Introduction to the Linux OS

Peter Huszár
KFA: Department of Atmospheric Physics

Pavel Řezníček
ÚČJF: Institute of particle and nuclear physics

December 13, 2022

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

KFA MFF UK

ÚČJF MFF UK

sed - stream editor

- SED is the ultimate stream editor. It can substitue strings, add parts of text and delete part of text according to rules given by the users.
- SED can be used in two basic ways:

```
# 1
cat /my/input/file.txt | sed -e  'sed-script' # this generate output on stdout
# 2
sed -i 'sed-script' /my/input/file.txt # this will change the input file inline.
```

- The most used functionality of sed is string substitution

```
# on each line finds the first match for regex and replaces with 'string'
cat /my/file.txt | sed -e 's/regex/string/'
      # "/" here serves as a command delimiter
      # one can use different one too
cat /my/file.txt | sed -e 's:regex:string:'
cat /my/file.txt | sed -e 's/regex/string/2' # replace the 2nd occurence on line
cat /my/file.txt | sed -e 's/regex/string/g' # replace all occurence on the line
```

- In subtitution, one can remember the substituted string

```
echo "123 abc" | sed 's/[0-9][0-9]*/& &/'   # & will stand for the match (123)
                                            # -> 123 123 abc
sed -re 's/([a-z]*) ([a-z]*)/\2 \1/' # \1 and \2 memorize the searched string
                                     # and replace them with \2 \1
                                     # -r extended regex !!!
```

- By default, sed prints all lines in the output, not only those where the replacement occured

```
cat /my/file.txt | sed -n -re 's/regex/string/g p'
# p - print, sed will print the matched lines (not only replacing)
# -n, suppress printing lines, overwritten by "p" (effective for matched lines)
cat /my/file.txt | sed  -re 's/regex/string/g p' # will print matched lines 2x
```

- So far we have learn substitution and printing to all lines
- However, in sed, you can specify, for which line (or line range) to do it. We call it restriction.

- Specifying lines and line range - *restrictions*. The sed command than look like *sed restriction command*

```
cat /my/file.txt | sed -e '5 s/[0-9]/x/' # do the substitution on the 5th
cat /my/file.txt | sed -e '$ s/[0-9]/x/' # do the substitution on the last line ($)
...  | sed -e '10,30 p' # print lines 10-30 twice (with -n only those lines)
...  | sed -e '10,$ any-sed-command' # perform the command from the 10line till end
...  | sed -e '/pattern/ any-sed-command' #  perform the command on lines that match pattern
...  | sed -e '10,/pattern/ any-sed-command' # from line 10 to the 1st matched line (including)
...  | sed -e '/pattern1/,/pattern2/ any-sed-command' #
              # pattern1 switches the 'any-sed-command' on which is switched off by pattern2
```

- i - insert, before the restricted lines (if no restriction is present insert before each line)
- a - append, after the restricted lines (if no restriction is present appned after each line)

```
cat /my/file.txt | sed -e '10,30 i ???' # insert ??? before lines 10-30
cat /my/file.txt | sed -e '/pattern/ a ###' # append ### after lines matching the 'pattern'
```

- A superb SED howto: http://www.grymoire.com/Unix/Sed.html#uh-0

Command Line Editors

- VIM works effectively with keyboard only. Usage of mouse is discouraged. Extensive syntax highlighting support.
- Openning a (new file)

```
vim /my/file # this open the file for writing or even creates it if nonexistent
```

- At this moment, user is in the *command mode*. I.e. any keyboard typed is interpreted as command.
- Let's start with simple editing the file - press `Insert` or `i`. The `-- INSERT --` appears on the bottom. This is the *insert mode*, the basic mode for typing text.
- Pressing `i or Insert` again takes one to the Replace mode. The text is replaced from the cursor.
- To save the changes, press Esc to exit the insert mode and return to the command mode. However, your file is not saved yet!!!
- To save the file, press `:` to enter the *last line mode*. Now, the last line became a command line and waits for a command.

```
:w #  write the changes to the file
:w different_file # 'save as' option to different file
:q # quit (wq - write changes and quit)
:x # write and quit
:q! # ignore the changes and quit
```

- To copy/move parts of text, press `v` or `V` from the command mode to enter the Visual mode.
- Now you can select the text you want to copy/move.
- When the selection is done, press `y` for copy or `d` for cut(move)
- Move the cursor to the place where you want to insert the copied text
- Press `p` for paste
- To copy/move one line press `yy` or `dd` and then `p` for eventual paste
- Simple `d` delete texts. `dd` deletes the line where the cursor is.

- To substitute text, enter the last time command

```
:%s/pattern/replace/g #  global replace of pattern to replace (% = each line)
                      # g = each occurence on line
:1,10 s/pattern/replace/ # replace "pattern" with "replace" on lines 1 to 10
```

- Undo the last change, `:u`
- Search some pattern `/pattern`
- :set nowrap - unwraps long lines
- :syntax on
- :10 - jump to the 10th line
- `Shift + G` jumps to the end of the file
- Very useful tab identation for e.g. Python (tab = 4 spaces):
  `:set tabstop=4 expandtab shiftwidth=4 softtabstop=4`
- `:help shiftwidth`

For almost complete VIM tutorial, see:
https://www.tutorialspoint.com/vim/index.htm

NANO (Nano is ANOther editor) is a small, free and friendly editor

- NANO is part of every Linux distribution's base instalation (which is not true for VIM)
- Easier control over editing. Using `Ctrl+` combination

```
GNU nano 2.5.3              File: myfile                            Modified
Our first text in Nano.
The second line.
^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify   ^C Cur Pos
^X Exit      ^R Read File ^\ Replace   ^U Uncut Text^T To Spell  ^_ Go To Line
```

- To save the just edited "buffer": `Ctrl+O` then enter the filename to save
- Insert content of different file: `Ctrl+R`, to delete the current line: `Ctrl+K`
- To exit: `Ctrl+X`, if one does not want to write the file, press `N`
- NANO supports syntax highlighting

```
ucitel@mypc:~$ ls /usr/share/nano/
# add this line to ~/.nanorc for every language
include /usr/share/nano/python.nanorc # e.g.
```

For almost complete NANO tutorial, see:
https://www.howtogeek.com/howto/42980/
the-beginners-guide-to-nano-the-linux-command-line-text-editor/

EMACS is a GNU project originally created by Richard Stallman. A very powerfull command line editor (with possible graphical interface)
Not covered with this course, for more information:
http://xahlee.info/emacs/emacs/emacs.html

Shell Variables

You can use variables as in any programming languages. There are no data types. A variable in bash can contain a number, a character, a string of characters. You have no need to declare a variable, just assigning a value to its reference will create it.

- Creation and assigning a variable

```
STR="Hello World!"
echo ${STR} # to refer to variable value, use $
MYVAR=1000000
echo ${MYVAR} MYVAR # this prints '1000000 MYVAR'
```

- There are system variables that control the behavior of the system/shell/GUI:
- The command `set` will list all the system/shell variables (and functions - see later)
- E.g. $HOME - the HOME directory, $LANGUAGE - the system language, $PS1 - the look of the prompt
- E.g. $PATH - the list of paths, where BASH looks for binary files
- User can define his own system variables by setting them in .bashrc (in your home directory)
- use `export MYVAR="value"` in order the variable behaves as global
- A variable can be in three states: defined with a value (MYVAR=value), defined with NULL value (MYVAR=) and unset. To unset a variable, use `unset MYVAR`.
- You can define new variables with existing ones: `NEWVAR=${OLDVAR1}${OLDVAR2}` (this example merges two strings)

- Bash enables numerous operations on variable value and gathering information on the variable (besides "asking" for its value)

```
${#MYVAR} # the length of variable value
${!prefix*} # prints all variables with their names starting with "prefix"
${MYVAR#pattern} # removes the match for pattern from the beginnig of MYVAR value
${MYVAR%pattern} # same as above but from the end of value
${MYVAR/pattern/string} # replaces pattern in MYVAR with string
${MYVAR^^} and ${MYVAR,,} # makes variables characters upper/lower case
```

- In the above examples, variables are "expanded" to a new value, which can be written out (with echo) or just saved to different variable(s).
- In the followig example, we rename all jpg files in a directory to JPG

```
for f in *.jpg; do # we will learn later
 echo "Renaming $f ..."
 mv $f ${f/.jpg/.JPG}
done
```

For a full list of variable expansion possibilities, see https://www.gnu.org/software/bash/manual/html_node/Shell-Parameter-Expansion.html

- The output of any command can be assigned to a variable as value in two syntactical way:

```
MYVAR=$( mycommand ) # preferred
MYVAR=‘ mycommand ‘ # not preferred
# the first way enables nesting:
MYVAR=$( mycommand $( anothercommand )   ) # mycommand takes the output of another command as argument
```

- eval - is a built-in Linux command which is used to execute arguments as a shell command. It combines arguments into a single string and uses it as an input to the shell and execute the commands.

```
MYVAR="ls -l /mydir"
eval $MYVAR
MYVAR=’$’
MYVAR2=value
eval echo ${MYVAR}MYVAR2
```

- Use variable as a (part of) name for another variable.

```
MYVAR_A="123"
i=A
echo $((MYVAR_$i))
eval MYVAR_$i="456"
echo $MYVAR_A
```

- Bash supports 1-dimensional arrays with arbitrary integer indexing

```
MYARR= ( 1 2 a b ahoj abc) # definition of an array, in this case indexing is starting from zero
echo ${MYARR[0]} -> 1 etc.
MYARR[100] = value # we can define/add arbitraty index
MYARR=([7]=a [10]=b [100]=c) # possibility of defining arbitrary index
MYARR+=(newelement1 newelement2) # extenstion of array
```

- Different information can retreived of arrays, including its length, list of elemets, list of indexes

```
echo ${MYARR[*]} # prints all the elements
... ${#MYARR[*]} # number of elements
... ${!MYARR[*]} # the list of indexes
```