Linux Introduction (with HPC focus)

Jan Steiner
Zentrum für Informations- und Medientechnik

July 5-6, 2022

Agenda

Day 1

1. Introduction
2. Connecting to the cluster
   Exercise 1
3. Historical background
4. The command line
5. Directory structure
   Exercise 2
6. Files
7. Text display, search
   Exercise 3

Day 2

8. Users and permissions
9. Processes
   Exercise 4
10. The vim text editor
    Exercise 5
11. Shell scripting
    Exercise 6
12. Environment variables
13. System configuration files
14. Various tips
    Exercise 7
15. Beyond the cluster

Agenda

1. **Introduction**
2. Connecting to the cluster
   Exercise 1
3. Historical background
4. The command line
5. Directory structure
   Exercise 2
6. Files
7. Text display, search
   Exercise 3
8. Users and permissions
9. Processes
   Exercise 4
10. The vim text editor
    Exercise 5
11. Shell scripting
    Exercise 6
12. Environment variables
13. System configuration files
14. Various tips
    Exercise 7
15. Beyond the cluster

Who am I

- Jan Steiner
  - Aerospace Engineering, Uni Stuttgart (grad. 2010)
  - German Aerospace Center (DLR) Braunschweig
    - Numerical simulation of aircraft icing
  - At ZIMT since July 2017
- Area:
  - HPC training and support
  - Courses (like this one)
  - Cluster website
- Additional support: performance optimization

Round of introductions!

- What department/institute are you with?
- What is your field / research topic?
- How do you use / intend to use Linux?
- What is your previous experience?
- Is there something specific you want to learn today?

Introductory remarks

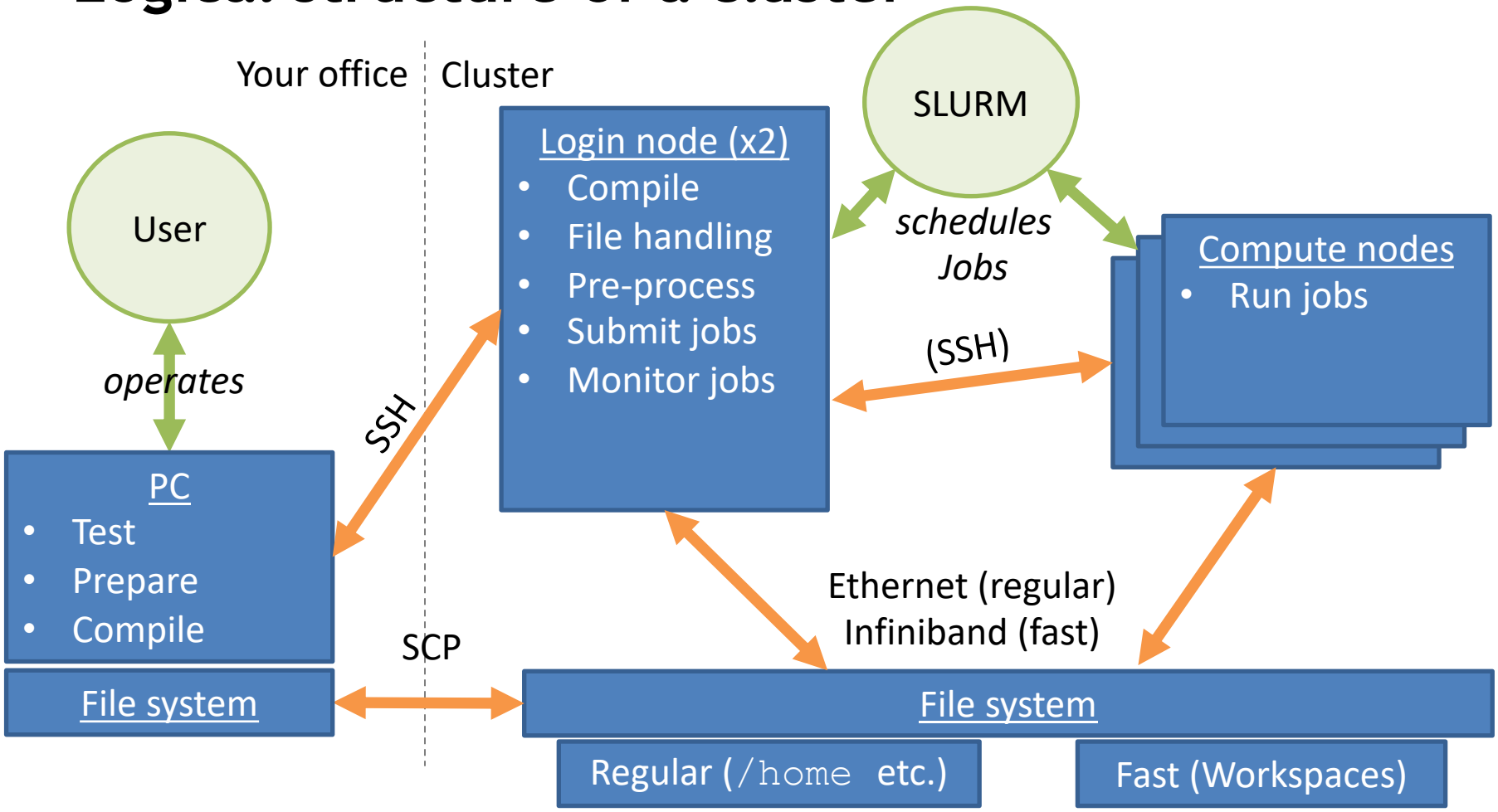
- Course is primarily for users of the OMNI cluster
  - Some special situations addressed
  - Assumes you have no root access
  - May use cluster if no own computer
- Covers things that are distro-independent
  - Only command-line
- Exercises are designed to be open
  - Proceed at your own speed
  - Ask for help and cooperate
  - If bored, get creative
  - You may use Google

Introductory remarks

- Three possibilities for this course
  1. If you have a Linux-capable computer, use that (use `bash`)
  2. If you have HPC access, work on the OMNI cluster
  3. If not, connect to OMNI cluster with the following information:

- Client to connect from Windows: MobaXTerm
  - You can download and install this in the first exercise

Logical structure of a cluster



Agenda

1. Introduction
2. **Connecting to the cluster**

   Exercise 1
3. Historical background
4. The command line
5. Directory structure

   Exercise 2
6. Files
7. Text display, search

   Exercise 3
8. Users and permissions
9. Processes

   Exercise 4
10. The vim text editor

    Exercise 5
11. Shell scripting

    Exercise 6
12. Environment variables
13. System configuration files
14. Various tips

    Exercise 7
15. Beyond the cluster

Getting access

- Get an account
  - Employees: Nutzerkontenverwaltung
    - Wait
  - Students: supervisor runs script
- Account usable the next day

- Set up an SSH connection
  - Explained in a moment

Problems

- Check cluster website: https://cluster.uni-siegen.de

- If website did not help: open a ticket
  - E-mail to hpc-support@uni-siegen.de
  - Central ZIMT dispatch sees it
  - Attach job script and logfile

- Please don't email us directly
  - Person might be on vacation etc.
  - Entire team has an overview what's wrong
  - Also not good: hpc-team@uni-siegen.de

Connecting to the cluster

- You can connect from any system
  - Linux: Easiest
  - Mac OS: Relatively easy
  - Windows: hardest, but feasible (several options)

- Outside university network:
  - Cluster directly reachable
  - VPN not necessary

Connecting to the cluster

- Cluster address:

  `<removed>`

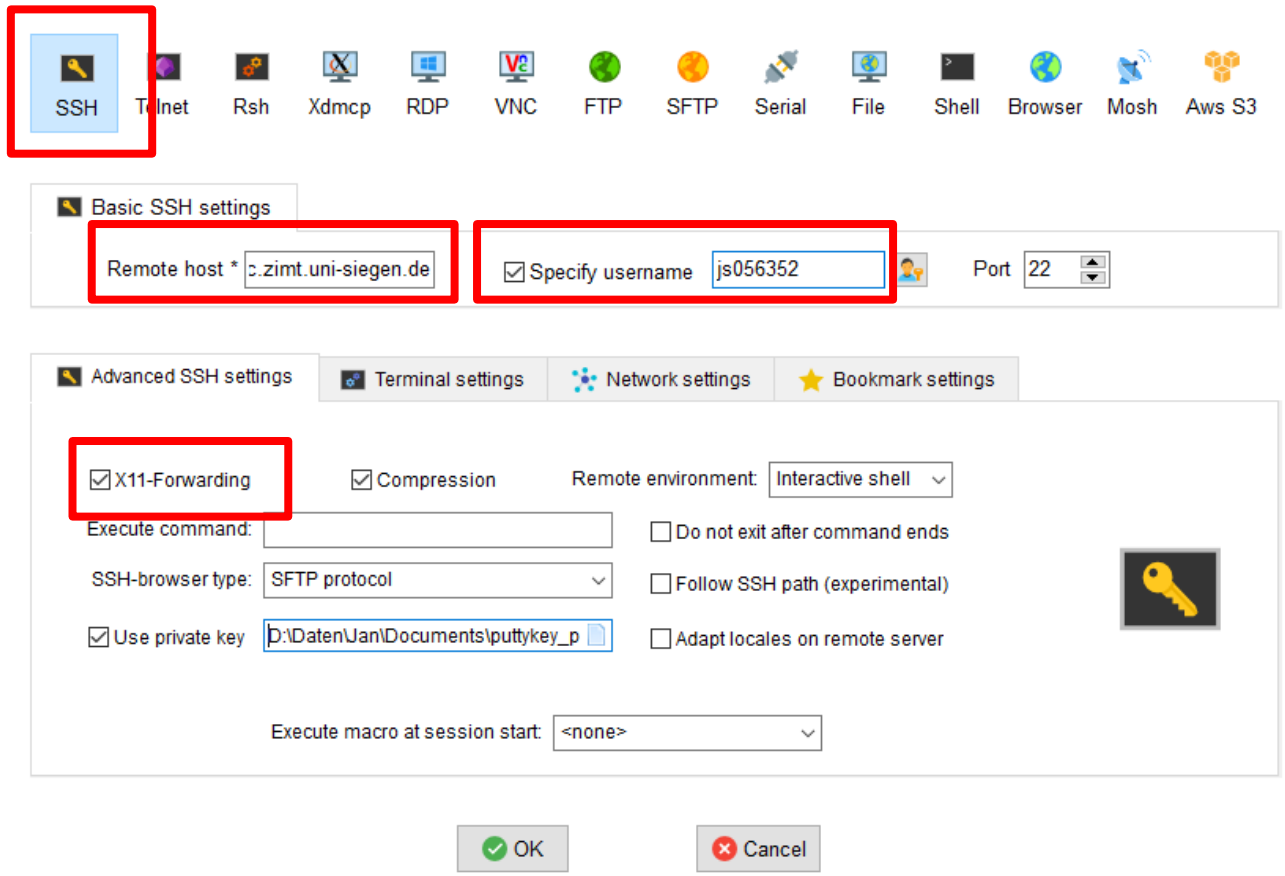
  - Redirects to one of the login nodes
- Linux or Mac OS system:

  `ssh -X <Username>@<address>`

  - Enter password
  - Mac OS (optional): iTerm software
- Windows: various ways:
  - Basically emulate Linux way

Connecting to the cluster

- MobaXTerm: practical, free, easy to use, many things built in
- Connection to cluster: Sessions→New→SSH
- Remote host: `<removed>`



Exercise 1

Goals:

- You can connect to the OMNI cluster via `ssh`

Tasks:

1. If on Windows, download MobaXTerm (Portable): https://mobaxterm.mobatek.net/download-home-edition.html
2. Set up an SSH session in MobaXTerm or your Linux console
   - Either your own account
   - Or `schulungXY` user

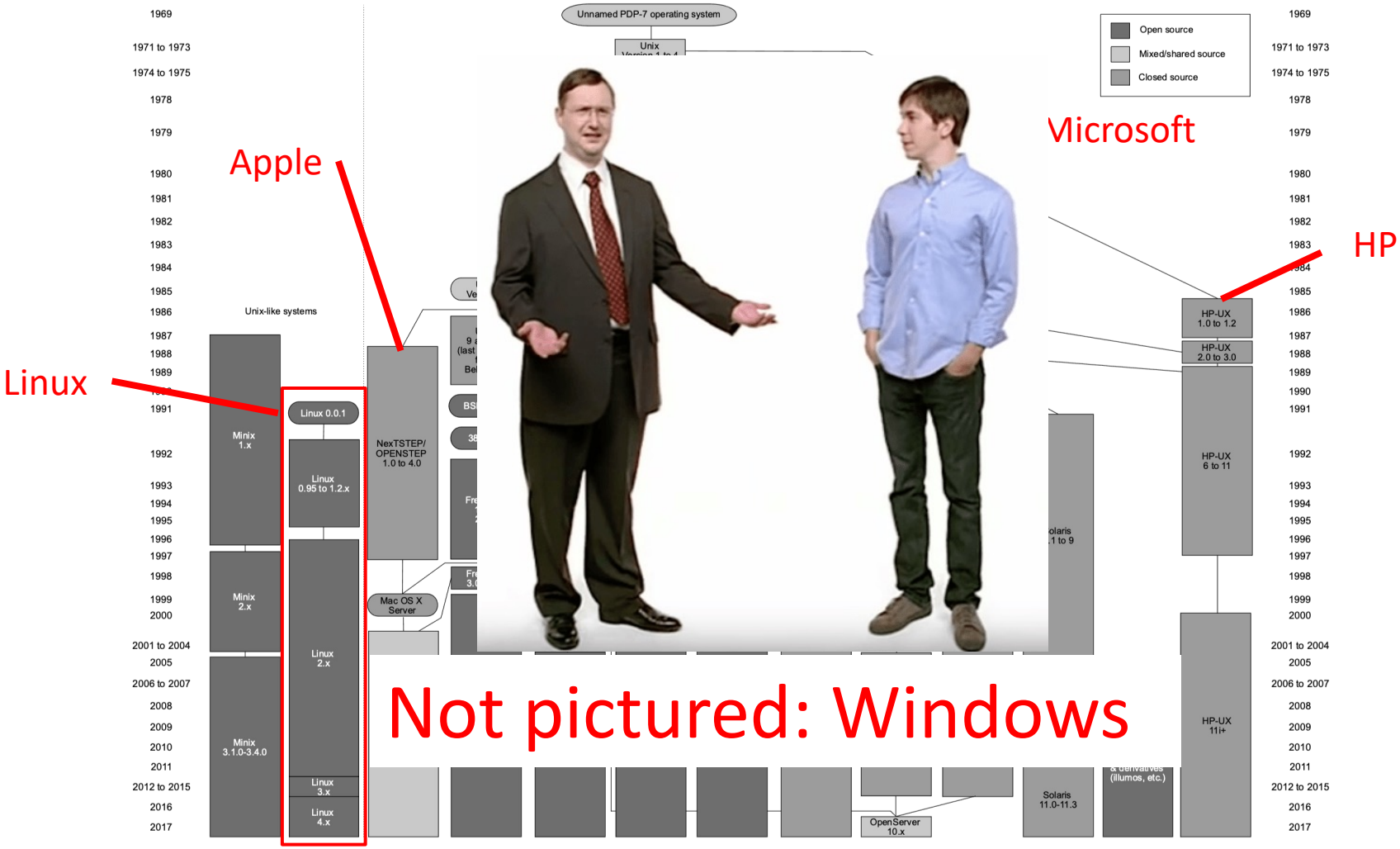
Agenda

1. Introduction
2. Connecting to the cluster
   Exercise 1
3. **Historical background**
4. The command line
5. Directory structure
   Exercise 2
6. Files
7. Text display, search
   Exercise 3
8. Users and permissions
9. Processes
   Exercise 4
10. The vim text editor
    Exercise 5
11. Shell scripting
    Exercise 6
12. Environment variables
13. System configuration files
14. Various tips
    Exercise 7
15. Beyond the cluster

Historical background

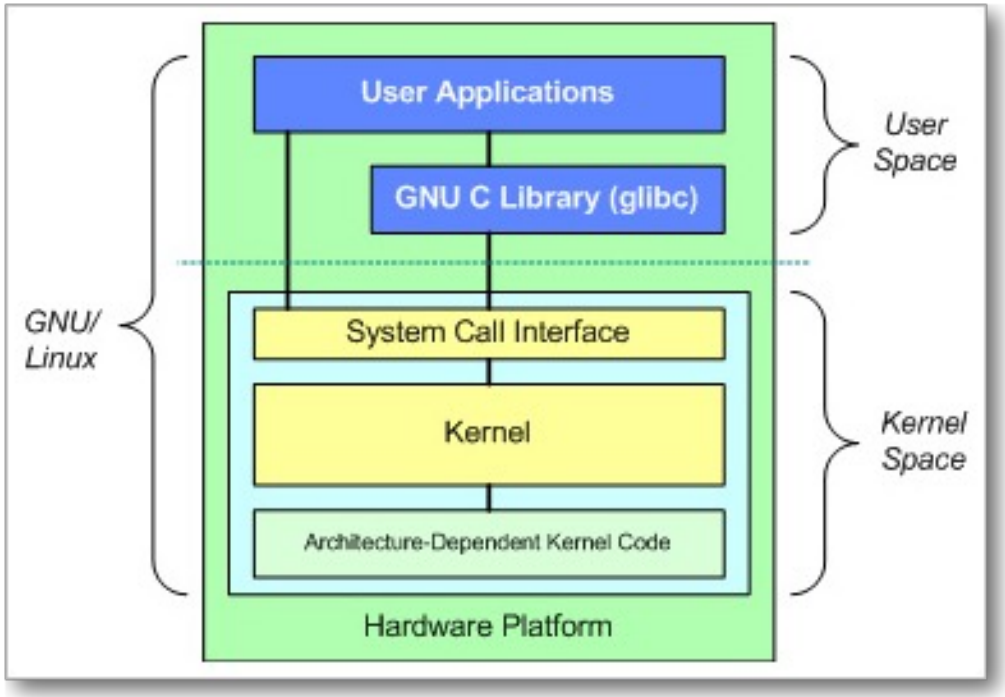
- 1969: Unix (Bell Laboratories)
  - Written in C
  - Already a successor to Multics
  - Over time, many variations

- 1990: POSIX Standard
  - Interface that all Unix systems implement
  - Adopted by Unix-like systems (including Linux)
  - Already many tools that we still use


Apple
Microsoft
HP
Linux
Not pictured: Windows
Unnamed PDP-7 operating system
Unix
Unix-like systems
Open source
Mixed/shared source
Closed source
Minix 1.x
Linux 0.0.1
Linux 0.95 to 1.2.x
Minix 2.x
Linux 2.x
Minix 3.1.0-3.4.0
Linux 3.x
Linux 4.x
NeXTSTEP/ OPENSTEP 1.0 to 4.0
Mac OS X Server
HP-UX 1.0 to 1.2
HP-UX 2.0 to 3.0
HP-UX 6 to 11
HP-UX 11i+
Solaris 11.0-11.3
OpenServer 10.x
1969
1971 to 1973
1974 to 1975
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001 to 2004
2005
2006 to 2007
2008
2009
2010
2011
2012 to 2015
2016
2017

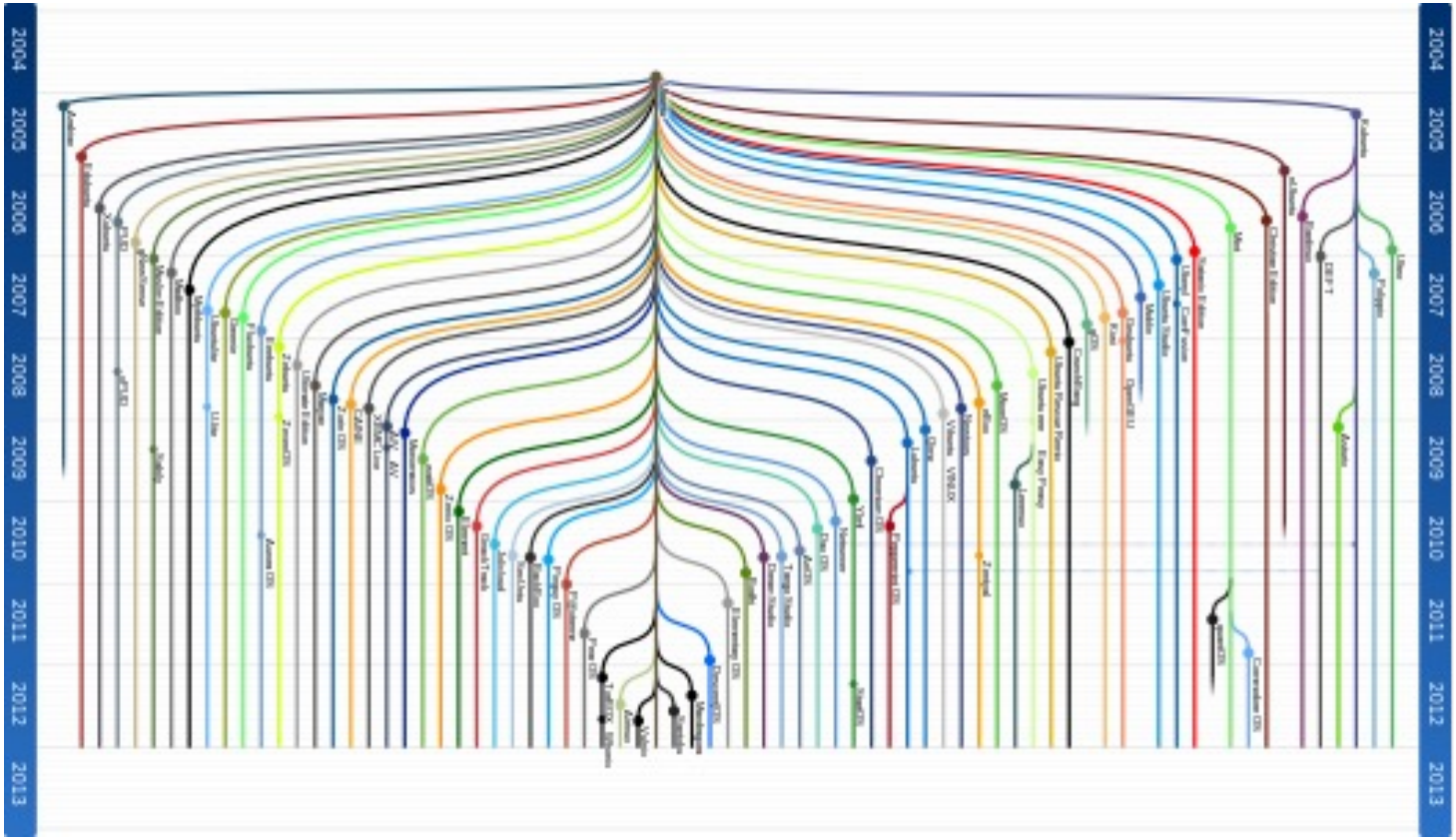
History of Linux

- Two separate initiatives:
  - GNU (GNU's Not Unix)
    - 1984: Richard Stallman and others
  - Linux
    - 1991: Linus Torvalds
- Nowadays: GNU/Linux:
  - Linux kernel
  - GNU utilities
- Many distributions (**distros**)



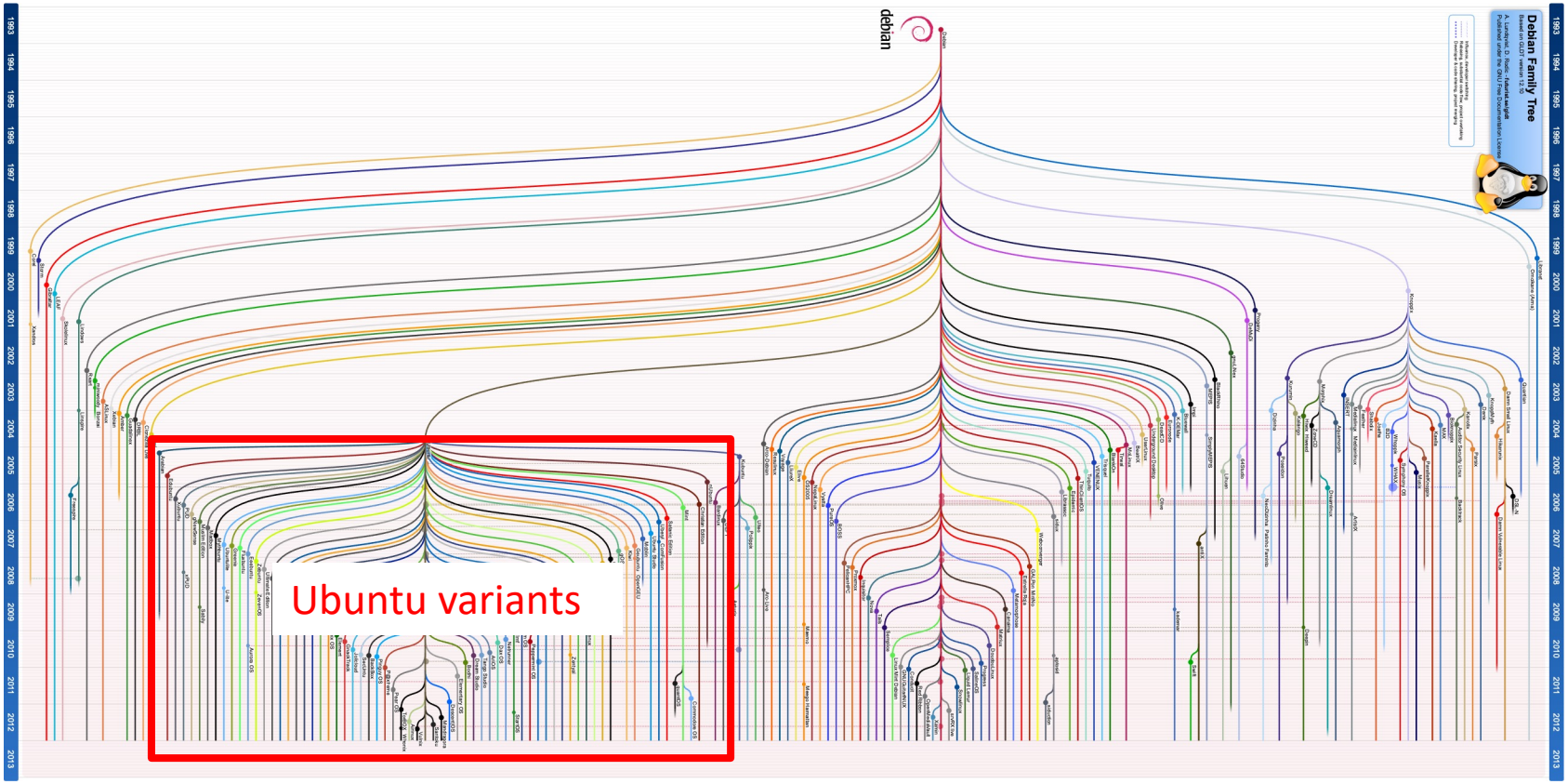
Tree of ~~Linux distributions~~

Ubuntu variants

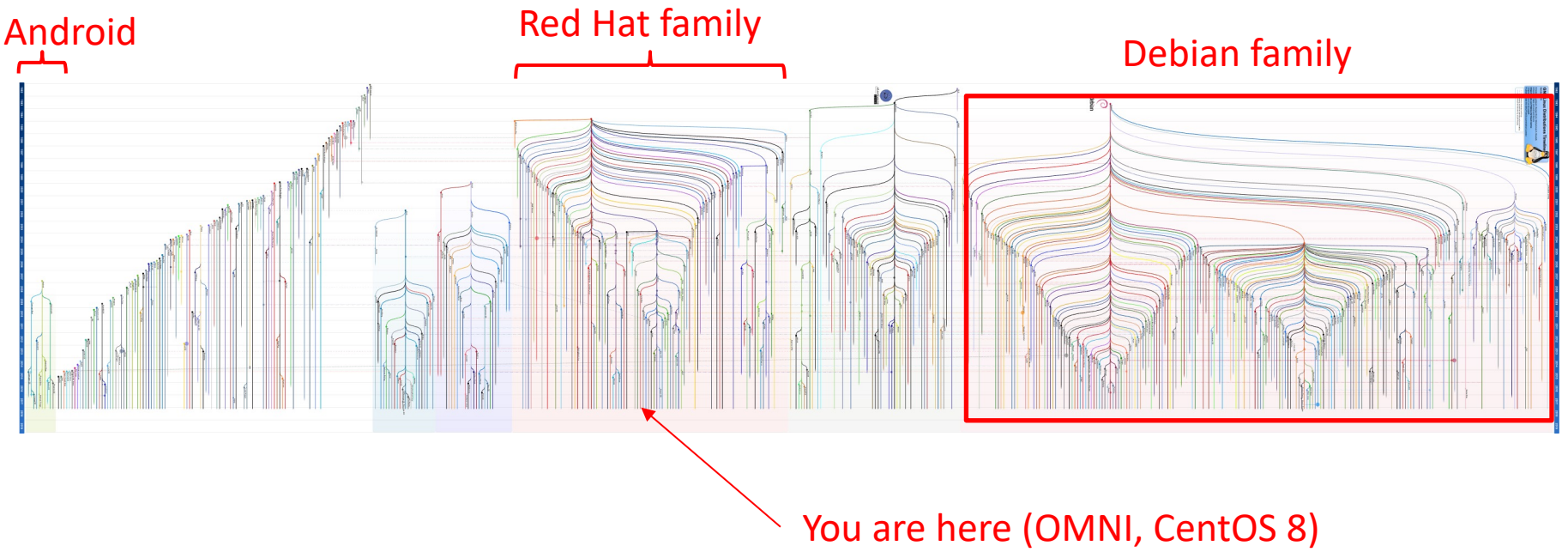


Tree of ~~Linux~~ distributions

Debian family



Tree of Linux distributions



Some important distros

- High reliability (e.g. servers): Red Hat Enterprise Linux
  - Developer ”playground”: Fedora
  - Community variant: CentOS (OMNI cluster)
- User-friendliness: Ubuntu
  - Community variant: Mint
  - “Parent”: Debian
- Workplace (especially Germany): Suse
- Specialized: e.g. Kali Linux (hacking tools)
  - Also runnable without installation

Popularity of distros

- Computers with Linux:
  - Desktop PCs: 1-2%
  - Mobile devices: 60-80 % of mobile devices (almost all Android)
  - (Web) servers: 95%
  - 500 out of the Top 500 supercomputers (2018)
- Popular desktop distros (no good figures):
  - Linux Mint
  - Ubuntu

Which Linux should I use

- In principle: any Linux will do
  - So much for the theory…
- What do I want to use it for?
  - Server vs. desktop
  - Stability vs. flexibility
  - Easy to learn vs. lots of features
- What software comes with it/is available?
  - Legal status of packages
- What is support/maintenance like?
  - Including documentation

Agenda

1. Introduction
2. Connecting to the cluster
   Exercise 1
3. Historical background
4. **The command line**
5. Directory structure
   Exercise 2
6. Files
7. Text display, search
   Exercise 3
8. Users and permissions
9. Processes
   Exercise 4
10. The vim text editor
    Exercise 5
11. Shell scripting
    Exercise 6
12. Environment variables
13. System configuration files
14. Various tips
    Exercise 7
15. Beyond the cluster

Command line

- A line where you type commands
- Other terms:
  - Console/Terminal
  - Terminal
  - CLI (command line interface)
  - Shell
- Advantages: simple (always works), fast *if you know commands*
- Disadvantage: lots of memorizing (vs. GUI buttons)

Why the name “shell”

- Linux: console below everything
- Programs within shells within shells
- Important for user: be aware where you are
  - Things not available in parent shell by default
  - What gets stopped if you close shell

Elements of the console

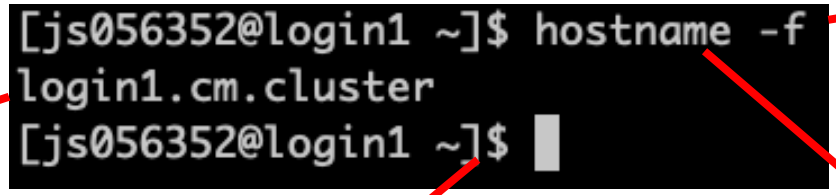
Command prompt
- *Indicates system is ready to receive input*
- $: convention (also >)

User name
Who is using this

Host name
What computer are you on?

Working directory
Where on the computer are we?

Elements of the console



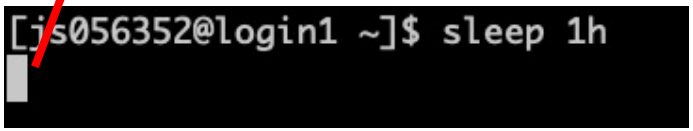
Options
- Command-specific
- Several conventions

Output
- *What the command returns*
- No rules

Command
- *What to do*
- Typed by user

New command prompt
Execution completed

Running



Navigating the command line

- Enter: run command
- Up-Arrow and Down-Arrow: command history
- Tab: auto-completion
  - More than one possibility: nothing shown
  - Second Tab to list possible commands
- Ctrl-C: abort current command.

Demo 4-1

Command line conventions

- Always case-sensitive
  - Popular source for errors

- Command line options:
  - Usually start with minus sign
  - Often: double minus sign, shortcuts with single minus sign
    - Example: `sbatch --time 0:30:00` is identical to `sbatch -t 0:30:00`
  - However: up to program developers, many programs do not follow this

How to find correct command/options

- Internet (seriously)
  - Very extensive community
  - Stack Overflow/Stack Exchange
- Man page:

```
man <command name>
```

- Built-in help:
  - Often `-h` or `--help` option
  - Often identical to man page

Demo 4-2

Agenda

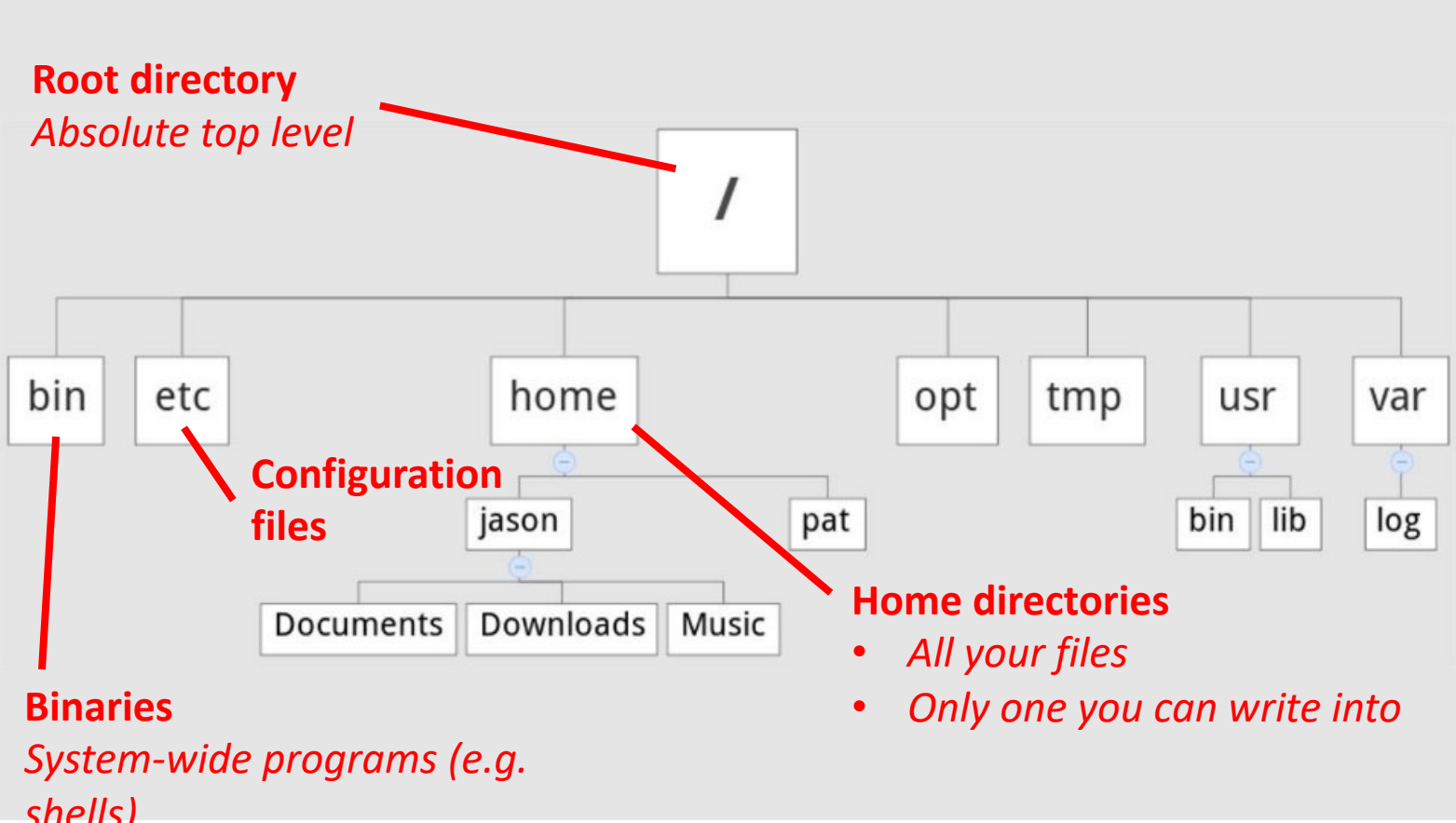
1. Introduction
2. Connecting to the cluster
   Exercise 1
3. Historical background
4. The command line
5. **Directory structure**
   Exercise 2
6. Files
7. Text display, search
   Exercise 3
8. Users and permissions
9. Processes
   Exercise 4
10. The vim text editor
    Exercise 5
11. Shell scripting
    Exercise 6
12. Environment variables
13. System configuration files
14. Various tips
    Exercise 7
15. Beyond the cluster

Directory structure

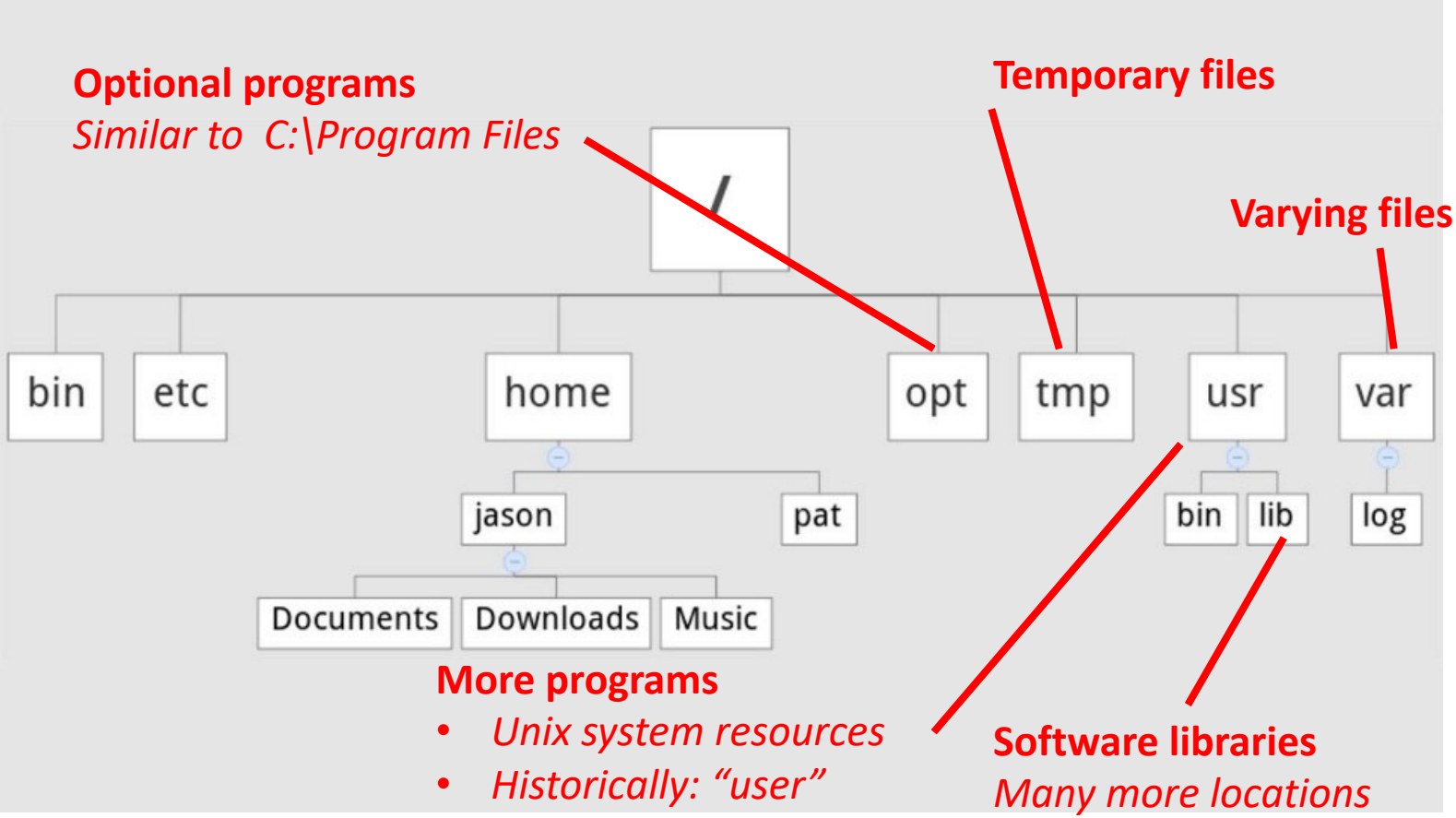
- Directory tree structure different from Windows
  - No drive letters (C:\)
  - Top level (mostly) identical on every Linux system
  - "Mounting points": location of hard drive in tree structure
- "Path": location inside file system
  - Example:
  - Absolute path (starts with `/`)
  - Relative path: relative to (current) **working directory**
- Print working directory: `pwd`

Demo 5-1

Example directory structure



Example directory structure



More about files and directories

- Linux principle: everything is a file

  `/dev`: Device files

  `/proc`: System information files

- (Almost) every command is a program or script somewhere

  `which <Commandname>` to see

- Special abbreviations for directories:

  . (period): current directory

  .. (two periods): parent directory

  ~ (tilde sign): your home directory

Navigating directories

- `cd` Command (change directory)
  - Part of POSIX standard
- Usage: `cd <Path>`
  - Can be relative or absolute
  - Must have at least execute permissions
    - Possible to execute but not read a file
  - May be special character, e.g. `cd ..` (parent directory)
- Common trap: `cd..` (no space in between) usually defined but not always

Navigating directories

- `ls` Command
  - Short for “List”
  - List directory contents
  - One of the most common commands in Linux (like `dir` in Windows)
  - `ls -l` is so common that it often has its own shortcut: `ll`
  - Can also show hidden files with `-a`
  - Can sort results, e.g. `-t` to sort by time modified

Demo 5-2

Console-specific commands and shortcuts

- Middle Mouse: paste highlighted text
  - NOT Ctrl+C / Ctrl+V, see below
- Ctrl+C: stop current command
- Ctrl+Z: suspend current command
- Ctrl+D: send “End-of-File” to application
  - Will usually quit console
- Quit console with `exit` (SSH connection: back to local console)
- Clear screen: `clear` command

First, a warning

- Console has no “Undo” button
- Usually no “Are you sure you want to delete” dialog
- If root: can theoretically destroy entire system

- Never run a command which you don’t understand
  - “Lol, try `sudo rm -rf /`“ - many idiots on the internet
- Make sure you are in the right directory
- Make sure you are not root unless necessary
- Check for spelling errors

Exercise 2

Goals:

- You can navigate the command line and the file system
- You can get help for Linux commands

Tasks:

1. Use `cd` and `ls` to look around your file system, locate discussed directories
2. Find out what options `cd` and `ls` have
3. Play around with tab auto-completion (don't run random commands please)
4. If bored, get creative

Agenda

1. Introduction
2. Connecting to the cluster
   Exercise 1
3. Historical background
4. The command line
5. Directory structure
   Exercise 2
6. **Files**
7. Text display, search
   Exercise 3
8. Users and permissions
9. Processes
   Exercise 4
10. The vim text editor
    Exercise 5
11. Shell scripting
    Exercise 6
12. Environment variables
13. System configuration files
14. Various tips
    Exercise 7
15. Beyond the cluster

File manipulation commands

- Simple commands to handle files
  - Most also work on directories
- You already know `ls`
- Rename file/directory: `mv <oldname> <newname>` (move)
- Copy file/directory: `cp <filename> <newname>` (copy)
  - Also needs `-r` for directories

Linux file basics

- Linux: extensions do not matter
  - But: conventions to help humans
  - Some programs also look at extensions
- Most important: text file or not?
  - Configuration files
  - Scripts
  - System information files
- Binary file: generally not searchable
- Command `file <filename>` to display what it is

Demo 6-1

Basic file manipulation

- Create directory: `mkdir <dirname>`
- Create empty file: `touch filename`
  - Mostly for testing purposes
- Remove file/directory: `rm <filename>`
  - Watch the permissions!
  - To delete content of subdirectories: `rm -r` (recursive)
  - Common option: `-f` (force) → never prompt for confirmation

Demo 6-2

Wild cards

- So far every command was for one file

- Option to specify patterns: wildcards
  - Also called globbing

- Most important
  - * zero or more characters
  - ? exactly one character
  - [] Range of characters

Wild cards vs. regular expressions

- Regular expressions: “sequence of characters that defines a search pattern”
  - Much more powerful
  - But also much harder to learn
- Similar concept, but not the same thing
- Will not be covered here

Searching for files

- Use `find` command
- Syntax: `find <targetdir> <options>`
  - Example: `find . -name "ex1.txt" -type f`
- Allows very complex searches
  - Wildcards
  - Only files modified after X
- Allows executing command for every found file: `-exec`

Wild cards and find command

- Wildcards: common source of problems, especially in scripts
  - Expanded by shell before being given to program
  - Problem not limited to `find` command
- Example: `find` command `-name` option

```
$ find . -type f -name *test*
```

  - The `find` command is handed multiple names, cannot handle this
- Fix: `$ find . -type f -name "*test*"`
  - Now string with wildcards is handed to `find` command

Demo 7

Agenda

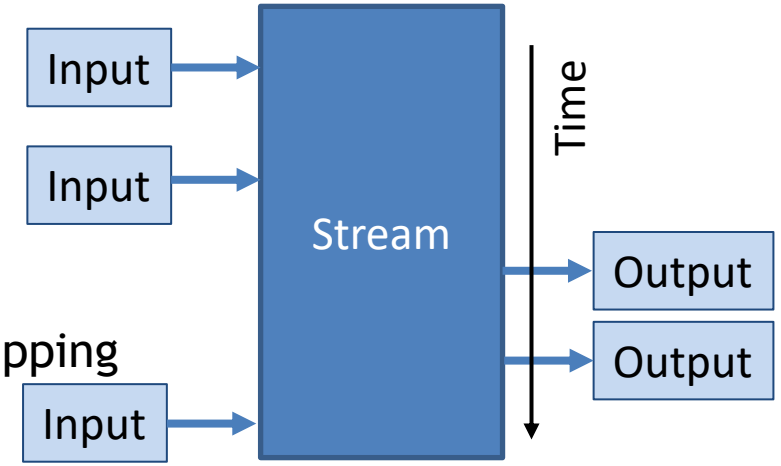
1. Introduction
2. Connecting to the cluster
   Exercise 1
3. Historical background
4. The command line
5. Directory structure
   Exercise 2
6. Files
7. **Text display, search**
   Exercise 3
8. Users and permissions
9. Processes
   Exercise 4
10. The vim text editor
    Exercise 5
11. Shell scripting
    Exercise 6
12. Environment variables
13. System configuration files
14. Various tips
    Exercise 7
15. Beyond the cluster

Console input and output

- Console has three main ways of communicating with process (so-called streams)
  - Standard input (`stdin`)
  - Standard output (`stdout`)
  - Standard error (`stderr`)
- `stdin`: what you type into console
- `stdout` + `stderr`: what you see in console
  - Reason: separate error messages from normal output

Side note: why are they called “streams”

- What is a “stream” in computing terms?
  - Intermediate storage
  - Stuff is put into it, stuff gets taken out
  - Those two may happen at any time, overlapping
- Example: streaming video
  - Video gets partially downloaded, you can already view it
  - You can pause, jump etc.
- In console: text gets written into stream and taken out
  - Input and output can be (re)directed to other sources/targets



Console input and output

- Input/output streams can be redirected
  - Other commands
  - Files
- Redirect stdout

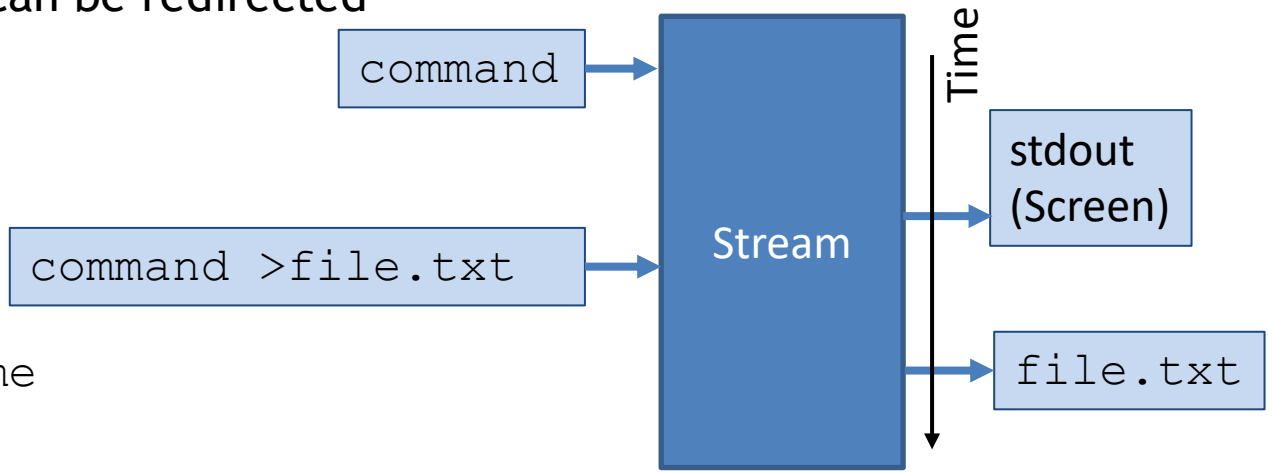
```
command > filename
```

- Redirect stdin

```
command < filename
```

- Use output of one command as input to another: pipe symbol

```
command1 | command2
```



Advanced redirection

- Stream redirection can do even more

  `command >> filename` will append to file without overwriting

  - Streams are numbered:

    0: `stdin`, 1: `stdout`, 2: `stderr`

  - Examples:

    ```
    command > out.log 2> err.log
    ```

    ```
    command 2>&1 > out_err.log
    ```

Demo 7-1

Text display

- Many different ways to display and edit text
  - Simplest: `cat` command
    - Outputs contents of a text file to console
  - More advanced: `less` command
    - Allows going back and forth
    - Also used by man pages
  - Many others:

    `head`: display first lines, `tail`: display last lines

Demo 7-2

Searching file contents

- Use `grep` command
- Syntax: `grep <options> <string> <filename>`
  - Example `grep -i -r "test" example*.txt`
- Like `find`, very powerful due to options + wildcards
- Common options:
  - `-r` Recursive (include subdirectories)
  - `-i` Ignore upper/lower case
  - `-I` Ignore binary files (capital i)

Use grep on search results

- Common situation:
  - Command with a lot of text output
  - You are looking for something inside output
- Solution: pipe output into `grep`

```
$ ll | grep -i test
```

- Note that there is no file specified in the `grep` call
- See how pipes can be useful?

Exercise 3

Goals:

- You can create and manipulate files and directories
- You can display and search text files

Tasks:

1. Create, rename, copy and delete some files
2. Explore different ways of displaying and searching files
   - Good file to practice: `/proc/meminfo`
3. Write the contents of `/proc/meminfo` into text file with redirection
4. If bored, get creative

Agenda

1. Introduction
2. Connecting to the cluster
   Exercise 1
3. Historical background
4. The command line
5. Directory structure
   Exercise 2
6. Files
7. Text display, search
   Exercise 3
8. **Users and permissions**
9. Processes
   Exercise 4
10. The vim text editor
    Exercise 5
11. Shell scripting
    Exercise 6
12. Environment variables
13. System configuration files
14. Various tips
    Exercise 7
15. Beyond the cluster

Users and groups

- Linux is a *multi-user* system
  - Everyone should only be able to access own files
    - Others only see/change what you want them to
  - Some files/directories should only be accessible to admins

- Everyone is logged in as a specific user (account)
  - Every user has certain permissions

- Only admins can set permissions for others

File/directory permissions

Each file and directory has certain permissions

- Determines what you can do
  - *You can't break what you can't use!*
- `root` user (superuser) can do everything
- Users may get temporary root permissions

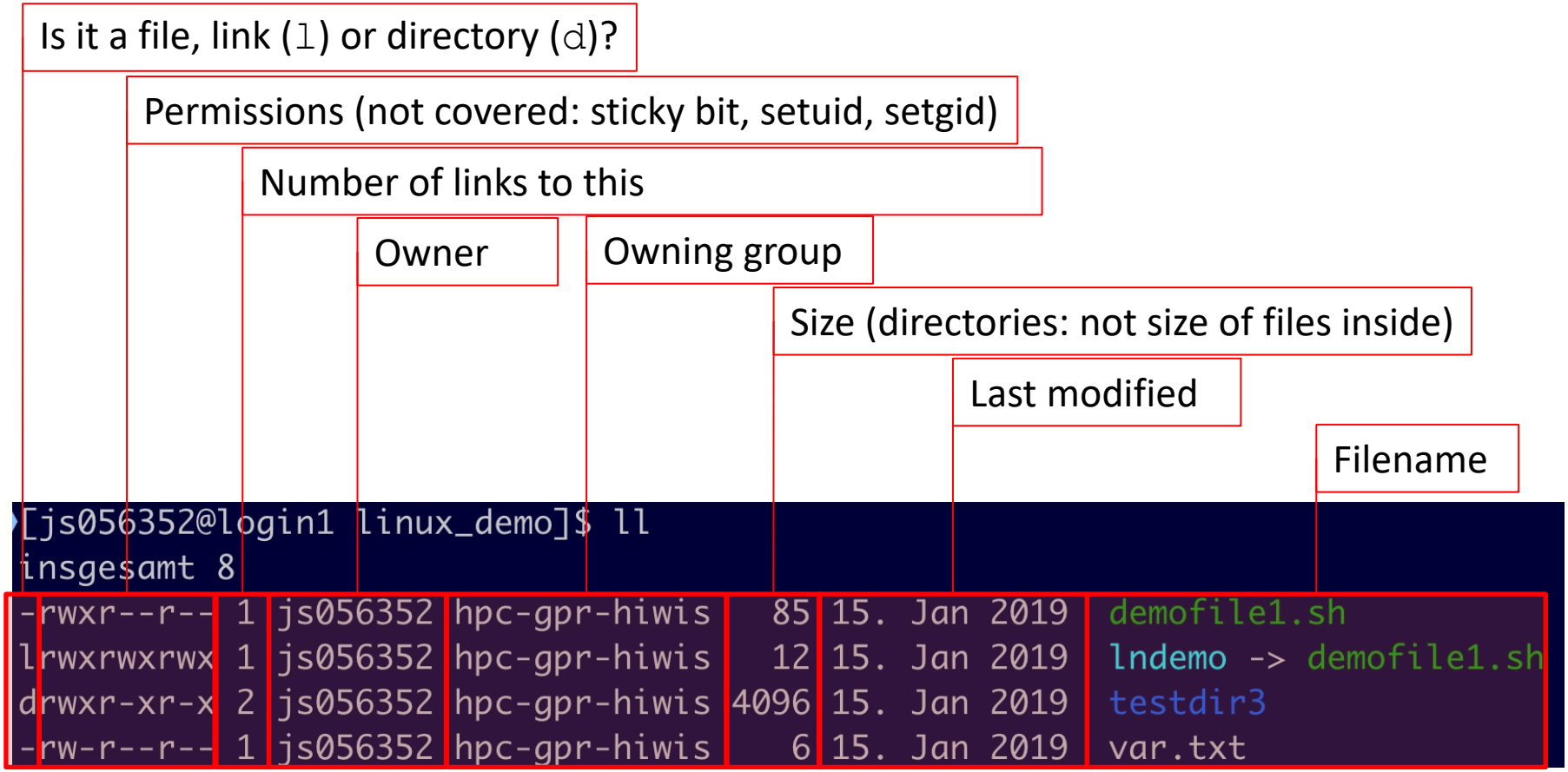
```
sudo <Command>
```

- Users belong to groups
  - Each user has a primary group

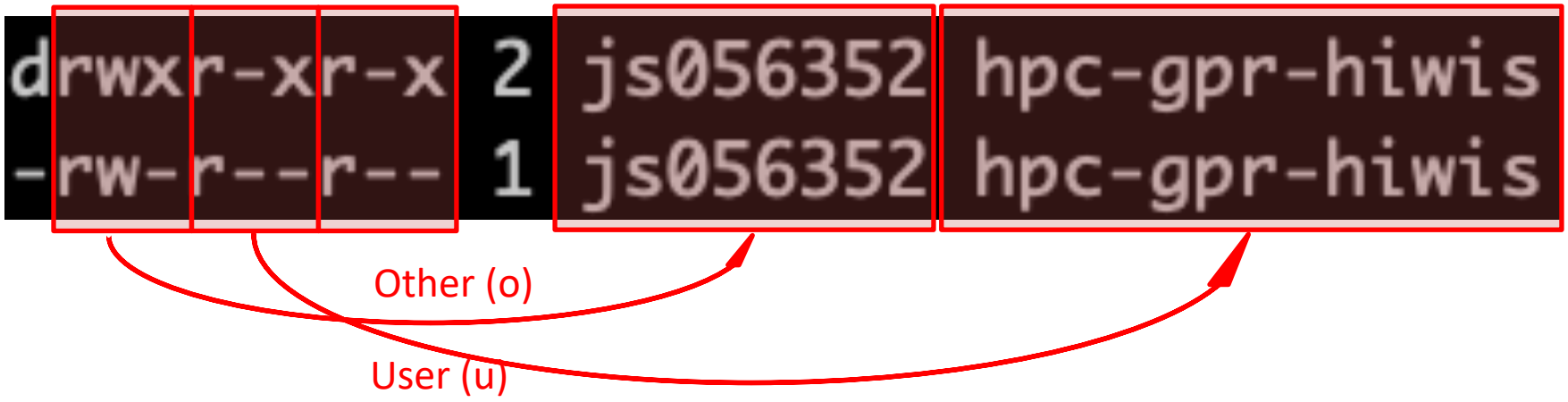
File/directory permissions

- Read:
  - Who can read contents of file/directory
- Write
  - Who can change contents of file/directory
- Execute
  - File: who can execute file (like any program)
  - Directory: who can traverse directory
    - Can execute files inside but not see them

Example output of ls -l



Permissions display explained



- (-) not set
- (r) read
- (w) write
- (x) execute

Changing permissions

- Modify owner/group (needs `root`):
  - `chown <NewOwner> <filename>`
  - `chown <NewOwner>:<NewGroup> <filename>`
- Modify permissions:

  `chmod u+x <Filename>`

  `u`=User, `g`=Group, `o`=Other, `a`=All

  `+` or `−`

  `r`=Read, `w`=Write, `x`=Execute

Demo 8-1

Agenda

1. Introduction
2. Connecting to the cluster
   Exercise 1
3. Historical background
4. The command line
5. Directory structure
   Exercise 2
6. Files
7. Text display, search
   Exercise 3
8. Users and permissions
9. **Processes**
   Exercise 4
10. The vim text editor
    Exercise 5
11. Shell scripting
    Exercise 6
12. Environment variables
13. System configuration files
14. Various tips
    Exercise 7
15. Beyond the cluster

Processes

- Process: running instance of a program
  - System
  - User
  - User (manually launched)
- Like Windows
  - Equivalent to Task Manager: `top`
  - Short overview: `pstree`
- Each process has an owner
  - Process can/can't do what owner can/can't do
- Each process has an ID number (PID)

Demo 9-1

Output and navigation in top

Total resource use

```
top - 11:23:45 up 50 days, 51 min,  9 users,  load average: 3.12, 7.60, 8.63
Tasks: 335 total,   4 running, 331 sleeping,   0 stopped,   0 zombie
%Cpu(s): 22.3 us,  0.9 sy,  0.0 ni, 76.6 id,  0.0 wa,  0.0 hi,  0.3 si,  0.0 st
KiB Mem : 14854156+total, 85480256 free,  2977128 used, 60084180 buff/cache
KiB Swap: 12582908 total, 11918224 free,   664684 used. 14356795+avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
37852 zx057379  20   0    4508    792    588 R 100.0  0.0  21915:30 cl
 6552 gk634     20   0  423588 287224   7180 R  99.7  0.2   0:04.69 cc1plus
14160 gk687     20   0  175052   3124   1328 R  66.4  0.0   9:48.26 sshd
 2355 gk339     20   0 9052548 1.116g 206716 S   7.6  0.8  10:06.16 MATLAB
14162 gk687     20   0   67812   2876   2108 S   6.0  0.0   0:42.50 sftp-server
 2193 gk339     20   0  175956   3552   1272 S   3.3  0.0   1:27.24 sshd
 6444 root      20   0       0      0      0 S   0.3  0.0   0:02.80 kworker/3:1
10801 root      20   0       0      0      0 S   0.3  0.0   0:02.30 kworker/5:0
    1 root      20   0  191612   2964   1556 S   0.0  0.0  11:44.62 systemd
    2 root      20   0       0      0      0 S   0.0  0.0   0:02.66 kthreadd
```

Command name

Process ID

Owner

Resource use

Run time

Output and navigation in top

- Single-letter commands to navigate `top`
  - `u`: filter processes from a specific user
  - `k`: kill a specific process
  - `h`: show help
  - `f`: toggle displayed columns
  - `q`: Quit top
  - See man-page

Demo 9-2

Processes

- If you enter command, it runs in the shell
- Enter `<command> &` to start it in background
  - Good if command launches window, console still usable

- Bring to background by Ctrl-Z (pauses it) and typing `bg`
- Bring to foreground with `fg <Job-ID>`
  - Caution: job ID is different from process ID!
  - Can be displayed with `jobs`

Demo 9-3

Exercise 4

Goals:

- You understand permissions and processes
- You can navigate `top` and understand its output

Tasks:

1. Create an empty file, make it read-only, make it executable
2. Start a process (e.g. `sleep 10m`)
   - Use a second console to look at it in `top`
   - Kill it
3. Start a process, bring it into background/foreground
4. If bored…

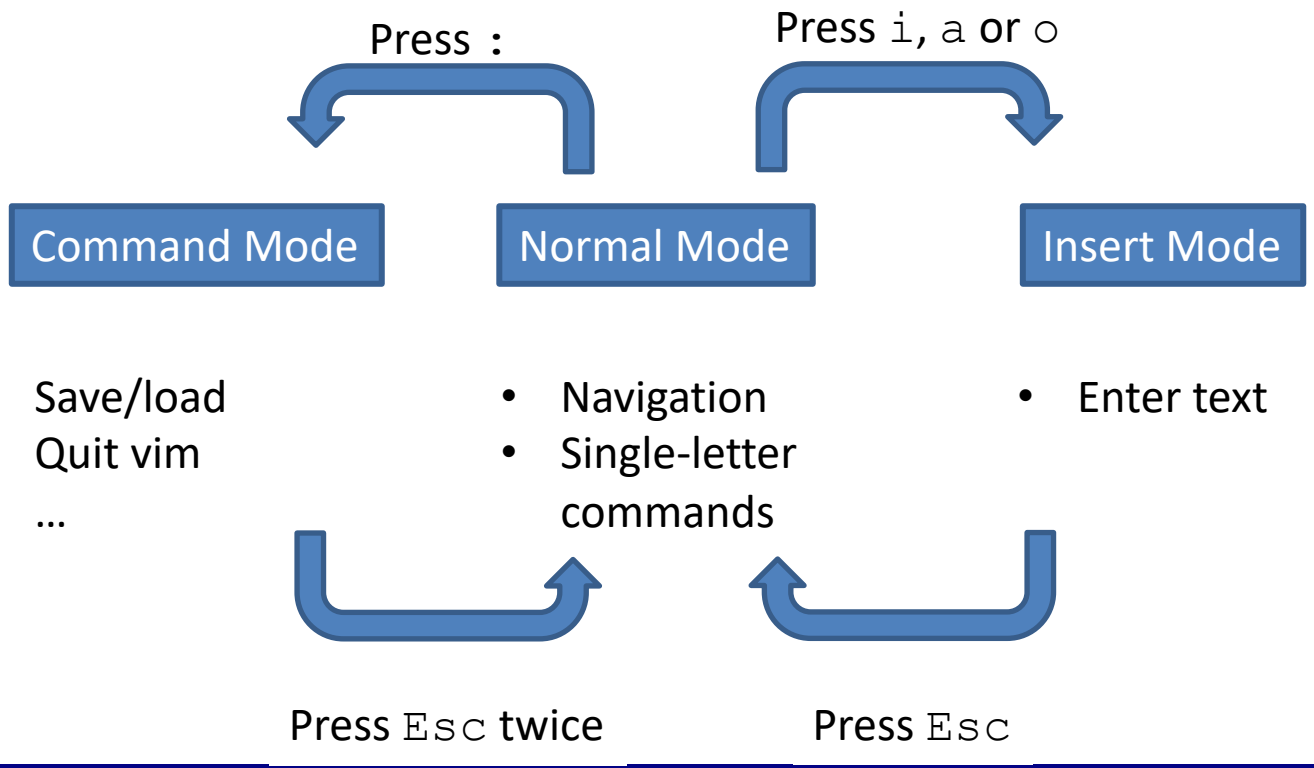
Agenda

1. Introduction
2. Connecting to the cluster
   Exercise 1
3. Historical background
4. The command line
5. Directory structure
   Exercise 2
6. Files
7. Text display, search
   Exercise 3
8. Users and permissions
9. Processes
   Exercise 4
10. **The vim text editor**
    Exercise 5
11. Shell scripting
    Exercise 6
12. Environment variables
13. System configuration files
14. Various tips
    Exercise 7
15. Beyond the cluster

Text editor vim

- Default Linux text editor: `vi`
  - Usually: `vim` (vi improved), includes syntax highlighting
- Completely inside console
- Advantages:
  - Always available
  - Very fast *once you know commands*
- Disadvantages:
  - Interface unlike most text editors
  - Very hard to learn

Text editor vim



Common vim commands

Opening and closing vim:

- `:w` Write (save) file
- `:w <filename>` Write as new filename

- `:wq or :x or ZZ` Write file and quit vim
- `:q!` Close file without saving

Demo 10-1

Common vim commands

Cursor movement:

- `<arrow keys>` Move cursor in arrow direction
- `h, j, k, l` Move cursor left, down, up, right
- `0, $` Move to beginning/end of line
- `gg, G` Move to first/last line of document
- `b, w` Jump back forward one word
- `%` Jump to matching character (e.g. pairs of brackets)

Common vim commands

Editing:

- `u` Undo last change
- `Ctrl+R` Redo last change
- `.` Repeat last command

- `x` Delete character
- `dd` Delete (cut) entire line

- `yy` or `Y` Yank (copy) entire line
- `p` Paste (after cursor)

Demo 10-2

Common vim commands

Search and replace:

- `/pattern` Forward search (for regular expression)
- `?pattern` Backward search

- `n` Find next search result
- `N` Find previous search result

- `%s/old/new/` Replace old pattern with new on current line
- `%s/old/new/g` Replace old pattern with new in entire file

Demo 10-3

Additional thoughts on vim

- Most common vim problem: forgetting which mode you are in
  - Run commands when you meant to type text
  - Remember `u` for undo

When in doubt: keep pressing Esc

- When to use vim:
  - Either only for simple things
  - Or commit to learning it (worth it in the long run)

Otherwise, you will spend a lot of time looking up commands

Alternative text editors

If all else fails, vim usually still works

→ **Knowing vim basics is important for all Linux users**

However I don't blame you if you look for something simpler for everyday use

- Most Linux computers have at least one text editor in addition to vim
  - `gedit` (requires X window connection)
  - `nano`
  - `emacs` (also very powerful and hard to master)
  - Not on cluster but common: `kate` (graphical)
  - MobaXTerm: built-in text editor

Exercise 5

Goals:

- You can navigate `vim`

Tasks:

1. Create a text file with vim, type text in it, save it
2. If you know a programming language, create a source code file of that, see if `vim` can do the syntax highlighting
3. If bored, you know the drill

Agenda

1. Introduction
2. Connecting to the cluster
   Exercise 1
3. Historical background
4. The command line
5. Directory structure
   Exercise 2
6. Files
7. Text display, search
   Exercise 3
8. Users and permissions
9. Processes
   Exercise 4
10. The vim text editor
    Exercise 5
11. **Shell scripting**
    Exercise 6
12. Environment variables
13. System configuration files
14. Various tips
    Exercise 7
15. Beyond the cluster

Shell scripts

- Interaction with Linux: just a series of commands
  - Commands can be put into a text file
  - Text file is fed to console
  - Console runs commands one after the other
- Advantage: very easy automation
- Shell script: execute like a program
  - Remember “execute” permissions

Executing shell scripts

- Command to run script
  - Full script name (including location)
  - Commonly: `./scriptname.sh`
- Why not only script name?
  - Linux only looks up commands in specific folders (environment)
    - Safety feature (not everyone can run everything)
- File needs execute permissions
  - Another safety feature
  - Remember `chmod` command (e.g. `chmod u+x`)

Example shell script

So-called "shebang"
- *Always has to be first line*
- *Comment plus exclamation point*
- *Specifies interpreter (here bash)*
- *Does not have to be Linux console (/usr/bin/python)*

```
#!/bin/bash

# This is a comment line.
echo "Hello world."

ls -l
sleep 3s
ls \
 -l
```

Comment symbol
- *Line comments only*
- *Sometimes meta-commands*

Echo command
- *Common command*
- *Debugging, logging*

List of commands
- *Same as when entered manually*

Line break
Backslash as last character

Demo 11-1

Variables

- Assignment: equals sign
  - Example: `var="value"`
  - Important: no spaces around =
  - Always text
  - Quotes necessary when whitespace or special characters
- Retrieve with `$` sign

  `$var`

  - Example: `echo $var` prints value to screen

Variables

- Common newbie trap: brackets and quotes in variables
  - Single quotes: exact text
  - Double quotes: variables will be expanded
  - Parentheses (round brackets): command inside will be evaluated

`var="bla"` will save text bla to var

`var='$bla'` will save text $bla to var

`var="$bla"` will look for a variable named bla

`var=$(bla)` will execute command bla and save its output to var

Demo 11-2

Shell scripts: additional tips

- Use command line arguments: `$0` - `$9`
  - Example: script was called with `script.sh -f 5.0`
  - Then: `$0=script.sh`, `$1=-f`, `$2=5.0`
- Loops and if statements, similar to most programming languages

```
for file in $( ls ); do
    echo item: $file
done

if [ -e $filename ]; then
    echo „$Filename exists.“
fi
```

Shell scripts: various things

- Shell scripts are good for running series of commands
  - Not so good for more complicated programming
    - Loops, ifs etc. are an afterthought
    - I don't know of an IDE or debugger
    - Can delete wrong file(s) very easily
  - Better: "proper" scripting language (e.g. Python)

- Default shell in most Linux systems (e.g. Ubuntu, CentOS): `bash`
  - Many alternatives: C-Shell (`csh`), Z shell (`zsh`), Fish (`fish`)
    - Often completely different syntax

Exercise 6

Goals:

- You understand environment variables
- You can write and execute a shell script

Tasks:

1. Implement the example script from earlier in this section
2. Add the following features
   - Save the timestamp (the output of the date command) to a variable at the beginning
   - Output the beginning and end timestamps at the end

Note next slide

Exercise 6, optional tasks

If bored, get creative:

- Find a way to execute a script without setting execute permissions
- Find out how to do other programming things in `bash` (e.g. functions, classes)
  - How convenient do they look?
- Look at different ways you can define `if` conditions
- Find out what different types of quotes (single ‘ vs. double “) do

Exercise 6, example solution

```
1  #!/bin/bash
2
3  echo "Hello world"
4
5  olddate=$(date)
6  sleep 10s
7  echo "Old date: "$olddate
8  echo "New date: "$(date)
```

Agenda

1. Introduction
2. Connecting to the cluster
   Exercise 1
3. Historical background
4. The command line
5. Directory structure
   Exercise 2
6. Files
7. Text display, search
   Exercise 3
8. Users and permissions
9. Processes
   Exercise 4
10. The vim text editor
    Exercise 5
11. Shell scripting
    Exercise 6
12. **Environment variables**
13. System configuration files
14. Various tips
    Exercise 7
15. Beyond the cluster

Environment

- "Environment": which variables are defined and available
  - To a process
  - Within a shell
- Avoids hardcoding varying information
- Example: current user's home directory

```
HOME=/home/Schulung12
```

- A program may need to know this directory
  - What if installation directory changes?
  - What if using program on different computer?

Environment

- Many environment variables already defined
  - By system (e.g. `$USER`)
  - By installed software
- Command `env` to show all environment variables
  - Convention: usually capital letters
- Define own environment variables:

```
export MY_VAR="value"
```

  - Available in child processes

Demo 12-1

Excursion: environment modules

- Cluster: different environments for different people
  - Admins cannot predict who needs what
  - Different version of same software: collision of environment variables!
- Solution: make it easy to switch environments
  - Environment modules: sets of environment settings
  - Not limited to clusters

Excursion: environment modules

- Use of modules covered in Cluster Introduction Course
  - Now: what actually happens when module is loaded?

- Each module has a definition file
  - Actually a LUA script

- Let's examine a module file:
  - OpenMPI module (compiled with GCC)

```
$ module show  openmpi/gcc/64/1.10.3
```

Demo 12-2

Excursion: environment modules

- Contains usually at least three things
  - Description what module does
  - Prepend to path and other variables
  - Add new variables

- Anyone want to guess why it prepends rather than appending?

PATH Variable

- Environment variable `PATH`
  - List of directories (separated by :)
  - Console will look for command names
    - Command may be in multiple directories: first hit is used
  - Own commands: add directory to path
- Core concept of operating system
  - Same principle in Windows console
- Also used by other software
  - Example `PYTHONPATH`

Demo 12-3

Agenda

1. Introduction
2. Connecting to the cluster
   Exercise 1
3. Historical background
4. The command line
5. Directory structure
   Exercise 2
6. Files
7. Text display, search
   Exercise 3
8. Users and permissions
9. Processes
   Exercise 4
10. The vim text editor
    Exercise 5
11. Shell scripting
    Exercise 6
12. Environment variables
13. **System configuration files**
14. Various tips
    Exercise 7
15. Beyond the cluster

System information files

- Files in `/proc` are not regular files
  - Text containing system information
  - E.g. `/proc/cpuinfo`, `/proc/meminfo`
  - Display with `cat` or similar
  - Cannot be edited

Demo 13-1

Aliases

- Problem: long command, has to be typed often
  - One option: script (but overkill)
- Built into the shell: aliases
  - Define with `alias name='command'`
  - List with `alias` (no arguments)
- Common aliases:

```
alias ll='ls -l'
alias cd..='cd ..'
```

Demo 13-2

Configuration files

- Console settings usually temporary
  - Environment variables, aliases etc.
  - Adding a directory to PATH
  - Disappear when you close console/disconnect SSH
- Making them permanent: put settings into configuration file
  - Specific files that are read when console is started
  - Examples for Bash:

    `~/.bashrc`

    `~/.bash_profile`

Configuration files

- Other configuration files
  - Example: `~/.vimrc`
- CAUTION WHEN EDITING THESE FILES
  - Breaking `.bashrc` can make it impossible to log in
- Applying changes:
  - Type `source <filename>`
  - Alternative: log out and back in

Demo 13-3

Locales

- Linux determines language and keyboard settings with a so-called locale
- Dictionary definition:

  „Locale (noun): a place or locality, especially with reference to events or circumstances connected with it“

- Grouped into various settings
- See and set with `locale` command
- Sometimes causes weird problems

Example locale output

```
$ locale
LANG=de_DE.UTF-8          # Default for all below variables that are not explicitly set
LC_CTYPE="de_DE.UTF-8"    # Printable characters, used by some C functions
LC_NUMERIC="de_DE.UTF-8"  # Number format (e.g. decimal point or comma)
LC_TIME="de_DE.UTF-8"     # Date and time format
...
LC_ALL=                   # Hard override for all variables above (e.g. for testing)
```

- Output from cluster
  - Some settings omitted for brevity
  - de_DE.UTF-8
    - German language
    - Germany region (as opposed to e.g. Austria)
    - UTF-8 character encoding

Agenda

1. Introduction
2. Connecting to the cluster
   Exercise 1
3. Historical background
4. The command line
5. Directory structure
   Exercise 2
6. Files
7. Text display, search
   Exercise 3
8. Users and permissions
9. Processes
   Exercise 4
10. The vim text editor
    Exercise 5
11. Shell scripting
    Exercise 6
12. Environment variables
13. System configuration files
14. **Various tips**
    Exercise 7
15. Beyond the cluster

Various tips

- You should make backups regularly.
- It is recommended that you back up important files at regular intervals.
- If you break something, you can restore it from a backup that you should have made earlier.
- When working with Linux, be prepared for mistakes, which may require you to use the backups that you hopefully made.
- When you save a copy of a file to a different location regularly with the purpose of copying that file back to the original location to replace the original file after the original file became unusable, that is called a backup, and you should do that.

Various tips

- Useful commands: `du`
  - Shows disk usage
  - Common options: `-h` (human-readable) `-s` (Show total), `-c` (Show individual files)
  - Example: `du -sch .`
- Counterpart: `df`
  - Disk free

Various tips

- Useful commands: `history`
  - Lists previous commands (same as Up-Arrow/Down-Arrow)
  - Text file in your home directory: `~/.bash_history`
  - Advantage: searchable
  - Example: `history | grep <commandname>`
    - When you forgot what options you used

Demo 14-1

Various tips

- Useful commands: `ln -s`
  - Creates a symbolic link
  - Similar to Windows links
  - Visible with `ls -l` or `which`
  - Usage: `ln [Options] <Target> <Link name>`
  - Example: `ln -s myfile.txt mylink`
  - Also possible: “hard links” (not covered here)

Demo 14-2

Various tips

- Useful commands: `watch`
  - Runs target command every 2 seconds
  - Any target command possible
  - Interval modifiable
  - Example: `watch tail mylog.txt` will show what is written to log file
  - Leave with Ctrl+C

Demo 14-3

Various tips

- Useful commands: calculator `$(( ))`
  - For simple integer math
  - Example: `echo $(( 5 + 3 ))`

Various tips

- Stream editor `sed`
  - For simple text operations (e.g. replacing text)
  - Example: `sed -i "s/old/new/g" example.txt`
    - `-i` Edit in place
    - `s` Replace (followed by three-slash syntax)
    - Search text "`old`", replace with "`new`"
    - `g` Repeat for all occurrences in file
- Similar purpose and idea, but more powerful: `awk`
- Both commonly used, I cannot recommend them due to complexity

Exercise 7

Goals:

- You understand system information files
- You understand how to change basic system settings
- You are familiar with various Linux commands

Tasks:

1. Find out the number of CPUs and amount of memory you have
2. Add your script from exercise 6 to the PATH, make an alias for it
3. Permanently change your vim color scheme
4. If bored, *try everything we did in this section*

Agenda

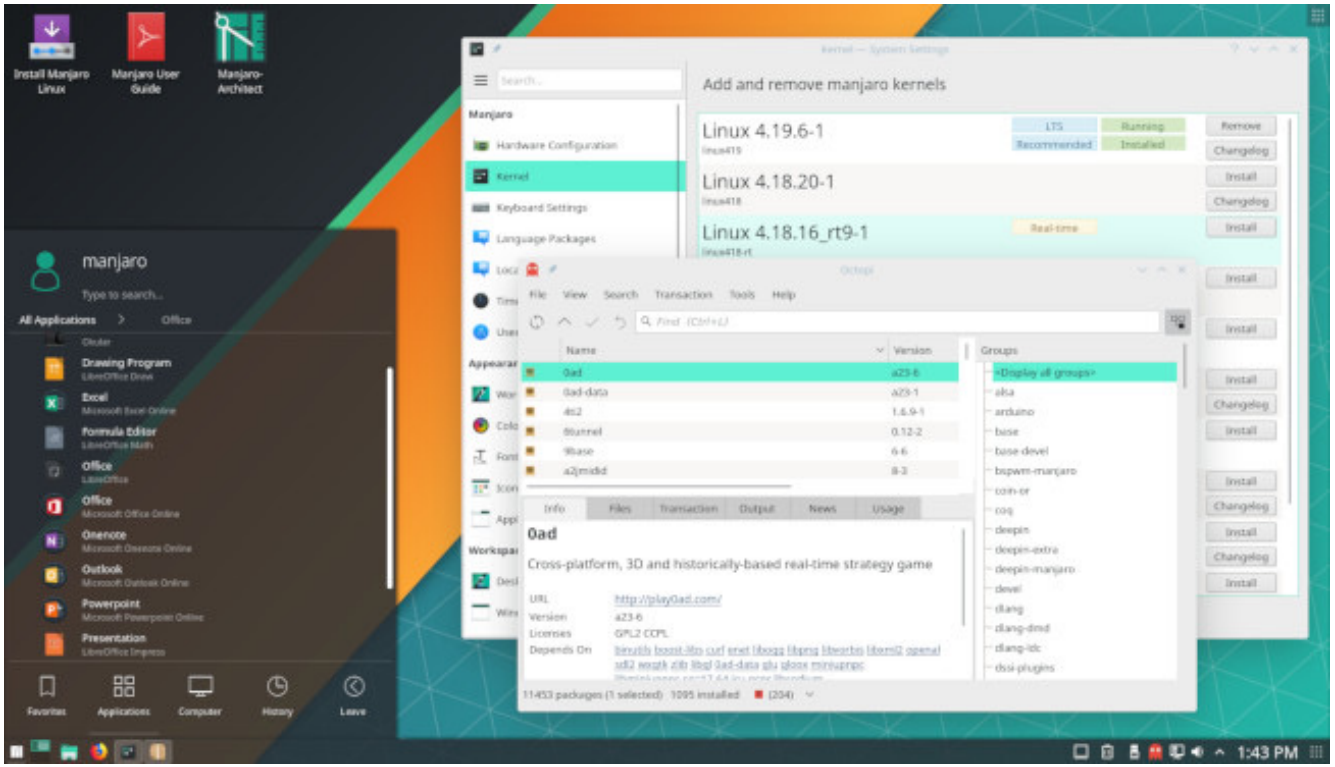
1. Introduction
2. Connecting to the cluster
   Exercise 1
3. Historical background
4. The command line
5. Directory structure
   Exercise 2
6. Files
7. Text display, search
   Exercise 3
8. Users and permissions
9. Processes
   Exercise 4
10. The vim text editor
    Exercise 5
11. Shell scripting
    Exercise 6
12. Environment variables
13. System configuration files
14. Various tips
    Exercise 7
15. **Beyond the cluster**

Graphical User Interfaces

- Key components
  - X Window System
  - Desktop, Graphical User Interface (GUI)
    - Gnome (Ubuntu), KDE, XFCE (Mint)
    - Look is highly distro-dependent

- When remote: X server
  - Displays windows from other computer (cluster)
  - Careful with wording: server is on your machine, client is program that runs on cluster

Example desktop (KDE on Manjaro)



Most features should look familiar to users of Windows and other OSes

Package managers

- Software is often installed as packages
  - Organized in internet repositories
- Distro-dependent
  - Often maintain their own repository
- Not possible on cluster (exception: inside of application, e.g. Python, R)
- In general, three different package managers:

  `apt-get` (Debian family), package format `.deb`

  `yum` (Red Hat family), package format `.rpm`

  `zypper` (Suse), package format `.rpm`

Wrap-up

- Test user accounts on cluster will be reset
  - Copy everything over that you want to keep
- Feedback round after this
- This course, like most ZIMT courses, will be repeated every semester
  - Alternating between German and English

Thank you for your attention

Feedback round

- What were your expectations, and where they fulfilled?
- What was your favorite part about the course?
- What did you dislike or what do you feel can be improved?
- How did you learn about this course?
- What other topics would you like to see in future ZIMT courses?