Faculty of Mathematics and Physics
CHARLES UNIVERSITY

NOFY077

Introduction to the Linux OS

Peter Huszár

KFA: Department of Atmospheric Physics

Pavel Řezníček

ÚČJF: Institute of particle and nuclear physics

January 3, 2023

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

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

Client-Server Services

Services for networking

- `ssh` for remote login to a command line
- `kerberos` for more secure and complex remote logins
- `telnet` old, now insecure, remote login to a command line
- `scp` to copy files via ssh
- `ftp` to copy files between PCs
  - `sftp` as it secure replacement (uses `ssh`, but emulates `ftp` commands)
- Web-server
- Command line tools to access web-pages
  - `wget`, `curl`: downloading www-page source
  - `lynx`, `elinks`: text-based www browsers, i.e. translating www-page source to the www-page
  - Limited usage on web-pages with e.g. java scripts etc.

SSH

Remote login to a command line, remote file transfer via `scp`

- SSH allows for password-less login using public/private keys
  - Create key pair (of type rsa) via `ssh-keygen -t rsa`
  - Key will be by default saved to `/home/$USER/.ssh/id_rsa`
  - Will ask for password (not the login one, but an optional new one guarding the key)
  - Copy public key `/home/$USER/.ssh/id_rsa.pub` to `/.ssh/authorized_keys` file on the remote server(s)
  - If password for the key was set, one will be asked for it on login on the local PC
- Comments on SCP
  - Works like `cp` command
  - To interpret wildcards on remote machine, use `""` or backslash \
  - Syntax or remote path: `username@machine:path`

FTP

Transfer files from/to FTP server

- Login to an FTP server: `ftp machine`
- Most common commands
  - `?` to request help or information about the FTP commands
  - `ascii` to set the mode of file transfer to ASCII (this is the default and transmits seven bits per character)
  - `binary` to set the mode of file transfer to binary (the binary mode transmits all eight bits per byte and thus provides less chance of a transmission error and must be used to transmit files other than ASCII files)
  - `bye`, `quite` to exit the FTP environment
  - `cd` to change directory on the remote machine
  - `close` to terminate a connection with another computer
  - `delete` to delete (remove) a file in the current remote directory (same as rm in UNIX)
  - `get` to copy one file from the remote machine to the local machine
    - `get ABC DEF` copies file ABC in the current remote directory to (or on top of) a file named DEF in your current local directory.
    - `get ABC` copies file ABC in the current remote directory to (or on top of) a file with the same name, ABC, in your current local directory.
  - `help` to request a list of all available FTP commands
  - `lcd` to change directory on your local machine (same as UNIX cd)
  - `ls` to list the names of the files in the current remote directory
  - `mkdir` to make a new directory within the current remote directory
  - `mget` to copy multiple files from the remote machine to the local machine; you are prompted for a y/n answer before transferring each file. `mget *` copies all the files in the current remote directory to your current local directory, using the same filenames. Notice the use of the wild card character
  - `mput` to copy multiple files from the local machine to the remote machine; you are prompted for a y/n answer before transferring each file
  - `open` to open a connection with another computer
  - `put` to copy one file from the local machine to the remote machine
  - `pwd` to find out the pathname of the current directory on the remote machine
  - `rmdir` to to remove (delete) a directory in the current remote directory

Servers

SSH Server

- Install `openssh-server` package
- Configuration in `/etc/sshd` directory, controlling who and how can access the server
- Serves for `scp`

FTP Server

- Several packages: `vsftpd`, `proftpd`

WWW Server

- Based on `apache` package
- Complex configuration, can use of simplier packages as `lighttpd`

Programming on Linux

Programming on Linux

- Number of GUI tools for program development
  - *Visual Studio Code*, *Eclipse*, *KDevelop*, ...
  - Debugging: *kdbg*, *ddd*, *gdb*
- But in the background, the GUI tools use command-line programs

Compilation

- Processing of source code files (e.g. `*.cpp`) and creation of an object file (`*.o`)
- Does not create executable
- Does not care about whether libraries, on which the source depends, are present

Linking

- Create executable or library from multiple object files
- Check that all the dependencies are satisified

- For simple programs compilation and linking can be done in a single step
- Tools:
  - `g++`: compilator/linker for C++ and C programs
  - `gcc`: compilator/linker for pure C programs
  - `gfortran`: compilator/linker for FORTRAN programs
  - ... number of tools for other programming languages

Example of a C++ program and library

- Simple C code showing main program myprog and usage of a library mylib
- mylib.h

```
#ifndef mylib_h__
#define mylib_h__
extern void mylib(void);
#endif  // mylib_h__
```

- mylib.c

```
#include <stdio.h>
void mylib(void)
{
  puts("Hello, I am a shared library");
}
```

- myprog.c

```
#include <stdio.h>
#include "mylib.h"

int main(void)
{
    puts("This is a shared library test...");
    mylib();
    return 0;
}
```

Compile and link the program

- Compile

```
gcc -c -Wall -Werror -fpic mylib.c
```

- Create shared library

```
gcc -shared -o libmylib.so mylib.o
```

- Compile and link the program with the shared library

```
gcc -Wall -o test main.c -lmylib
gcc -L/home/reznicek/tmp/test -Wall -o test main.c -lmylib
```

- Run the program

```
export LD_LIBRARY_PATH=/home/reznicek/tmp/test:$LD_LIBRARY_PATH
./test
```

- One can use *rpath* to advise the program, where to look for the library

```
gcc -L/home/reznicek/tmp/test -Wl,-rpath=/home/reznicek/tmp/test -Wall -o test main.c -lmylib
./test
```

- Check dependence of executable/library on other libraries

```
ldd ./test
```

- Libraries are by default searched in system paths defined (and updated by) `ldconfig` program with its configuration in `/etc/ld.so.conf*` files

Compile and link the program - static library

- Compile

```
gcc -c -Wall -Werror -fpic mylib.c
```

- Create static library

```
ar crv libmylib.a mylib.o
```

- Compile and link the program with the static library. The library is included directly in the executable (see `test_static` size vs `test` size) instead of being shared (e.g. with other programs).

```
gcc -Wall -o test_static ./libmylib.a main.c
```

Example of a FORTRAN program and library

- Simple FORTRAN code showing main program myprog and usage of a library mylib
- mylib.f

```
subroutine mylib()
  print*, ’Hello, I am a FORTRAN library’
end subroutine func
```

- myprog.f

```
program myprog
  print*, ’This is a FORTRAN library test...’
  call mylib()
end program myprog
```

- Compile and link using gfortran

```
gfortran -shared -fPIC -o libmylib.so mylib.f
gfortran -L. myprog.f -lmylib -o test_fortran
```

Notes on Python language

- Interpreted language, i.e. no real compilation to machine code
- tabs and spaces to define blocks
- Python libraries through `import` command
- Example:

```
import os
os.listdir()
```

- Still, there is a possibility to do compilation into a byte code
  - Every Python program is translated to byte code, before being executed by the Python's virtual machine
  - Compiling the program into the byte-code thus speeds up execution

```
python -m compileall .
```

```
import py_compile
py_compile.compile('abc.py')
```

Makefile and other build systems

- Writting all the compiler commands for complicated projects is annoying
- `Makefile` and `make` commands are there to make this easier
- Example of simple makefile for C++:

```
TARGET_EXEC ?= a.out

BUILD_DIR ?= ./build
SRC_DIRS ?= ./src

SRCS := $(shell find $(SRC_DIRS) -name *.cpp -or -name *.c -or -name *.s)
OBJS := $(SRCS:%=$(BUILD_DIR)/%.o)
DEPS := $(OBJS:.o=.d)

INC_DIRS := $(shell find $(SRC_DIRS) -type d)
INC_FLAGS := $(addprefix -I,$(INC_DIRS))

CPPFLAGS ?= $(INC_FLAGS) -MMD -MP

$(BUILD_DIR)/$(TARGET_EXEC): $(OBJS)
        $(CC) $(OBJS) -o $@ $(LDFLAGS)

# assembly
$(BUILD_DIR)/%.s.o: %.s
        $(MKDIR_P) $(dir $@)
        $(AS) $(ASFLAGS) -c $< -o $@

# c source
$(BUILD_DIR)/%.c.o: %.c
        $(MKDIR_P) $(dir $@)
        $(CC) $(CPPFLAGS) $(CFLAGS) -c $< -o $@

# c++ source
$(BUILD_DIR)/%.cpp.o: %.cpp
        $(MKDIR_P) $(dir $@)
        $(CXX) $(CPPFLAGS) $(CXXFLAGS) -c $< -o $@


.PHONY: clean

clean:
```

Makefile and other build systems

- Makefiles are universal, not just for programming
- Example of simple makefile for Latex:

```
# LaTeX Makefile
FILE = Template_slides_biber
TEX  = pdflatex
BIB  = biber

all: $(FILE).pdf

.PHONY: clean $(FILE).pdf

cleanmin:
        #rm -f ${FILE}.pdf
        rm -f *.aux
        #rm -f *.bbl
        rm -f *.bcf
        rm -f *.blg
        rm -f *.blx.bib
        rm -f *.idx
        rm -f *.ilg
        rm -f *.ind
        rm -f *.lof
        rm -f *.log
        rm -f *.lot
        rm -f *.nav
        rm -f *.out
        #rm -f *.run.xml
        rm -f *.snm
        rm -f *.toc
        rm -f *.vrb

clean: cleanmin
        rm -f *.bbl
        rm -f *.run.xml
        rm -f ${FILE}.pdf

$(FILE).pdf: *.tex
        $(TEX) $(FILE).tex
        $(TEX) $(FILE).tex
        $(BIB) $(FILE)
        $(TEX) $(FILE).tex
        $(TEX) $(FILE).tex
```

Makefile superstructures

- Makefiles are powerful, but still not flexible enough when one wants to include various build configurations (debug, plugins, search for needed libraries)

Automake tools

- Configuration via `configure` script with `--help` to find options
- Creates `Makefile` from simplier `Makefile.am`

```
autoconf
./configure
make
make install
```

CMake

- Even more complex system creating Makefiles
- Uses out-of-source build directory (not mixing build and source files)
- List of options via `cmake -LAH` command

```
cmake source_dir
make
make install
```