HPC.NRW

# INTRODUCTION TO LINUX

(in an HPC context)

Version 20.09 | HPC.NRW Competence Network

THE COMPETENCE NETWORK FOR HIGH-PERFORMANCE COMPUTING IN NRW.

# BACKGROUND AND HISTORY

HPC.NRW Competence Network

# INTRODUCTION TO LINUX

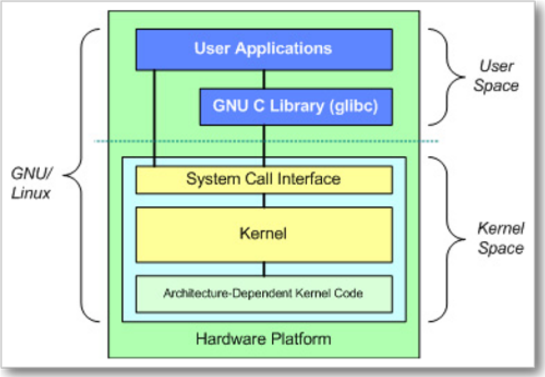
# HISTORICAL BACKGROUND

- 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

# HISTORY OF LINUX

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



# SOME IMPORTANT DISTROS

- High reliability (e.g. servers): Red Hat Enterprise Linux
  - Developer “playground”: Fedora
  - Community variant: CentOS (HoRUS 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:
  - **500 out of the Top 500 supercomputers (2020)**
  - (Web) servers: 95 %
  - Mobile devices: 60 – 80 % of mobile devices (almost all Android)
  - Desktop PCs: 1 – 2 %

- Popular desktop distros (no good figures):
  - Ubuntu
  - Linux Mint

HPC.NRW

# INTRODUCTION TO LINUX

(in an HPC context)

Version 20.09 | HPC.NRW Competence Network

THE COMPETENCE NETWORK FOR HIGH-PERFORMANCE COMPUTING IN NRW.

# THE COMMAND LINE

HPC.NRW Competence Network

## INTRODUCTION TO LINUX

# COMMAND LINE

- A line where you type commands
- Other terms:
  - CLI (command line interface)
  - Console / Terminal
  - Shell
- Advantages
  - Simple (nearly always works)
  - Easy to program (in comparison to GUI)
  - Fast and efficient to use (if you know the commands)
- Disadvantage: lots of memorizing (vs. GUI buttons)

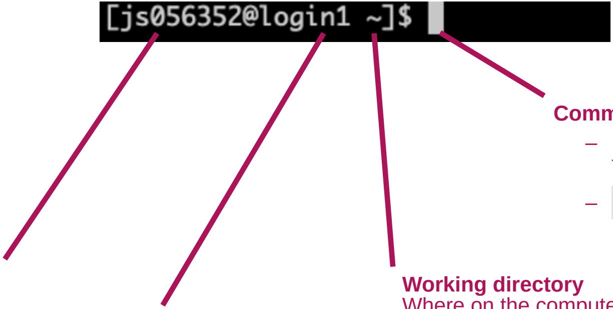
# WHY THE NAME "SHELL"

- 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

# COMMON ELEMENTS OF THE CONSOLE PROMPT STATEMENT



```
[js056352@login1 ~]$ 
```

**Command prompt**

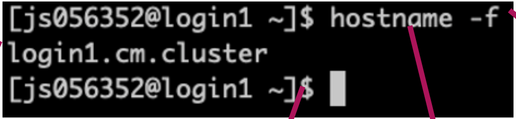
- Indicates system is ready to receive input
- `$`: convention (also `>`)

**User name**
Who is using this

**Host name**
What computer are you on?

**Working directory**
Where on the computer are we?

# COMMON ELEMENTS OF THE CONSOLE PROMPT STATEMENT



```
[js056352@login1 ~]$ hostname -f
login1.cm.cluster
[js056352@login1 ~]$ 
```

**Options**

- Command-specific
- Several conventions

**New command prompt**
Execution completed

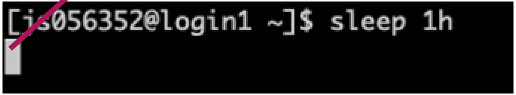
**Command**

- What to do
- Typed by user

**Output**

- What the command returns
- No rules

Running



```
[js056352@login1 ~]$ sleep 1h

```

# NAVIGATING THE COMMAND LINE

**`<Enter>`** run command

**`<Up-Arrow>`** navigate command history back in time

**`<Down-Arrow>`** navigate command history forward in time

**`<Tab>`** auto-completion (a.k.a. “tab completion”)

- If enough letters to identify command: completes command
- More than one possibility: nothing shown
- Second Tab to list possible completions

**`<Ctrl-C>`** abort current command.

# COMMAND LINE CONVENTIONS

- Always case-sensitive
  - Popular source for errors

- Command line options:
  - Usually start with dash
  - Common convension: single dash for “short options”, double dash for “long options”
    - Example: `sbatch --time 0:30:00` is identical to
      `sbatch -t 0:30:00`
  - Note: specific commands may deviate from convention

# 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

# 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

HPC.NRW

# INTRODUCTION TO LINUX

(in an HPC context)

Version 20.09 | HPC.NRW Competence Network

THE COMPETENCE NETWORK FOR HIGH-PERFORMANCE COMPUTING IN NRW.

# LINUX DIRECTORY STRUCTURE

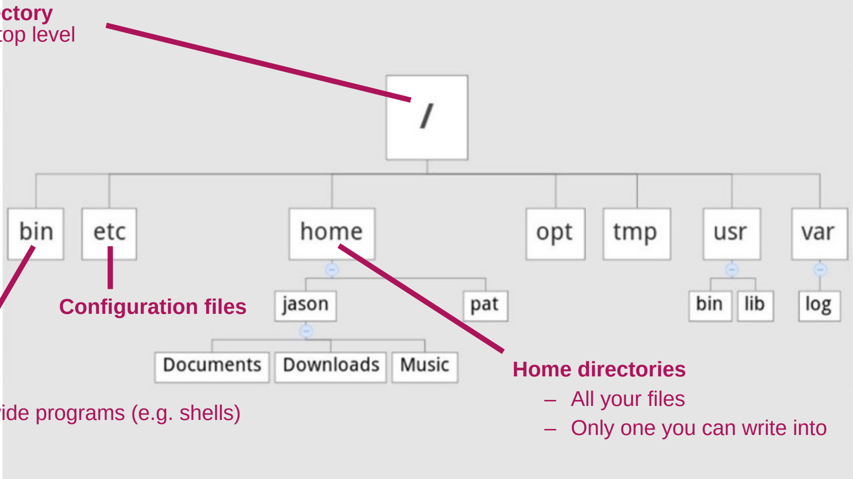
HPC.NRW Competence Network

# INTRODUCTION TO LINUX

# 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: `/home/bob/Documents/pdf`
  - Absolute path (starts with `/`)
  - Relative path: relative to (current) **working directory**
- Print working directory: `pwd`

# EXAMPLE DIRECTORY STRUCTURE



**Root directory**
Absolute top level

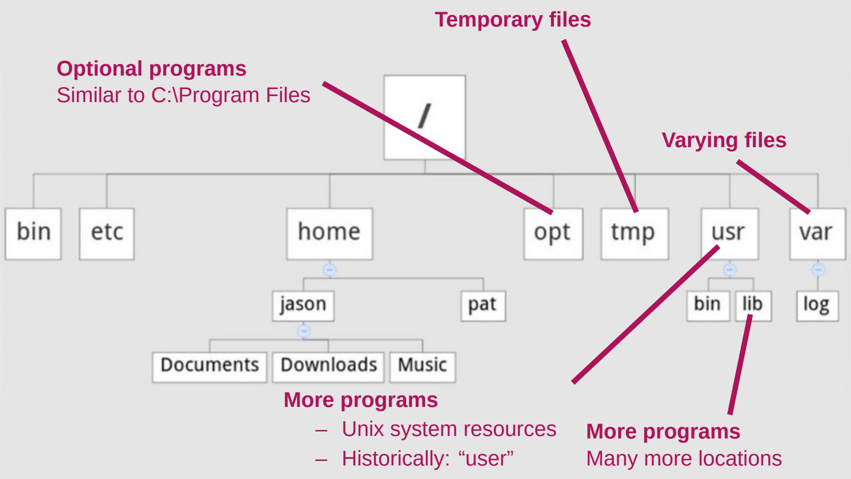
**Configuration files**

**Home directories**

- All your files
- Only one you can write into

**Binaries**
System-wide programs (e.g. shells)

# EXAMPLE DIRECTORY STRUCTURE



**Temporary files**

**Optional programs**
Similar to C:\Program Files

**Varying files**

**More programs**

- Unix system resources
- Historically: “user”

**More programs**
Many more locations

# 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 mistake: `cd..` (no space in between)
  - Often defined as an alias to mitigate typo

# 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

# CONSOLE-SPECIFIC COMMANDS AND SHORTCUTS

**`<Middle Mouse>`** paste selected 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 when on an empty command line

- Quit console with `exit` (SSH connection: back to local console)
- Clear screen: `clear` command

HPC.NRW

# INTRODUCTION TO LINUX

(in an HPC context)

Version 20.09 | HPC.NRW Competence Network

THE COMPETENCE NETWORK FOR HIGH-PERFORMANCE COMPUTING IN NRW.

# FILES

HPC.NRW Competence Network

# INTRODUCTION TO LINUX

# 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
- Use `file <filename>` to identify file type

# 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

# BASIC FILE MANIPULATION

- Create directory: `mkdir <dirname>`
- Create empty file: `touch filename`
  - Updates access time on an existing file
- Remove file/directory: `rm <filename>`
  - Check access permissions!
  - To delete content of subdirectories: `rm -r` (recursive)
  - Common option: `-f` (force) → never prompt for confirmation

# WILD CARDS

- Previous examples: one command, one file
- Select multiple files according to patterns
- Wildcard (placeholder) characters
  - Also called globbing
- Most important
  - `*` zero or more characters
  - `?` exactly one character
  - `[]` range of characters

# 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

HPC.NRW

# INTRODUCTION TO LINUX

(in an HPC context)

Version 20.09 | HPC.NRW Competence Network

THE COMPETENCE NETWORK FOR HIGH-PERFORMANCE COMPUTING IN NRW.

# TEXT DISPLAY AND SEARCH

HPC.NRW Competence Network

# INTRODUCTION TO LINUX

## CONSOLE INPUT AND OUTPUT

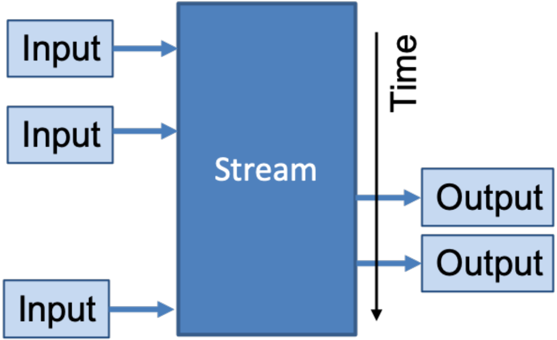
- Console has three main ways of communicating with process (so-called streams)
  - Standard input (`stdin`)
  - Standard output (`stdout`)
  - Standard error (`stderr`)

- `stdin`: what you type into console

- `stdout` + `stderr`: what you see in console
  - Two separate streams so you can separate error messages from normal output

# WHY THE TERM "STREAM"

- What is a “stream” in computing terms?
  - Intermediate storage
  - Input and output may overlap
- Example: streaming video
  - Video gets partially downloaded, you can already view it
- 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 stderr

```
command 2> 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` 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
    ```

# 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
  - Others:
    - `head`: display first lines
    - `tail`: display last lines

# 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?

HPC.NRW

# INTRODUCTION TO LINUX

(in an HPC context)

Version 20.09 | HPC.NRW Competence Network

THE COMPETENCE NETWORK FOR HIGH-PERFORMANCE COMPUTING IN NRW.

# USERS AND PERMISSIONS

HPC.NRW Competence Network

# INTRODUCTION TO LINUX

# 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**

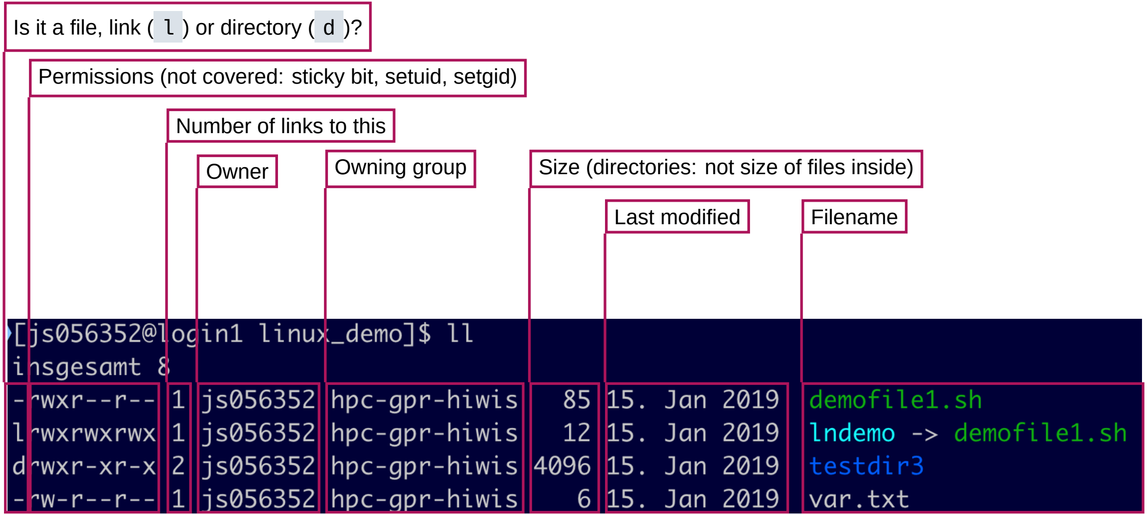
- Determines what you can do
  - *You can't break what you can't use!*
- `root` user (superuser) can do everything
- Users may get temporary root permissions

  ```
  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



Is it a file, link ( `l` ) or directory ( `d` )?

Permissions (not covered: sticky bit, setuid, setgid)

Number of links to this

Owner

Owning group

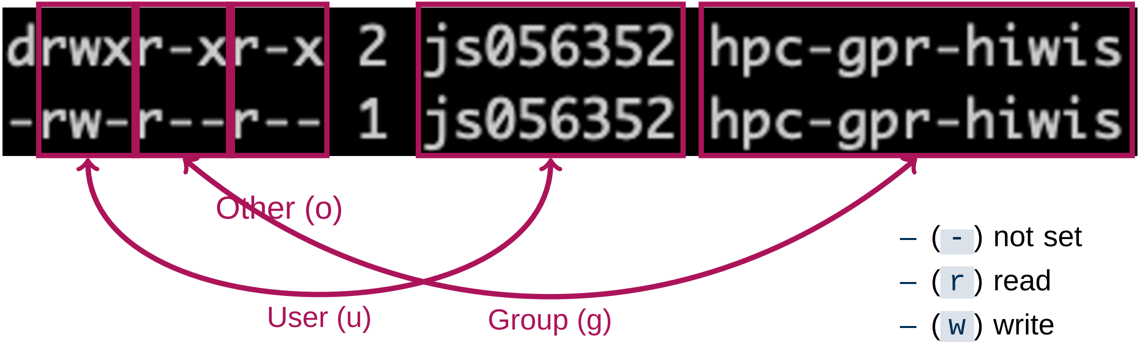
Size (directories: not size of files inside)

Last modified

Filename

```
[js056352@login1 linux_demo]$ ll
insgesamt 8
-rwxr--r-- 1 js056352 hpc-gpr-hiwis   85 15. Jan 2019 demofile1.sh
lrwxrwxrwx 1 js056352 hpc-gpr-hiwis   12 15. Jan 2019 lndemo -> demofile1.sh
drwxr-xr-x 2 js056352 hpc-gpr-hiwis 4096 15. Jan 2019 testdir3
-rw-r--r-- 1 js056352 hpc-gpr-hiwis    6 15. Jan 2019 var.txt
```

# PERMISSIONS DISPLAY EXPLAINED



```
drwxr-xr-x 2 js056352 hpc-gpr-hiwis
-rw-r--r-- 1 js056352 hpc-gpr-hiwis
```

Other (o)

User (u)

Group (g)

- ( `-` ) 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

HPC.NRW

# INTRODUCTION TO LINUX

(in an HPC context)

Version 20.09 | HPC.NRW Competence Network

THE COMPETENCE NETWORK FOR HIGH-PERFORMANCE COMPUTING IN NRW.

# PROCESSES

HPC.NRW Competence Network

# INTRODUCTION TO LINUX

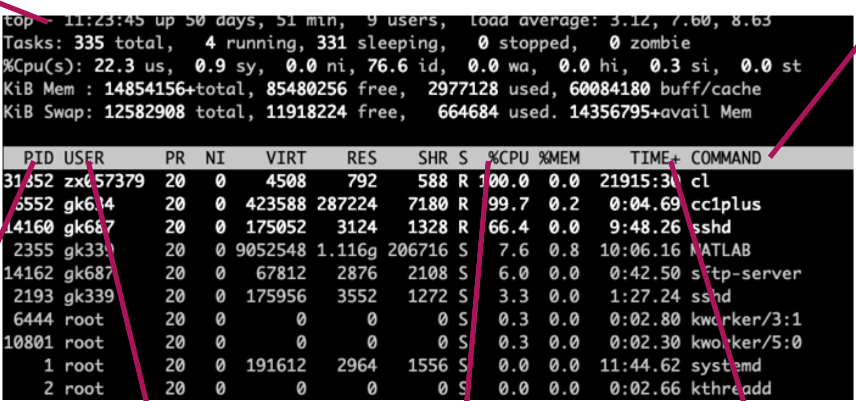
# PROCESSES

- 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)

# OUTPUT AND NAVIGATION IN TOP

Total resource use

Command name



```
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
31352 zx057379  20   0    4508    792    588 R 100.0  0.0  21915:30 cl
 5552 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
```

Process ID

Owner

Resource use

Runtime

# 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
  - `x`: highlight current sort columnt
  - `<>`: select column to sort for
  - `R`: Reverse sorting
  - `q`: quit top

See man-page

# 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

- Send foreground command 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`

HPC.NRW

# INTRODUCTION TO LINUX

(in an HPC context)

Version 20.09 | HPC.NRW Competence Network

THE COMPETENCE NETWORK FOR HIGH-PERFORMANCE COMPUTING IN NRW.

# THE VIM TEXT EDITOR

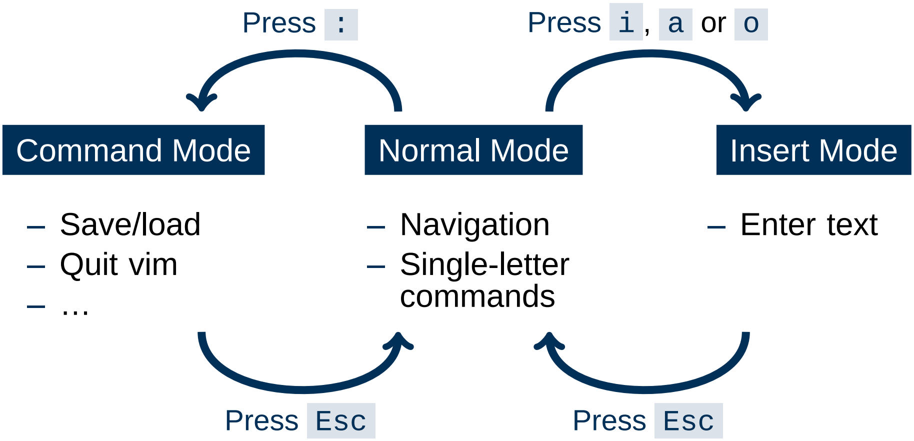
HPC.NRW Competence Network

# INTRODUCTION TO LINUX

# 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
  - Steep learning curve

# TEXT EDITOR VIM



Press `:`

Press `i`, `a` or `o`

**Command Mode**

- Save/load
- Quit vim
- …

**Normal Mode**

- Navigation
- Single-letter commands

**Insert Mode**

- Enter text

Press `Esc`

Press `Esc`

# COMMON VIM COMMANDS (OPENING AND QUITTING)

Opening: either `vim` or `vim <filename>`

**`:w`** Write (save) file

**`:w <filename>`** Write as new filename

**`:wq`** **or** **`:x`** **or** **`ZZ`** Write (save) file and quit

**`:q!`** Close file without saving

# COMMON VIM COMMANDS (MOVEMENT)

**<arrow keys>** move cursor in arrow direction

`h`, `j`, `k`, `l` move cursor left, down, up, right

`$` Move to end of line

`gg` Move cursor to first line

`G` Move cursor to last line

`w` Jump forward to next word

`b` Jump backward to previous word

`%` Jump to matching character (default pairs: `()`, `{}`, `[]`)

# COMMON VIM COMMANDS (EDITING)

HPC.NRW

`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

# COMMON VIM COMMANDS (SEARCHING)

`/pattern` Forward search for regular expression

`?pattern` Backward search for regular expression

`n` Repeat last search

`N` Repeat last search in opposite direction

`%s/old/new/` Replace `old` pattern with `new` pattern on current line

`%s/old/new/g` Replace `old` pattern with `new` pattern in entire file

# 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

HPC.NRW

# INTRODUCTION TO LINUX

(in an HPC context)

Version 20.09 | HPC.NRW Competence Network

THE COMPETENCE NETWORK FOR HIGH-PERFORMANCE COMPUTING IN NRW.

# SHELL SCRIPTS

HPC.NRW Competence Network

# INTRODUCTION TO LINUX

# 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

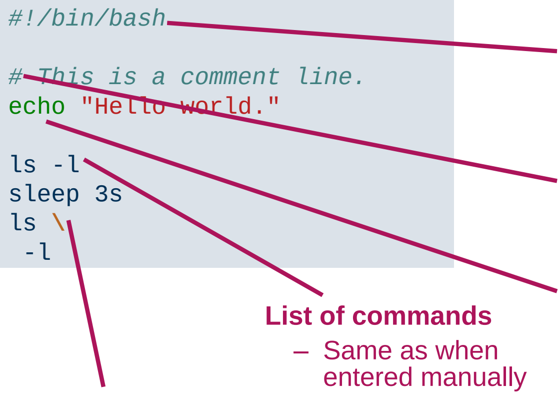
- Advantage: very easy automation

- 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
    - 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



```
#!/bin/bash

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

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

**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)

**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

# VARIABLES

- Store output of commands
- Assignment via `=` (equal sign)
  - Example: `var="value"`
  - Important: no spaces around `=`
  - Always text
  - Quotes necessary when whitespace, special characters in value
- 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 the text bla to var
- `var=‘$bla’` will save the 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

# SHELL SCRIPTS: ADDITIONAL TIPS

- Use command line arguments: `$0` - `$9`, `${10}`
  - 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 complex 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
    - Prefer portable shell programming where possible

HPC.NRW

# INTRODUCTION TO LINUX

(in an HPC context)

Version 20.09 | HPC.NRW Competence Network

THE COMPETENCE NETWORK FOR HIGH-PERFORMANCE COMPUTING IN NRW.

# ENVIRONMENT VARIABLES

HPC.NRW Competence Network

## INTRODUCTION TO LINUX

# 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
```

- Helpful to provide configuration scripts
  - Change information in a single location
  - Keep business logic apart from config

# ENVIRONMENT

- Many environment variables already defined
  - By system (e.g. `$USER`)
  - By installed software

- Command `env` to show all currently defined variables
  - Convention: usually capital letters

- Passing on environment variables:

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

  - Available in child processes

# 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

- Example: OpenMPI module (compiled with GCC)

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

# EXCURSION: ENVIRONMENT MODULES

- Each module has a definition file
  - Actually a LUA or Tcl script
- Contains at least three things
  - Description what module does
  - Prepend to path and other variables
  - Add new variables
- Usually 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`

HPC.NRW

# INTRODUCTION TO LINUX

(in an HPC context)

Version 20.09 | HPC.NRW Competence Network

THE COMPETENCE NETWORK FOR HIGH-PERFORMANCE COMPUTING IN NRW.

# SYSTEM CONFIGURATION

HPC.NRW Competence Network

# INTRODUCTION TO LINUX

# 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

# 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 ..'
```

# 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

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

HPC.NRW

# INTRODUCTION TO LINUX

(in an HPC context)

Version 20.09 | HPC.NRW Competence Network

THE COMPETENCE NETWORK FOR HIGH-PERFORMANCE COMPUTING IN NRW.

# SSH CONNECTIONS

HPC.NRW Competence Network

# INTRODUCTION TO LINUX

# SSH CONNECTIONS

- Clusters typically accessed via Secure Shell (SSH) protocol
- Most commonly OpenSSH software
- Available for all operating systems
  - Linux: original
  - Mac OS: basically identical
  - Windows 10 (since 2019): integrated in cmd/Powershell
- Additional tools, especially on Windows: Putty, MobaXTerm

# SSH BASIC USE

- Connect with `ssh` command: `ssh [options] <username>@<hostname>`
- You will be asked for password
  - Alternative: set up public/private key pair
- Can specify configurations to simplify login
- Console-based, but opening windows is possible
- Multiple simultaneous connections possible

## SSH CONFIGURATION

- OpenSSH allows setting presets

- Directory `~/.ssh` contains config file
  - Simply named `config`
  - Editable text file

- One preset per cluster
  - Specify username
  - Other options (many possibilities)

- Use `ssh <presetname>` instead of `ssh [options] <user>@<host>`

# SSH KEY-BASED AUTHENTICATION

- Login with public/private key pair instead of password
- Convenient
  - Good for automated connections
- Potentially more secure
- Only as secure as your PC
  - **Treat private key file like a physical key**

# KEY PAIR WORKFLOW

- You generate key pair
  - On your PC
  - Tool `ssh-keygen` (comes with OpenSSH)

- You copy public key to cluster
  - `ssh-copy-id` (comes with OpenSSH)
  - Windows: manually copy and paste key

- When logging in, OpenSSH will select key

# KEY GENERATION

- Run SSH key generator
  - On local PC, type `ssh-keygen`
  - Enter filename for new key
    - Should be inside `~/.ssh` directory
    - Caution: will overwrite without asking
  - Enter passphrase
    - Can be left empty, but not recommended
  - Confirm passphrase

# COPYING KEY TO CLUSTER

- On local PC, use the `ssh-copy-id` command
  - Syntax: `ssh-copy-id -i <keyfile> <user>@<host>`
  - Not available in Windows

- Alternative: copy manually
  - On local PC, open public key file with text editor
  - One line of text, three parts: algorithm, key, comment
  - On cluster, open `~/.ssh/authorized_keys`
  - Paste line, adjust comment as needed

# LOGGING IN WITH KEY

- When logging in, key will be used automatically
  - May specify key file manually if needed (option `-i`)
  - If you get asked for password, key was not recognized

- Tips:
  - Use one key per PC (in case of theft/compromise)
  - Not recommended to leave passphrase empty
    - But only needs to be entered once

HPC.NRW

# INTRODUCTION TO LINUX

(in an HPC context)

Version 20.09 | HPC.NRW Competence Network

THE COMPETENCE NETWORK FOR HIGH-PERFORMANCE COMPUTING IN NRW.

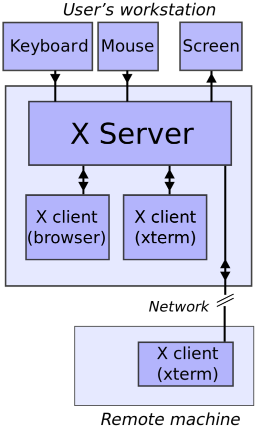
# SSH: GRAPHICS AND FILE TRANSFER

HPC.NRW Competence Network

# INTRODUCTION TO LINUX

# GRAPHICAL USER INTERFACES

- X Window System
- Basis of all Linux displays
- Can also display windows from other computers
- X server needs to run on PC
- X client is software that window belongs to



# GRAPHICS VIA SSH

- Requirements
  - X server installed on PC
  - SSH connection with X support
  - (Cluster supports X windows)
- Available for all OSes
  - Linux: X server built in
  - Mac OS: XQuartz
  - Windows: xming, MobaXTerm

# CONNECTING WITH X SUPPORT

- Enable X support in SSH
  - `ssh -X <user@<host>`
  - Must be upper case X
- Sometimes `-Y` is used
  - “Trusted” connection
  - Less safe, sometimes necessary for things to work
- In config file: `ForwardX11 yes` or `ForwardX11Trusted yes`

# FILE TRANSFER

- Copying files between PC and cluster
  - Use `scp` command (secure copy)
- Syntax similar to Linux `cp` command
- Uses SSH, can use same settings/presets
- Console-based, many third-party frontends for all OSes
  - WinSCP, SSHFS, MobaXTerm

# FILE TRANSFER

- Syntax:

```
scp [options] sourcehost:sourcefile targethost:targetfile
```

  - Host may be left out if local
  - Host may be SSH preset
  - Source or target or both can be remote

- Same rule as `cp` about `-r` when copying entire directories

- Unlike `cp`: will print status of file transfer to screen

- Not only possibility (`rsync`)

HPC.NRW

# INTRODUCTION TO LINUX

(in an HPC context)

Version 20.09 | HPC.NRW Competence Network

THE COMPETENCE NETWORK FOR HIGH-PERFORMANCE COMPUTING IN NRW.

# VARIOUS TIPS

HPC.NRW Competence Network

# INTRODUCTION TO LINUX

# 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 forget what options you used

# VARIOUS TIPS

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

# 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

# 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

# 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`