Introduction to the Linux OS

Peter Huszár
KFA: Department of Atmospheric Physics

Pavel Řezníček
ÚČJF: Institute of particle and nuclear physics

November 18, 2025

Overview and Organization

Introduction to the Operation system Linux, focus on the command line, scripting, basic services and tools used in (not only) physics: tasks automation in data processing and modeling

Organization

- Graded Assessment (KZ): attendance to the lectures, worked out homeworks

Literature

- C. Herborth: Unix a Linux - Názorný průvodce, Computer Press, Praha, 2006
- D. J. Barrett: Linux - Kapesní přehled, Computer Press, Praha, 2006
- M. Sobell: Mistrovství v RedHat a Fedora Linux, Computer Press, Praha, 2006
- M. Sobell: Linux - praktický průvodce, Computer Press, Praha, 2002
- E. Siever: Linux v kostce, Computer Press, Praha, 1999
- Number of online sources...

Study materials and homeworks

- http://kfa.mff.cuni.cz/linux

KFA MFF UK

ÚČJF MFF UK

Syllabus

1. UNIX systems, history, installation, basic applications
2. Structure of the Linux OS, file systems, hierarchy of the file system
3. Command line, shells, remote access (ssh, ftp)
4. Processes and their administration, basic system commands, packages, printing
5. Users, file and directory permissions
6. Work with files and directories, file compression, links, partition
7. Text-file processing commands, redirection, pipeline
8. Regular expressions
9. Command line based text editors
10. User and system variables, output processing
11. Scripts: basic construction, conditionals, loops, functions, automation
12. Networking, server-client services: http, (s)ftp, scp, ssh, sshfs, nfs
13. Programming in Linux (examples of Fortran, C/C++, Python), version control systems, documents in Latex

Text manipulation

Basic text manipulation commands

Commands to view and transform text(files); and to extract information about the text

- cat, more and less - to view text file contents (read-only!)

```
cat /my/text/file.txt # writes the contents of the file
                      # on the terminal - the standard output (stdout) and returns to prompt
                      # seemingly useless command in case of long files, but wait-for-it;-)
                      # (e.g. for putting two file ater "each other"  cat files1 file2 > file_final)
more /my/text/file.txt # view text by pages ("Space"), q or ctrl-c to quit
less /my/text/file.txt # view text by pages, PageUp/PageDown,Up/Down keys, Space -- 'q' to quit
                       # less is more than more and unlike more, reads only that part of the file
                       # which is shown
```

- head and tail - we are oftne interested in what the begining/ending of text file contains

```
head -n 20 /my/text/file.txt # prints the first 20 lines of the textfile
head -n -20 my/text/file.txt # prints all but the last 20 lines
tail -n 20 /my/text/file.txt # prints the last 20 lines of the textfile
tail -n +20 /my/text/file.txt # prints the last lines starting with line number 20
                              # if -c is used instead -n in both cases
                              # printing applies for bytes (first N/last N bytes)
```

- wc - word count (counts bytes/characters/words/lines)

```
wc  /my/text/files.txt # Print newline, word, and byte counts for each file
                       # if more files provided, print info for each and the total
wc -l /my/text/files.txt # number of lines
wc -w /my/text/files.txt # number of words
wc -c /my/text/files.txt # number of bytes, -m number of characters (bytes and chars are not the same)
```

Basic text manipulation commands (cont'd)

Commands to view and transform text(files)

- cut - cuts parts of each line of a text ("vertical" version of head/tail). It uses "deliminator", default is the tabulator. Very useful to extract columns in a text file, where columns are separated by a certain character (e.g. comma)

```
cut -d"{deliminator}" -f1 /my/text/file.txt # for each line prints everything before the first appearance
                                            # of the {deliminato} when it is the tabulator, -d is not import
cut -d":" -f1,3,5,10-15 # takes the first column, the 3rd, the 5th and than the 10-15.
                        # --complement, select the rest
                        # -d".", -d";", -d" ", -d"-", -d"a" etc.
cut -c X-Y              # cut from Xth to the Yth character on each line of the input file
```

- paste - line-by-line merge files "next" to each other. This requires a delimiter, tabulator is again the default one

```
paste -d"{deliminator}" /my/text/file1.txt /my/text/file1.txt  # for each line1/2 of file1/2
                                                               # prints line1/2 next to each other
                                                               # separated by {deliminator}
```

- fmt - simple text formatter

```
fmt /my/file.txt # by default puts all words  in a single line and prints
fmt -w 10 /my/file.txt # print the text in the specified width (does not wrap words!!!)
fmt -t file.txt # add indentation for the first line different from others
fmt -u file.txt # uses one space between words and two spaces after sentences for formatting
```

Basic text manipulation commands (cont'd 2)

- sort - sort lines of file according to dictionary,numerical value etc.

```
sort -d /my/files.txt # dictionary sort
sort -b /my/files.txt # ignore leading blank characters
sort -n /my/file-with-numbers.txt # numeric sort
sort -r /my/file.txt # reverse the result
sort -u /my/file.txt # print only unique lines
sort -k 2 /my/file.txt # sort according to the second "column"
```

- uniq - report/omit repeated lines (related to sort -u)

```
uniq -c /my/file.txt # prefix lines by the number of occurrences
uniq -d/-u /my/file.txt # prints only duplicated lines/unique lines
```

- rev - a trivial utility that reverses each line characterwise

```
rev /my/file.txt # the output will be the file with same size but all lines in
                 # reversed order (you can doubt the usefullness :-))
```

- join - join lines of two files on a common field (joins two "column" files with different columns as fields delimited by a delimiter based on values in selected columns)

```
join -t, -1 FILE1COLUMN -2 FILE2COLUMN FILE1 FILE2 # joins lines from both file based on
                                                   # the value of FILE1COLUMN equals FILE2COLUMN
join -t, -1 1 -2 2 file1 file2 # join lines of file1 with file2 where the value of columns 1 from
                               # 1st file and column 2 from 2nd file equal;-t is the delimiter (e.g. -t,)
```

- For join to be successfull, files must be sorted, unless –nocheck-order is used
- –headers will skip the first line on each file as header line

Input/output redirection and command chaining

Input/output redirection

standard output, standard error output, standard input

When a linux command is launched, three data streams are relevant/created: `stdout`, `stderr`, `stdin`, the standard output, what we saw so far in the terminal, the standard error, what we also saw in the terminal if an error occured and standard input what is read in.

- Redirect the stdout and stderr to a file

```
any-linux-command [1]> /my/output.log # the stdout is saved to file /my/output.log,
                                      # if exists, rewrites it (.log is not neccesary!)
                                      # 1 is the default value, so does not need to be written
any-linux-command 2> /my/output_errors.log # the stderr is saved to file /my/output_errors.log
any-linux-command &> /my/output_all.log # the stdout+stderr is saved to file /my/output_all.log
any-linux-command 1> /my/output.log 2> /my/output_errors.log # split the output into two files
any-linux-command 2>&1 # everything comes out as stdout (can be redirected further)
any-linux-command >> /my/output.log # if output.log exists, it will append the output to existing content
                                    # can be used with 1,2,&
# a command without redirection is equivalent to any-linux-command 1> /dev/stdout  2> /dev/stderr
Flush the output: any-linux-command > /dev/null
```

- Redirecting from the standard input

```
any-linux-command <  /my/input # the command 'any-linux-command' will take the file
                               # /my/input as input (e.g. rev < /my/input.txt will
                               # reverse all lines of /my/input.txt and writes it out)
any-linux-command <( other_command ) # redirect the output of other command to any-linux-command
any-linux-command <  /my/input > output.log # you can combine redirection
# Redirect to file and the standard output? No problem for 'tee'.
command | tee output.log # puts the output of command in output.log
                         # and the terminal too. | is a pipe, see in next slides
```

- ; (semicolon) – execute commands in chain independently to the previous

```
command1 ; command2 ; command3 # first, command1 is executed and then command2, ...
```

- && (double AND) - logical "and" between commands. From now on, we will regard commands as logical expression which are evaluated TRUE, if executed without errors (i.e. is successfull), or FALSE, when executed with error.

```
command1 && command2 && command3 # if command1 is successfull (TRUE), command2 will be executed.
                                 # if command2 is successfull too, command3 is executed
                                 # if e.g. command1 is FALSE, than nor command2 neither command3 is executed
                                 # Why? False && anything && anything will be always false, and the Linux
                                 # command line environment tries to evaluate the whole expression but
    # at  the first FALSE it knows that the whole expression will be FALSE
    # too and stops the evaluation
```

- || (double vertical bar) - logical "or" between commands. Commands in a || sequence will be executed until the last successfull

```
command1 || command2 || command3 # if command1 is successfull (TRUE), command2 will
                                 # NOT be executed.
                                 # command3 will be executed ONLY IF commands 1 and 2 are FALSE
```

- ! command – negating the command

- & (single AND) – execute the command before in the background

```
command1 & command2 # command2 will be started immediately after command1 is started
                    # (because it runs in the background)
```

- Note: ctrl-z stops executing the running command, bg resumes it and sends it to background, i.e. command & == command + ctrl-z + bg
- command == ctrl-z + fg
- | - the pipe, one of the most important way combining commands. It is pipe because it serves as a "medium" for data stream. In particular, it takes the output from the command before the pipe and provides it as input for the command that follows the pipe.

```
command1 | command2 | command3 # the output of command1 goes as input for command2. The output of command2
                              # goes to command3 as input.
                              # if we did not have pipe:
                              # command1 > out1 && command2 < out1 > out2 && command3 < out2
```

- By default, stderr is not "piped" and is written out to the terminal. From BASH v4 use |&. Or use 2 >&1 | for older BASH.

- ( ) (paranthesis) – execute the command(s) in the paranthesis in a separate sub-shell. Way how to combine all the methods for command chaining.

```
( command1 && command2 ) || ( command3 || command4 )
# If command1 is TRUE and command2 is TRUE, the other commands are not executed
# if command1 is TRUE but command2 is FALSE, the command line tries to execute one of commands3/4
# if command1 is FALSE, does not execute command2, and tries to execute one of command3/4
# More complex, but in theory working construction as an example:
(( command1 && command2 ) &> log_c1+2.log  || ( command3 || ! command4 2> error_c4.log )&  )
```

- { }(braces) – Placing a list of commands between curly braces causes the list to be executed in the current shell context. No subshell is created. The semicolon (or newline) following list is required.

```
{ command1 && command2 ; } ||  command3
```

- Excercise 1: How to redirect the stdout to a file, the stderr to a different file and send the stderr to the terminal too (hint: 2>&1 > /dev/null)
- Excercise 2: Write out the second largest file from directory /usr/bin (hint: ls, head or tail. and pipe)
- Excercise 3: Count the number of letters in the name of the first user sorted alphabetically (hint: cut, sort, tail or head, wc)
- Excercise 4: What will be the output of: true || echo aaa && echo bbb
- Excercise 5: what will be the standard output of the following crazy command and the content of the out.log file? Without tying to write to the prompt...
- `(! date > /dev/null && cat /etc/passwd ) || ( ( ! ls -l ~ || head -n 1 /etc/passwd | cut -d: -f1 ) ... ... ||( df -h &> df.log) ) | tail -n 1 > out.log`