.txt diff_test_file_2.txt | view -
# vimdiff can show more colors, launches vim (exit by <ESC>:q! Enter)
vimdiff diff_test_file_1.txt diff_test_file_2.txt
```

- Saves difference between two files — it can be later used as template to modify original file
- Single patch file can contain changes from multiple files

```
# Create the diff file
diff -u diff_test_file_1.txt diff_test_file_2.txt > difference.patch
# Apply the patch
patch < difference.patch # or
patch diff_test_file_1.txt difference.patch
```

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 138)
- 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)
```

- Task: Run `vimtutor` and follow instructions there.

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/`, `https://tuhdo.github.io/emacs-tutor.html` and `http://www.jesshamrick.com/2012/09/10/absolute-beginners-guide-to-emacs/`
- The most common is **Vim**
  - See `https://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...
- If regularly working in command line, master any command line editor

Emacs basics

- Feature-rich ecosystem — not only text editor, also file manager, debugger, integrated development environment, extra plugins are available, ...
- Keyboard commands are noted as e.g. C-x, where "C" stands for (mostly) `Ctrl` key, M-x, where "M" stands mostly for `Alt` key (`Meta`), sometimes for `Win` key

- C-h C-h help (twice `Ctrl` + H)
- C-x C-c quit (X-g for particular buffer, etc.)
- C-x C-f open file
- C-x C-s save file
- C-x C-w save file as
- C-_ undo
- C-s search forwards
- C-r search backwards
- C-`left` move one word left
- C-`right` move one word right
- C-`up` move one paragraph up
- C-`down` move one paragraph down
- M-% search and replace pattern

Vim

Vim has three different working modes

1. **"Normal"** — nothing is displayed in bottom left corner, every key has some meaning (next slide)
   - `i` or `Insert` key to enter *insert mode*, `:` to enter *command mode*
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, exit to normal mode by `Backspace` key (delete ":")

- See documentation `https://vim.fandom.com/wiki/Vim_Tips_Wiki`, `https://www.vim.org/docs.php`; česky `http://www.nti.tul.cz/~satrapa/docs/vim/`
- For interactive learning try `https://vim-adventures.com/` (Task: Play it for a while.:) or command `vimtutor`

Normal and command Vim modes

- **Normal mode**
  - `dd` cut current line
  - `r` replace single character below cursor (type then character to be placed instead of the original one)
  - `v` for selection of text (and then e.g. delete by `d` or copy by `y`)
  - `y` copy selection
  - `x` cut selection (always to clipboard, it can be pasted by `p`)
  - `p` paste
  - *number* to get to line of particular line number
  - `u` to undo last change(s)
  - `/ToSearch` search "ToSearch" (once press `Enter`, `n` for next occurrence, quit with `Esc`)
- **Command mode**
  - `w` write file
  - `q` quit
  - `q!` quit and discard changes
  - `%s/...` to search and replace as in `sed`
  - `syntax on/off` turn syntax highlight on/off

Regular expressions are useful...

- 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 (number) of occurrences of single character (from *n* to *m*)
- `\{n\}` — exactly *n* occurrences
- `\{n,\}` — at least *n* occurrences
- `\` — escape following special character (e.g. `\.` to literally search for dot and not “any single character”)

Regular expressions II

- `|` — either the preceding or following reg exp can be matched (alternation)
- `\(...\)` — remembered 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
- `[[:lower:]]` — lowercase characters, like `[a-z]`
- `[[:print:]]` — printable characters (includes white space)

Regular expressions III

- `[[: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 plus)
- `&` — 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...
- **grep**, **sed** and **vim** require escaping of `+`, `?`, `{`, `}`, `(` and `)` by backslash `\` (e.g. `\+`) — **egrep** (extended version, launched as `grep -E` or `egrep`), **sed** with extended reg exp (`sed -r`) and **perl** not

Regular expressions IV

- Read `https://en.wikibooks.org/wiki/Regular_Expressions`, `https://www.grymoire.com/Unix/Regular.html`, `https://www.regular-expressions.info/`; česky `http://www.nti.tul.cz/~satrapa/docs/regvyr/`, `https://www.root.cz/serialy/regularni-vyrazy/` a `https://www.regularnivyrazy.info/`
- Manuals for Grep, Vim, Sed, Awk, Perl (newer Perl 6 Raku), ...
- See sed examples, slide 150; and next slide
- macOS 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-se` formulae from Homebrew (slide 73)

Grep and sed examples

```
# Extract sequences with at least 5 A bases in line
grep "A\{5,\}" Oxalis_HybSeq_nrDNA_selection_alignment.fasta
# Extract DNA sequence string ATCG or ATGC
grep "ATCG\|ATGC" Oxalis_HybSeq_nrDNA_selection_alignment.fasta
# Get quality of Illumina reads mapping to reference in genomic VCF
zcat arabidopsis.vcf.gz | grep -o "MQ=[[:digit:]]\+"
# How many times there is a direct speech (text between "...")
grep -o '\"[[:upper:]][a-zA-Z0-9,\.\?\! ]\+\"' long_text.txt | wc -l
# Add after dot on the end of the line by extra line break
sed 's/\.$/.\n/' long_text.txt
# Add HTML paragraph tags (<p> and </p>)
sed -e 's/^/<p>/' -e 's/$/<\/p>/' long_text.txt | less
# Make first word of every paragraph bold in HTML (<strong>...</strong>)
sed -e 's/^/<strong>/' -e 's/^[[:graph:]]\+/&<\/strong>/' long_text.txt
# How many times is each word in the text
grep -o "\<[[:alpha:]]\+\>" long_text.txt | sort | uniq -ic | less
# List all Internet web links
grep -o "http[a-zA-Z0-9\.()/:\-]\+" long_text.txt
```

Scripting

Basics of writing scripts in BASH

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:

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

Functions in BASH

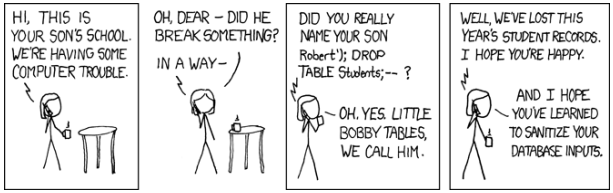
Pieces of code, which can be used repeatedly

```
# Declare new function within script
function MyNewFunction1 {
  echo "Hello, $USER from $(groups) 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 $(("$1" + "$2"))."
  }
# Use it in the script
...
MyNewFunction2 5 8 # For example
...
```

Special variables available in the script (selection)

- These variables can be used within script e.g. to parse arguments provided by the user
- $1, ... (number from 1 up to number of parameters) — individual positional parameters (see further for example)
- $0 — path of the starting script
- $# — number of command-line arguments
- $* — all of the positional parameters, seen as a single word, must be quoted (i.e. "$*")
- $@ — 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 previous command, function, or the script itself
- See more variables...

It is important to check user input...



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, damage data
  - 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

```
#!/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 $(("$1" + "$2"))." # $((calc...))
# "$#" is available every time and contains number of parameters
# (variables) given to the script
echo "Number of parameters is: $#"
# "$*" is available every time and contains all supplied parameters
echo "Those parameters were supplied: $*"
#  "$0" is available every time and contains script path
echo "Path to the scrip is: \"$0\""
echo
exit
```

When done, do:

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

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

Variables will be interactively provided by the user

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

Ensuring user interactively provides correct input (while)

- 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 -r 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 cycles 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... Such check is needed for every variable...
```

Ensuring user interactively provides correct input (function)

```
# Regular expression to check if the provided input is a~number
NUMBER='^[0-9]+$' # From beginning (^) to end ($) only numbers
# Function to read and check user input (this goes to beginning)
function checkinput {
  while :
    do # Star of the body of the cycles
      read -r 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 and continue
        else # What to do if the user did not provide 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
  } # Read variable is in $INPUT
# Replace line 4 of interactive2.sh by (similarly for line 6)
checkinput
V1=$INPUT
```

Provide named parameters

```
#!/bin/bash
# Script has only one parameter ($1) provided
# "case" is evaluating provided parameter and behaving accordingly
case "$1" in
  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
  *) # Any other input
    echo "Wrong option!"
    echo "Usage: 'd' or 'disk' for available disk space or 'u' or"
    echo "  '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` or by wrapping `case` by `while` loop), in the latter variant order of parameters does not have to 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
# From the beginning (^) to the end ($) at least one (+) number ([0-9])
NUMBER='^[0-9]+$'
function usagehelp { # Function to print help - we will use it twice
  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 # "=~" means testing if $1 fits to regular expression in $NUMBER
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 "$2" in
  plus) echo "$(($1 + $3))";;
  minus) echo "$(($1 - $3))";;
  product) echo "$(($1 * $3))";;
  quotient) echo "$(($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 `:`)

```
#!/bin/bash
# All provided values are evaluated in while cycles...
while getopts "hvi:o:a:" INITARGS; do
case "$INITARGS" in # $INITARGS contains the parameters to evaluate
  h|v) # Accept parameters "-h" or "-v" for help
    echo "Usage options..."
    exit # Terminate after providing help
    ;; # End of this option
  i) # Parameter "-i" accepts some value (e.g. "-i inputfile.txt")
    ... # Do some checking etc...
    INPUTFILE="$OPTARG" # $OPTARG always contains value of parameter
    ;; # End of this option
# ...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" # $OPTARG always contains value of parameter
      ;; # End of this option
    a) # Parameter "-a" accepts some value (e.g. "-a X" for number)
      # Check if provided value makes sense (integer between 10 and 300)
      if [[ "$OPTARG" =~ ^[0-9]+$ ]] && [ "$OPTARG" -ge 10 ] &&
        [ "$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"
         echo "  integer ranging from 10 to 300!"
         exit 1
        fi
      ;; # End of this option
# ...continues on following slide...
```

Multiple switches in classical UNIX form (no positional) III

```
# ...continuing from previous slide...
    ?)
      echo "Invalid option(s)!"
      echo "See \"$0 -h\" for usage options."
      exit 1
      ;; # End of this option
    esac
  done # ...the end.
# Check if all required values are provided
if [ -z "$INPUTFILE" ] || [ -z "$OUTPUTFILE" ]; then
  echo "Error! Name of input and/or output file was not provided!"
  echo "See \"$0 -h\" for help usage..."
  exit 1
  fi
if [ -z "$VALUE" ]; then
  echo "Warning! Value for \"-a\" was not provided! Using default (10)."
  VALUE=10
  fi # ...ends on following slide...
```

Multiple switches in classical UNIX form (no positional) IV

```
# ...continuing from previous slide...
# Do the job...
for I in $(seq 1 $VALUE); do # Repeat task number of times ($VALUE x)
  echo -ne "Cycle $I...\r" # Write number of cycle and return cursor to
       # the beginning of the line to overwrite the number in next step
  sleep 1s # Wait 1 second - just for fun ;-)
  # Do the task - append input to the output - note usage of variables
  cat "$INPUTFILE" >> "$OUTPUTFILE" # Do the task - append inp to output
  done
echo -ne "\n" # Reset cursor to new line
echo "Done!"
exit
```

- 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 out.txt -a 50 -i input.txt # Order doesn't matter
```

Simple providing of input file

```
#!/bin/bash
# We expect exactly one parameter
if [ "$#" -ne "1" ]; then
  echo "Error! Exactly one parameter is required!"
  exit 1
  fi
# Verify that file exists and is readable
if [ ! -r "$1" ]; then
  echo "Error! The file provided does not exist or is not readable!"
  exit 1
  fi
# Do the operation with input file..
echo "Size of the file is $(du -sh "$1" | cut -f 1)."
echo "The file has $(wc -l "$1" | cut -d ' ' -f 1) lines."
echo "Making backup of the file $1..."
cp "$1" "$1".bak || { echo "Error! Making backup of $1 failed!"; exit 1; }
echo "Done!"
exit
# Use the script with some text file...
./interactive6.sh long_text.txt
```

If branching (examples are elsewhere)

```
# Basic variant - commands are done only if condition is met
if condition; then
    commands
  fi
# Two branches - when condition is met and when not
if condition; then
    commands1 # condition is TRUE
  else
    commands2 # condition is FALSE - all other cases
  fi
# Join together two (or more) if branches
if condition1; then
    commands1
  elif condition2; 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 globing 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 ]]`
- `-eq` — Equal to
- `-lt` — Less than
- `-gt` — Greater than

Evaluation of conditions II

- `-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 (empty) string
- `-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

Evaluation of conditions III

- `-a` — Performs the AND function (`[ -a ]` or `[ ] && [ ]`)
- `-o` — Performs the OR function (`[ -o ]` or `[ ] || [ ]`)
- Do not confuse globing patterns and regular expressions when using `[[ ]]`
  - Shell globing: `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 parallelisation — 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 if-then branching it is using `case`
- See `raxml_case.sh` for whole script

```
# 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 JPG $JPGF"
  convert $JPGF -resize 100x100 thumbs-$JPGF
  echo "File thumbs-$file created"
  done
for ...; do # Start cycles as you need
  command1 # command1 will be executed in any case
  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 condition and if it is equal to 0 (TRUE)
# the cycle body is launched, repeatedly while the condition is met
while condition; do
    commands
  done
# Like while cycle, but until condition is not equal to zero
until condition; do
  commands
  done
while ...; do # Start cycles as you need
  commands...
  if [condition]; then # If something happens
    break; fi # End up the cycles and continue by following commands
while read TEXTLINE; do # Run cycles on text file
  commands... # TEXTLINE contains in each cycle one line of the file
  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
```

What is worth of scripting

- **Any task you need to repeat from time to time requiring more commands** — almost anything :-)
  - Take advantage of usage of loops, conditions, variables, ...
- Processing of multiple files
  - Various manipulations (conversions, ...)
  - Parsing (e.g. extracting information from multiple CSV)
- Repeated running of some analysis (requiring multiple steps)
  - Keep settings and all preparing and post-processing steps
- Anything you do often and it requires non-trivial set of commands
- And much more...

Start scripting, coding

Now you should have enough knowledge to start writing scripts to do such tasks. Start with something simple, solving some your practical need (e.g. for MetaCentrum, from slide 220). You'll gather experience and improve.

Scripting tasks

1. Write short script (or at least concept how it should work) to print file name of JPG file and its dimensions in px. Use e.g. `file` or ImageMagick `identify` to get the dimensions.
2. Write short script (or at least concept how it should work) taking as input arguments file name and taxon name, verify their validity, extract respective FASTA sequence and print it to standard output.
   - Work on file `Oxalis_HybSeq_nrDNA_selection_alignment.fasta`.
3. Think about several tasks where you would use BASH scripts to automatize repeated work and/or processing of multiple files.
   - Think about usage of various loops (and/or GNU Parallel), if-then branching, various conditions, etc.
   - Prepare concept(s) of such script(s).

Keep in mind...

Do not do simple repeated work worth of trained monkey manually. It is silly. Every such task is worth of (one-line) script to work instead of you.

Software

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
  - Repositories can be added whenever needed — check documentation for your distribution (at least basic “how-to”)
- 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 outside repositories
- 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 required
- 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)
- Common distributions use to provide convenient graphical tool to manage software
  - Ubuntu Software Center
  - Synaptic — feature rich, graphical, advances, for any DEB distribution

Package management III

Installation of software

- Aptitude — feature rich, command-line, advanced, for any DEB distribution (more advanced version of Apt)
- DPKG — low-level, any DEB-based distribution
- YaST Software for openSUSE (feature rich, graphical as well as command-line)
- Zypper — feature rich, command-line, advanced, for openSUSE
- DNF — feature rich, command-line, advanced, for Fedora and another RPM based distributions (replacing older Yum)
- RPM — low level, any RPM-based distribution
- GNOME software — in most of distributions using GNOME
- 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, ...

Package management IV

Installation of software

- The task is always same, the exact work-flow and commands more or less differ among distributions...
- Tools like Android Google Play, Apple Store or Windows Store are inspired from Linux...

Task

Install some new software into your Linux. Use graphical as well as command-line tools. See following slides for instructions.

Package management in command line in openSUSE and SLE (basic commands) I

- Root password is required: use `sudo` or `su -`
- Package name `*.rpm`
- `zypper in` *`package`* — install *package*
- `zypper rm` *`package`* — remove *package*
- `zypper ref` — refresh repositories
- `zypper up` — update
- `zypper dup` – upgrade to newer release of whole distribution
- `zypper se` *`term`* — search *term*
- `zypper pa --orphaned --unneeded` — list packages, which can be safely removed
- `yast sw_single` — interactive manager
- `zypper lr` — list repositories
- `zypper ar` *`repository`* — add *repository* (URL of remote `*.repo` file)

Package management in command line in openSUSE and SLE (basic commands) II

- `zypper rr` *`repository`* — remove *repository* (name according to `zypper lr`)
- `zypper mr` *`repository`* — modify *repository* (see `man zypper` first or use `yast sw_single`)
- `zypper pa` *`package`* — get information about particular *package* or another query (e.g. list of dependencies, see `man zypper`)
- `rpm*` commands for other tasks
- `man zypper`, `man rpm` — usage help

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

- Root password is required: use `sudo` or `su -`
- Package name `*.deb`
- `apt install` *package* — install *package*
- `apt remove` *package* — remove *package*
- `apt update` — refresh repositories
- `apt upgrade` – upgrade packages
- `apt dist-upgrad` – upgrade to newer release of whole distribution
- `apt search` *term* — search *term*
- `apt autoremove` — clear packages
- `aptitude` — interactive manager
- `cat /etc/apt/sources.list` — list repositories
- `nano /etc/apt/sources.list` — add/remove/edit repositories

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

- `dpkg-*`, `apt-*` commands for other tasks
- `aptitude` is used in similar way as `apt` (e.g. `aptitude install` *package*, ...)
- Similar tasks can be done with `apt`, `apt-get`, or `aptitude`
- `man aptitude`, `man apt-XXX` (e.g. `man apt`), `man dpkg-XXX`, `man apt` — usage help

Package management in command line in RedHat, Fedora, CENTOS and derivatives (basic commands) I

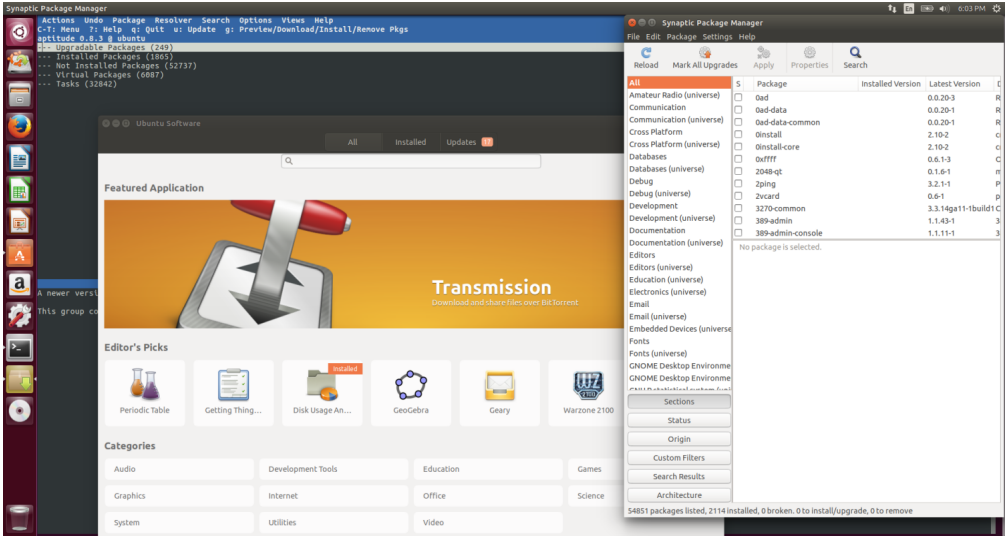
- Root password is required: use `sudo` or `su -`
- Package name `*.rpm`
- `dnf install` *package* — install *package*
- `dnf remove` *package* — remove *package*
- `dbf check-updat` — refresh repositories, check for updates
- `dnf upgrade` — upgrade packages
- `dnf search` *term* — search *term*
- `dnf autoremove` — clear packages
- `dnf info` *package* — get information about *package*
- `dnf repolist` — list repositories
- `dnf config-manage` — manage repositories and another settings
- `rpm -Uvh` *package*`.rpm` — install locally downloaded *package*`.rpm`

Package management in command line in RedHat, Fedora, CENTOS and derivatives (basic commands) II

- `rpm -e` *`package`* — remove *package*
- `rpm*` commands for other tasks
- In older releases of Fedora, RedHat and CENTOS, `yum` is used instead of `dnf` in nearly identical way
- `man dnf` (or `man dnf`), `man rpm` — usage help

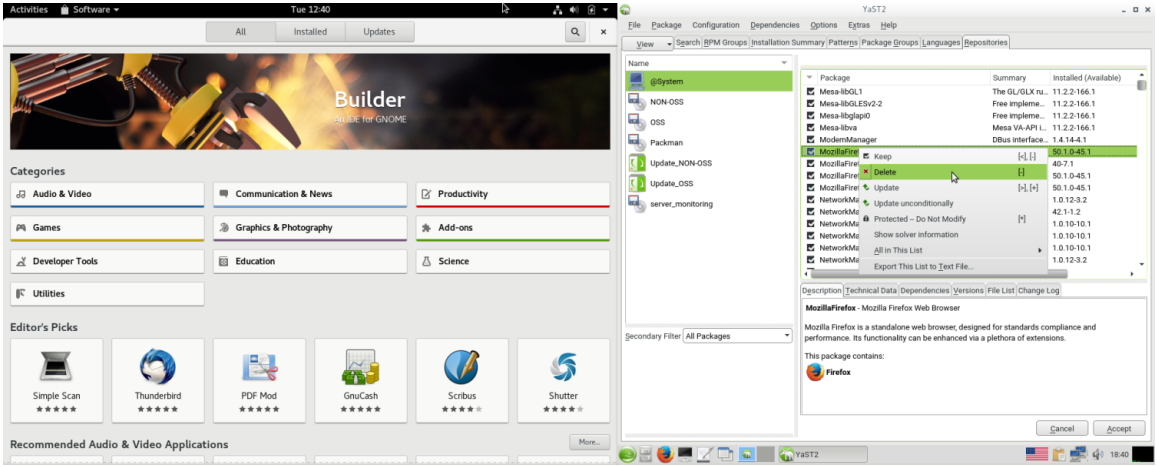
Graphical package managers I

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



Graphical package managers II

GNOME Software in Fedora and YaST in openSUSE



- Practically every common general distribution has some graphical tool... Explore it...

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/Portal:Zypper
- Red Hat, Fedora, CENTOS, Scientific Linux and others: http://yum.baseurl.org/ and newer https://fedoraproject.org/wiki/DNF
- 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...

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 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, Fedora and derivatives (2 options)
dnf groupinstall "Development Tools" "C Development Tools and Libraries"
yum groupinstall "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/v8.2.12
cd v8.2.12/ # 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/1.10/
  samtools-1.10.tar.bz2 # Download SAMtools
tar xjvf samtools-1.10.tar.bz2 # Unpack the archive
cd samtools-1.10/ # 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=96&num=3`

```
# Go to directory where you downloaded it
cd directory/with/downloaded/figtree
# Decompress downloaded archive
tar zxvf FigTree_v1.4.3.tgz
# Go to created directory
cd FigTree_v1.4.3/
ls * # List files, also in subdirectories
# Launch it (command java launches *.jar files)
java -jar lib/figtree.jar
# Limit its memory usage to 128 MB
java -Xmx128m -jar lib/figtree.jar
```

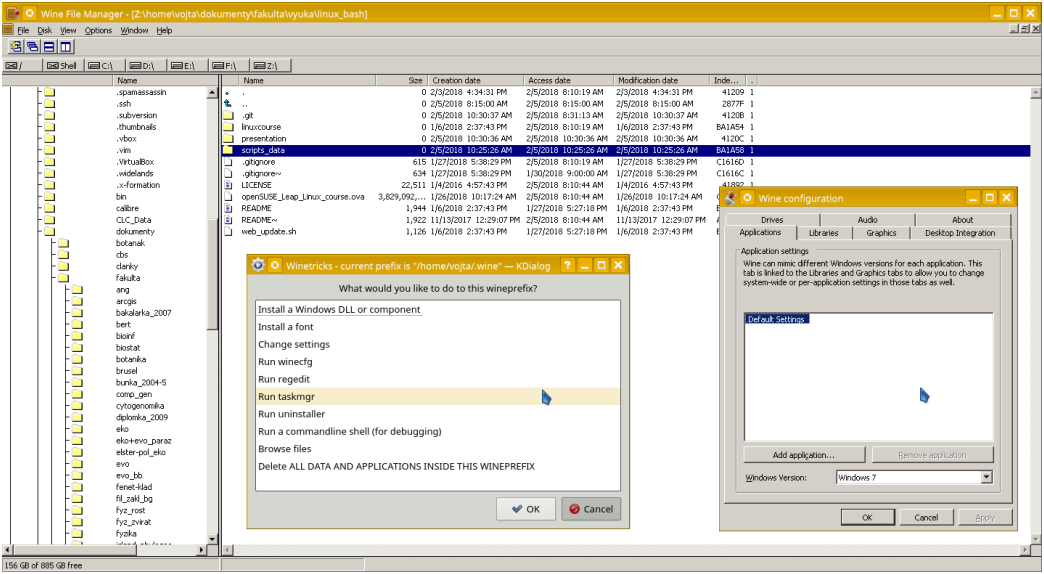
Windows applications on Linux I

- 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 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

Windows applications on Linux II

- 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
- 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. VirtualBox (or another virtualization platform), if needed

winefile, winetricks and winecfg



MetaCentrum

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
  - Large data storage
  - FileSender to be able to send up to 1.9 TB file
  - Cloud — 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) — similar to e.g. Dropbox
    - It is possible to connect by webDAV to ownCloud (slide 121) — many applications support it
    - It is possible to share calendars and/or address books via calDav and cardDav among devices and/or people
- Services accessible without registration
  - ownCloud `https://owncloud.cesnet.cz/`
  - FileSender `https://filesender.cesnet.cz/`
  - Go to web and log in with your institutional password

CESNET and MetaCentrum II

- 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
  - 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
    http://hostel.eduid.cz/en/index.html
- 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/
- 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 9 front ends where users log and thousands of computers doing the calculations — they are not accessed directly to run task
- Most of computers are running Debian GNU/Linux

MetaCentrum usage

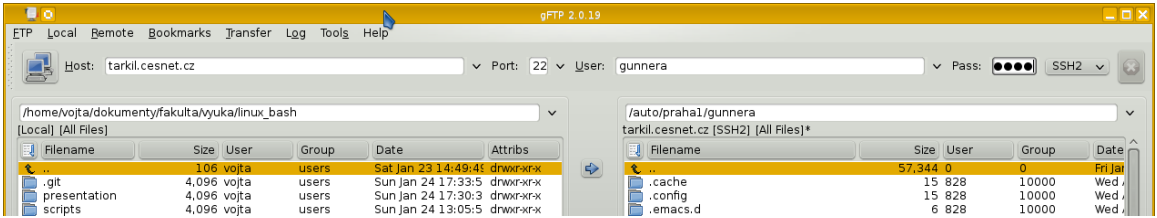
- User can transfer data on one of frontends (next slide) by e.g. `scp` or WinSCP from Windows or FileZilla from anywhere
- Same credentials are used for all front ends, for SSH login as well as file transmissions

```
# Login to selected server (tarkil is located in Prague)
ssh USER@tarkil.metacentrum.cz
# 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` the 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 front end
- All servers are accessible under domain `*.metacentrum.cz`: `skirit`, `perian`, `onyx`, `zuphux` (located in Brno), `alfrid`, `nympha`, `minos` (in Pilsen), and `tarkil` (in Prague) — so that e.g. `tarkil.grid.cesnet.cz` is synonymous to `tarkil.metacentrum.cz`
- See slide 121 and following to command-line transfers of files



Basic skeleton of script running tasks I

```
#!/bin/bash
# Modify the script according to your needs!
# Set data directories
WORKDIR="my_data_dir" # Or something else
DATADIR="/storage/praha1/home/$LOGNAME" # Or other storage
# There is directory /storage/praha1/home/USER/my_data_dir
# (in this case) containing all the data needed for calculations
# Clean-up of SCRATCH (it is temporal directory created by server)
# - the commands will be launched on the end when the job is done
trap 'clean_scratch' TERM EXIT
trap 'cp -a "$SCRATCHDIR" "$DATADIR"/ && clean_scratch' TERM
# Prepare the task - copy all needed files from working directory
# into particular computer which will finally do the calculations
cp -a "$DATADIR"/"$WORKDIR"/* "$SCRATCHDIR"/ || exit 1 # If fails, exit
# Change working directory - script goes to the directory where
# calculations are done
cd "$SCRATCHDIR"/ || exit 1 # 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"
module add parallel # In this (example) 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 select=1:ncpus=8 (see further)
find . -name "*.nexus" -print | parallel -j 8 'mb {} | tee -a {}.log'
# Copy results back to home directory
cp -a "$SCRATCHDIR" "$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/How_to_compute/Requesting_resources
- 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 (120 hours), require one physical computer
# with 8 CPU threads, 24 GB of RAM, 10 GB of disk space and we get all
# information mails (for abort, beginning, exit)
qsub -l walltime=120:0:0 -l select=1:ncpus=8:mem=24gb:scratch_local=10gb
  -m abe metacentrum.sh
# Check how the task is running (above web) and
qstat -u $USER # Information about $USER's jobs (queued and running)
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
```

Key MetaCentrum commands

- MetaCentrum is "just" normal Linux server — work there as on any other Linux system
- Command `module` loads/unloads selected application
- Tasks (BASH scripts) are submitted for computing by `qsub` — the script must copy the data into `$SCRATCHDIR` and do all calculations there
  - It has plenty of options how to specify requirements (see next slide and help)
- Queued and running jobs can be seen by `qstat -u $USER` (`qstat` has much more options) and any job can be terminated by `qdel 123456789` (number from `qstat`)

```
module add <TAB><TAB> # Load some module
module rm XXX # Unload selected module
module list # List of currently loaded modules
qsub ... # Submit task for computing - select any parameters needed
qstat -u $USER # See $USER's running and queued jobs
qdel 123456789 # Termite task (number from qstat)
```

Scheduling details I

- Specify needed time
  - Always `hours:minutes:seconds`, so e.g. for 4 weeks use `-l walltime=672:0:0` ($28 \cdot 24$), for two days and 12 hours `-l walltime=60:0:0`
  - User may ask to prolong the walltime — it is needed to write in advance
- Ask for as much RAM as you need (e.g. `-l mem=8gb` to request 8 GB)
  - 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...
- Disk space is relatively free resource, user can ask more to have some reserve (e.g. `-l scratch_local=10gb` to request 10 GB)
- Specify how many physical computer(s) you are going to use (e.g. `-l select=1` for one machine) and number of CPU threads on each machine (e.g. `-l select=1:ncpus=8` for 1 machine with 8 cores or `-l select=2:ncpus=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

Scheduling details II

  - 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 or killed by the server
  - If using more physical machines, ensure correct settings of e.g. MPI (see documentation for respective software you are using)
- If requesting e-mails (e.g. `-m abe` to get mail about abort, beginning and exit of the task) and submitting plenty of tasks by some script, it can result in hundreds of mails — receiving 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
- For more options read `https://wiki.metacentrum.cz/wiki/How_to_compute/Requesting_resources`

Scheduling details III

- Request special CPU (AMD, graphical, ...), e.g. CPU with AVX2 `-l select=cpu_flag=avx2`
- SSD local storage (e.g. `-l scratch_ssd=1gb`)
- Request particular location, ...
- `https://metavo.metacentrum.cz/pbsmon2/qsub_pbspro` helps with preparation of `qsub` command

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
  - Rather 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 sometimes 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 front end login to respective node running the task
ssh exec_host # No need to specify user name; e.g. mandos9
# Go to SCRATCH directory
cd SCRATCHDIR # e.g. /scratch/gunnera/job_1234567.arien-pro.ics.muni.cz
# There are working data of currently running task...
# Check whatever you need...
tee # Command recording outputs or software, normal output of software
    # is written into log, which is unavailable until the task is over
```

Running R tasks on MetaCentrum

- There are no R packages, user must create local package library and provide path
- Be careful about paths!
- In the `metacentrum.sh` script load R `module add R-3.3.1-int` and start R script as usually R CMD BATCH `script.r`

1. Login to selected front node via SSH
2. Go to working directory `cd workdir`
3. Create new directory for R packages `mkdir rpkgs`
4. Start R R
5. Install **all** R packages needed for the task — install them into the `rpkgs` directory `install.packages(pkgs=..., lib="rpkgs")`
6. In the R script `*.r` load the packages from the `rpkgs` directory `library(package=..., lib.loc="rpkgs")`
7. Ensure all needed outputs are saved from the R script

Interactive tasks

- See information at https://wiki.metacentrum.cz/wiki/How_to_compute/Interactive_jobs

```
# Secure we can log off in the meantime
screen # Or tmux
# Again launch qsub according to actual needs
# Note "-I" for interactive session and missing script name
qsub -I -l walltime=2:0:0 -l select=1:ncpus=1:mem=2gb:scratch_local=1gb
# Wait for job to start..
cd $SCRATCHDIR # Go to $SCRATCHDIR - work there
# 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"
# When you logout, the task is done (use screen to secure connection)
```

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

MetaCentrum task

Now work on MetaCentrum...

1. Upload to any MetaCentrum fronted files `metacentrum_oxalis.sh` and `oxalis_assembly_6235.aln.fasta` from `scripts_data`.
2. See `metacentrum_oxalis.sh`, be sure you know what it is doing, that you understand every command there.
3. If needed, edit `metacentrum_oxalis.sh` (e.g. correct paths).
4. Submit the task via `qsub`.
5. Monitor the task with `qstat`.
6. Login with SSH to computing node where the task is running, go to $SCRATCHDIR and see its progress. Look at output of `ps ux`. What does it say?
7. Wait for the task to be done and explore output.
8. If you encounter any error, try to find its source and fix it.

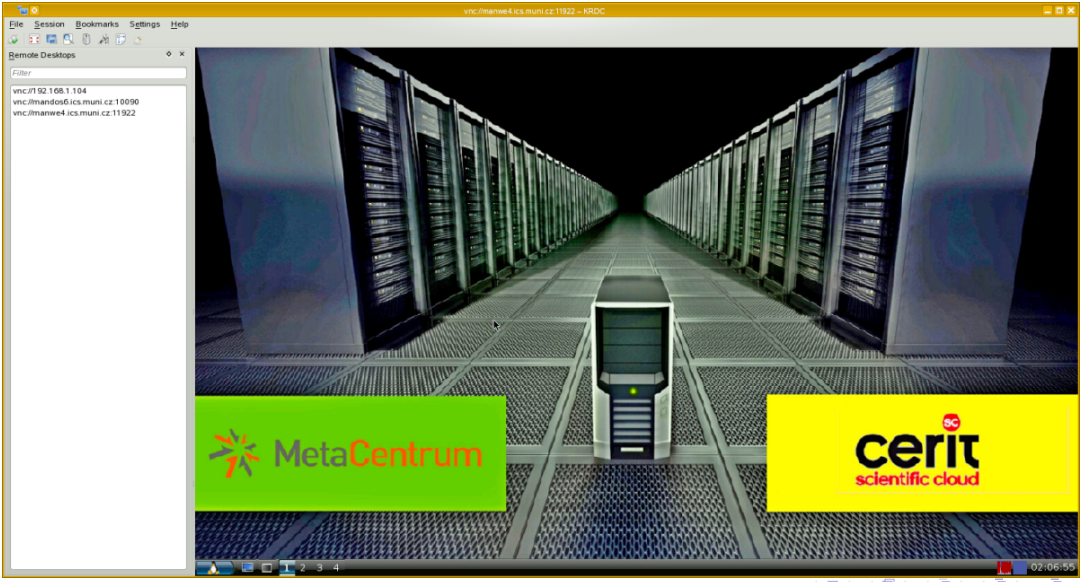
Graphical interactive task

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

```
screen # Secure we can log off in the meantime
# Again launch qsub according to actual needs
# Note "-I" for interactive session and missing script name
qsub -I -l walltime=2:0:0 -l select=1:ncpus=1:mem=2gb:scratch_local=1gb
# Wait for job to start..
# After we get the interactive task, we are on new server
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
- It provides limited amount of resources, not suitable for big tasks

Running VNC


vnc://manwe4.ics.muni.cz:11922 – KRDC
File Session Bookmarks Settings Help
Remote Desktops
Filter
vnc://192.168.1.104
vnc://mandos6.ics.muni.cz:10090
vnc://manwe4.ics.muni.cz:11922
MetaCentrum
cerit
scientific cloud
1 2 3 4
02:06:55

CESNET data storage

- Read documentation `https://du.cesnet.cz/` and connection instructions `https://du.cesnet.cz/en/navody/sluzby/start`
- See slide 220 for general information and 121 for connecting information
- Generally, it is possible to connect via FTPS, NFS, SAMBA (Windows network drive), SCP/SFTP or SSH
- After logging via SSH, it is possible to work as on any other server
- Users get information about selected storage location and paths after registration (there are three locations, user get space on one of them)
- All storage locations are accessible from any MetaCentrum front node via directories in `/storage` (e.g. `/storage/ostrava2-archive/tape_tape/VO_cuni_prf_..`)

Shared space on CESNET data storage

- Users can ask for creation of shared space
  - Normally, the space is private only for particular user
  - Groups allow more users to share data
  - Data storage admins will instruct users regarding locations, paths, permissions, etc. (it is specific for each case)
- Users must carefully set permissions!
  - Sharing is done by specific UNIX group
  - Users must set group ownership to particular group and permissions e.g. 770 for directories and 660 for files to avoid access of any other users
  - All members of the group must be able to manipulate the data

```
# Change group ownership to XXX
chgrp -R XXX /tape_tape/VO_XXX 2>/dev/null
# ("-R" to modify also subdirectories; "2>/dev/null" to discard errors)
# Set correct permissions to directories and files
find /tape_tape/VO_XXX -type d -exec chmod 770 {} \;
find /tape_tape/VO_XXX -type f -exec chmod 660 {} \;
```

Git

Git and version control I

- Git is version controlling system — it traces changes among all versions — absolutely crucial for any software development
- Older (nowadays not so common) version controlling systems is Subversion (SVN), there are many more (Bazaar, Mercurial, ...)
- 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
- Users work on a project in some directory as usually and from time to time commit local changes to stagging (temporal area) and then push them to remote or local repository (directory storing all Git history of particular project, can be anywhere)
- Every commit is checkpoint in the history
  - Can be tagged (named), e.g. particular released version of software, or project stage

Git and version control II

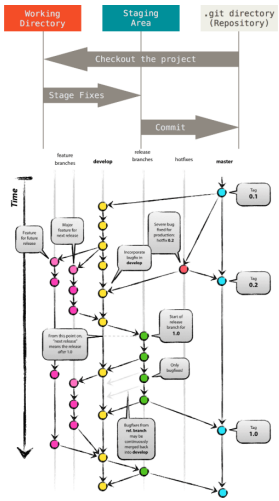
- User can compare any two commits or commit and current state
- Code can be branched
  - When new branch is created, it contains copy of current state
  - User selects in which branch she/he is working, the branches are diverging, user can commit in every branch independently
  - Commits in various branches can be compared and branches can be merged again
  - Key feature of Git — branches are useful when user starts to work on any big change in the project
- Stagging area serves as local "container" of changes to be pushed to central Git repository (or to be modified, discarded)
  - It is very hard to do any change in the central Git repository (e.g. to remove accidentaly pushed file or change description)
- Central repository keeps whole history of the project
  - Every user has full copy of the central Git repository — can be large
  - Can be placed on some server, network disk, in the local computer, ...
- Git was developed by Linus to trace development of Linux kernel, now it is probably the most used version control tool, used also by Microsoft to trace development of Windows :-)

GitHub and others

- GitHub is currently probably the most popular platform to host development of open-source projects, see documentation
- Probably second most common on-line service providing Git repository is GitLab
  - User can create an account on GitLab (as on GitHub), or download and install GitLab system on own server
- SourceForge used to be more popular in the past, still harbours plenty of interesting projects
- For on-line services, many tools are available only for paying customers
- All such services provide Git repositories accessible through standard tools (from command line, see further; or from some special application)
  - Exact usage might differ from standard Git, check respective help pages first
- Git repository can be easily hosted e.g. on any Linux server...

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



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

Working with Git — start and sending changes

```
# Create a new repository for new project (in empty directory)
git init # No need when cloning from existing repository
# Create a new central repository (e.g. on a server) in empty directory
git init --bare
# 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
```

Working with Git — branching and getting changes

```
# Push changes into the repository (regardless where it is)
git push origin master # See further for selection of branches
# 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
git push origin --delete <branch> # Delete remote branch
# New 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
# Download news from central server
git fetch
# Update local repository to the newest version from central repository
git pull # Fetch and merge remote changes (before commit)
# Merge another branch into the current one
git merge <branch>
```

Working with Git — tags, logs and settings

```
# 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>
# 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
       # Default behavior of Git can be heavily altered
```

Administration

Managing system services

- Different among distributions — several main methods
- In Linux, most common is SystemD, less common older init scripts and RC scripts
- macOS 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 first!
- 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 overridden config files after upgrade
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
# And much more...
```

The End

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 https://www.tldp.org/LDP/Bash-Beginners-Guide/html/Bash-Beginners-Guide.html (the site has plenty of good resources)
- Grymoire for UNIX wizards https://www.grymoire.com/Unix/

Resources to learn to work in the terminal II

- BASH Guide for Beginners http://linuxcourse.rutgers.edu/documents/Bash-Beginners-Guide/
- Linux tutorial https://ryanstutorials.net/linuxtutorial/
- Getting Started with BASH https://www.hypexr.org/bash_tutorial.php
- Bash Guide https://mywiki.wooledge.org/BashGuide
- Česky
  - Učebnice Linuxu https://www.abclinuxu.cz/ucebnice
  - Příkazová řádka Ubuntu http://wiki.ubuntu.cz/syst%C3%A9m/p%C5%99%C3%ADkazov%C3%A1_%C5%99%C3%A1dka/termin%C3%A1l
  - Příkazová řádka Fedory https://wiki.mojefedora.cz/domains/wiki.mojefedora.cz/doku.php?id=navody:prirucka:prompt

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 and version of distribution and of the problematic software
- **OSS is free as freedom of speech — not as free beer!**
  - As soon as you don't pay for support, you can't blame anyone for lack of responses
  - Most of software we use to process our data is provided under "best effort", without warranty...
  - Plenty of scientific software is not written by professional programmers, authors often do not foresee everything what could happen and they could have troubles wehn fixing reported issues...

How to ask for help II

- 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, unexpected interaction with another software, ...
- 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 — it takes time...
- Reading documentation is not wasting of time!
- Plenty of software packages have bug (issue) trackers (e.g. on GitHub) to report any problem with the software and discuss — search them on software homepage

Main general fora I

Question must have certain form!

Before asking, ensure your question is in answerable form (previous slide).

- Sloppily asked question can't be answered at all...
- Check documentation, manuals and search the Internet before asking

- 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/`
- Questions mainly (not only) related to servers `https://serverfault.com/`
- Do not hesitate to ask on the forum or contact developers, preferably through some public forum or mailing list, they usually respond quickly and helpfully... — they wish their software to be working and useful

Main general fora II

- Uncle Google is your friend here (*"how to XXX in BASH/Linux"*)...
- Plenty of bigger projects have their own web fora or e-mail conferences — search for it to ask on right place
- In Linux/OSS world, e-mail conferences and IRC are sometimes more popular, than web forums — try them
- If you find a bug, report it according to instructions given by the project
- Bioinformatics is discussed on Biostars (including various related programming issues)
- Plenty of software packages have bug (issue) trackers (e.g. on GitHub) to report any problem with the software and discuss — search them on software homepage
- Programming languages like R or Python have their own discussion fora, commonly specific for particular field

openSUSE

- Homepage https://www.opensuse.org/
- Wiki (knowledge base) https://en.opensuse.org/Main_Page
- Documentation https://doc.opensuse.org/
- News https://news.opensuse.org/, developers' blogs https://lizards.opensuse.org/, aggregation of blogs https://planet.opensuse.org/
- Forums https://forums.opensuse.org/
- Mailing lists https://lists.opensuse.org/
- Community https://opensuse-community.org/, short community guide https://opensuse-guide.org/

Debian, Ubuntu, Linux Mint and derivatives

- Debian https://www.debian.org/
  - Documentation, wiki https://wiki.debian.org/
  - Support https://www.debian.org/support
- Ubuntu https://ubuntu.com/
  - Support https://ubuntu.com/support
  - Ask Ubuntu (probably the best forum) https://askubuntu.com/
  - Forum https://ubuntuforums.org/
  - Documentation https://help.ubuntu.com/
  - Kubuntu https://kubuntu.org/
  - Kubuntu forum https://www.kubuntuforums.net/
  - Xubuntu https://xubuntu.org/
  - Lubuntu https://lubuntu.net/
- Linux Mint https://www.linuxmint.com/
  - Documentation https://www.linuxmint.com/documentation.php
  - Forums https://forums.linuxmint.com/

Fedora

- Homepage https://getfedora.org/
- Communication and help overview https://fedoraproject.org/wiki/Communicating_and_getting_help
- Wiki https://fedoraproject.org/wiki/Fedora_Project_Wiki
- Official forum https://ask.fedoraproject.org/
- Documentation https://docs.fedoraproject.org/
- Community forum https://fedoraforum.org/

GNOME, KDE and XFCE

- GNOME https://www.gnome.org/
  - Help for users https://help.gnome.org/users/
  - Wiki https://wiki.gnome.org/
- KDE https://kde.org/
  - Forum https://forum.kde.org/
  - UserBase wiki https://userbase.kde.org/Welcome_to_KDE_UserBase
  - Application store https://store.kde.org/
  - KDE for education https://edu.kde.org/
  - Blogs https://planet.kde.org/
- XFCE https://xfce.org/
  - Documentation https://docs.xfce.org/
  - Wiki https://wiki.xfce.org/
  - Forum https://forum.xfce.org/
  - Blogs https://blog.xfce.org/

LibreOffice

- LibreOffice https://www.libreoffice.org/
  - Document Foundation https://www.documentfoundation.org/
  - Ask LO https://ask.libreoffice.org/
  - Wiki of LO https://help.libreoffice.org/ and DF https://wiki.documentfoundation.org/Main_Page
  - Documentation https://documentation.libreoffice.org/
- Česky
  - Novinky a informace https://www.openoffice.cz/
  - Fórum https://forum.openoffice.cz/
  - Podrobná příručka https://www.knihaoffice.cz/ (jedna z vůbec nejlepších dostupných knih)

České weby — zdroje informací a fóra I

- ABC Linuxu https://www.abclinuxu.cz/
- Root https://www.root.cz/
- LinuxExpres https://www.linuxexpres.cz/
  - Správa linuxového serveru https://www.linuxexpres.cz/praxe/sprava-linuxoveho-serveru
- LinuxDays (největší konference) https://www.linuxdays.cz/
- Seminář Install fest https://installfest.cz/
- Konference OpenAlt https://openalt.cz/
- OpenOffice/LibreOffice https://www.openoffice.cz/
  - Podrobná příručka https://www.knihaoffice.cz/
  - Fórum https://forum.openoffice.cz/
- Ubuntu https://www.ubuntu.cz/
  - Wiki http://wiki.ubuntu.cz/
  - Fórum http://forum.ubuntu.cz/
- Fedora https://mojefedora.cz/
  - Wiki https://wiki.mojefedora.cz/doku.php

České weby — zdroje informací a fóra II

- Fórum https://forum.mojefedora.cz/
- Příručka https://wiki.mojefedora.cz/doku.php?id=navody:prirucka:obsah

- Linux Mint https://www.linux-mint-czech.cz/
  - Fórum https://forum.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?