Introduction to the Linux Operating System
Lecture notes on a course held in the winter 2019/2020

Stefan Kurtz

Research Group for Genome Informatics
Center for Bioinformatics Hamburg
University of Hamburg[1]

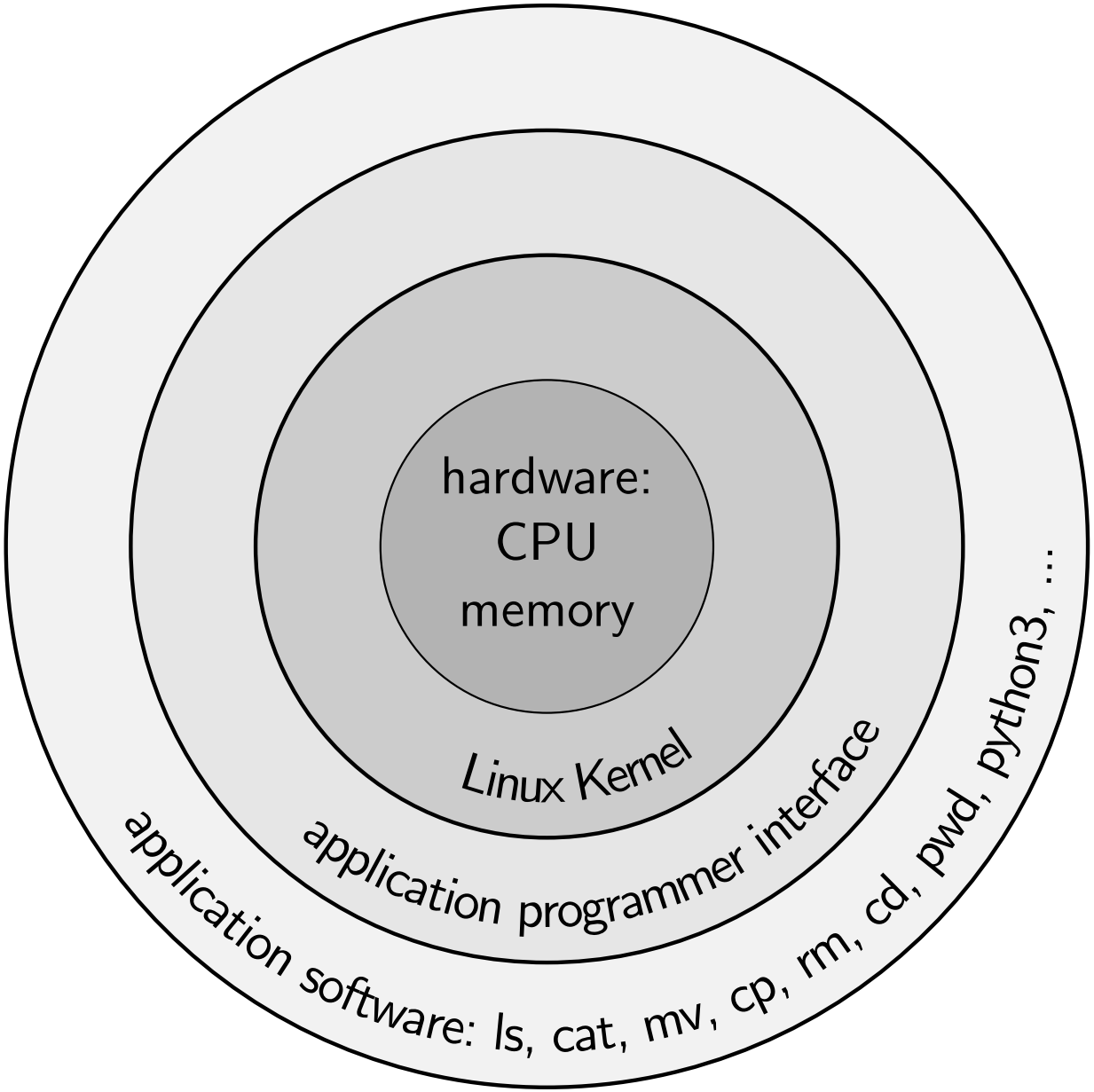
October 31, 2019

[1] parts of the slides are adapted from
`http://cbsu.tc.cornell.edu/lab/doc/linux_workshop_part1.pdf`

Why Unix/Linux and not MS Windows

- Linux is widely used in the sciences
- many software tools for scientific applications are developed for Linux and run as command line tools
- Linux contains a huge collection of useful programs with clearly defined interface
- on the command line these software and tools can easily be combined to provide flexibility
- Knowing linux and the command line is a plus in your career

Layered Structure of a Linux System



CPU: Central Processing Unit

Important Notions (1/3)

Linux kernel

- first program to run when machine boots
- runs all other programs
- part of an operating system
- interface with the CPU and memory

Use of Linux kernels in

- desktop OS (e.g. Ubuntu)
- smart phones and tablets (Android)
- embedded devices (e.g. router, WLAN hub)
- file and compute servers
- supercomputers (in 498 of 500 most powerful, as of June 2017)

Important Notions (2/3)

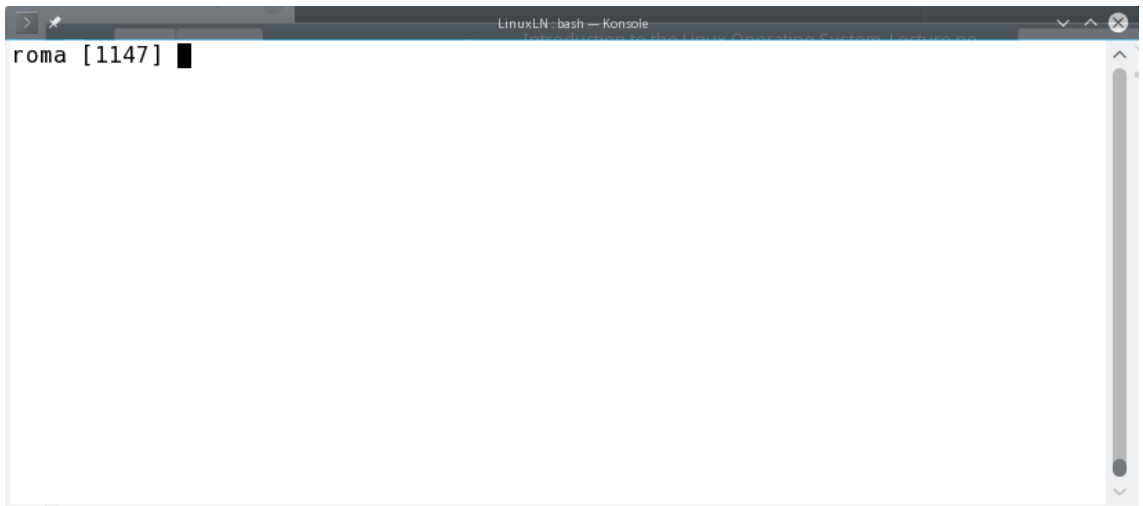
shell (usually the bash: bourne again shell)

- first program to run when user logs in
- reads and executes user commands: command line interpreter for application software
- this is the program you communicate with, when you ...
  - start the editor
  - run the python interpreter to execute your programs
  - compare your results with those expected

Important Notions (3/3)

Terminal

- can receive input and output in standard encodings (i.e. ASCII/UTF)
- waits for input typed by user following the prompt (roma [1147] below, in examples we use $ as a prompt)
- input is interpreted as command and executed by bash



commands explained below are executed by typing them in the terminal after the prompt, followed by return

- in exercises communication with computer will be via terminal window
- no special graphical user interfaces (like file browser) required

Basic Syntax of Linux Commands

`command` [options] [arguments]

commands

- built-in or programs
- will usually have default options

arguments

- usually one or more files

options

- depend on command
- usually given with single (old style, -) or double hyphens (new style, --).
- can usually be combined

special characters in commands

/	separate directory in file paths
>	redirect output to file
<	read input from file
${VAR}	evaluation of environment variable VAR; for example: home directory ${HOME}

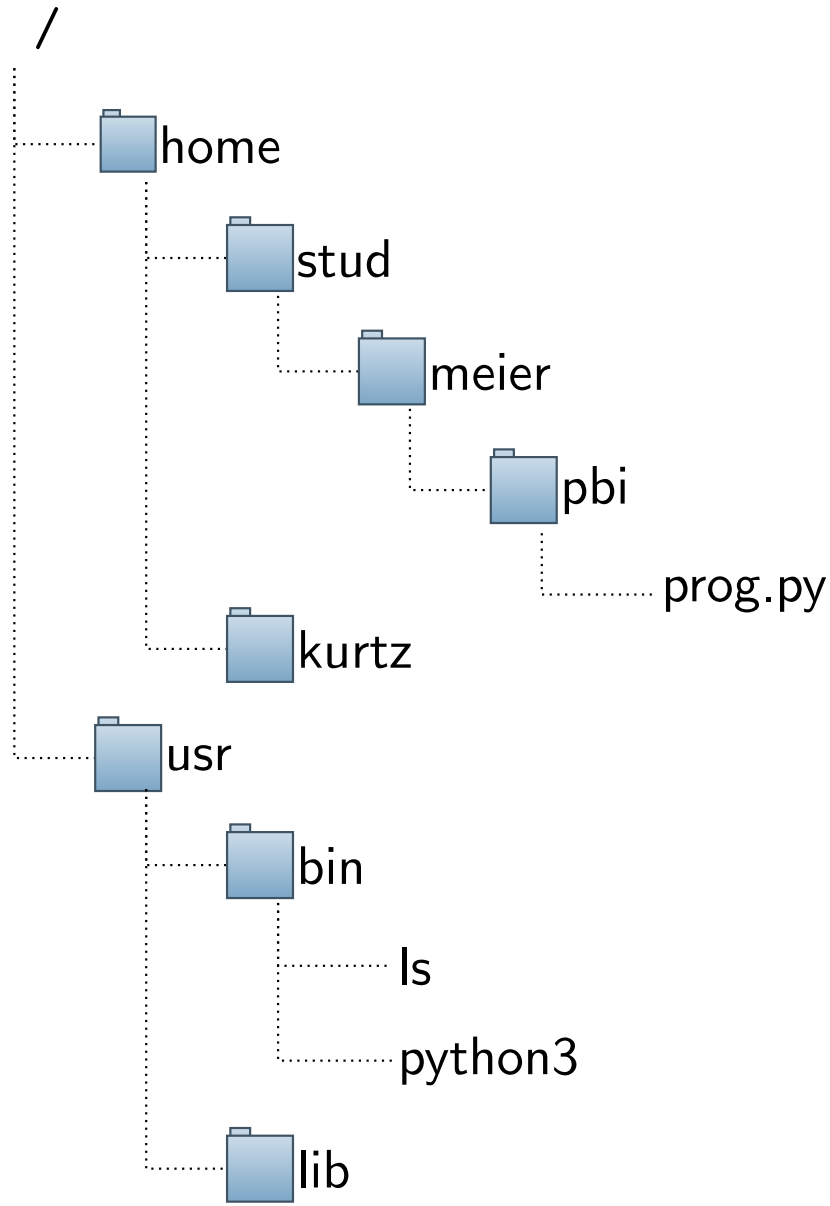
Some important commands related to files

name	meaning	example
`cp`	copy a file to another file	`cp original copy_of_original`
`mv`	rename and/or move a file	`mv oldname newname`
`rm`	remove a file	`rm file1`
`cat`	concatenate files and send output to screen	`cat file1 file2`
`vim`	start editor vim	`vim file.py`
`less`	display file contents	`less blatt1.tex`
`head`	display first lines of a file	`head -n 15 file1`
`tail`	display last lines of a file	`tail -n 10 file1`
`wc`	count all characters, words, and lines	`wc file1 file2 file3`
`sort`	sort lines of file	`sort -n data.txt`
`tr`	translate/delete single characters in file	`cat file | tr -d '\n'`
`grep`	search for a pattern in a file	`grep measles paper.txt`

Hierarchical Linux File System

notions:

- current working directory is a location in directory tree
- description by a path
- path can be absolute or relative
- / is root directory
- in path, / separates directories
- . is current directory
- .. is parent directory
- directories shown as boxes



Example (Copying files)

1. make a copy of file `sample.txt` from current directory in `/home/stud/meier`; use same name; target directory must exist

   ```
   $ cp sample.txt /home/stud/meier/.
   ```

2. same as before, but name of target file is sample1.txt

   ```
   $ cp sample.txt /home/stud/meier/sample1.txt
   ```

3. make copy of `p.py` from `/home/kurtz/pbi` in current directory using same file name

   ```
   $ cp /home/kurtz/pbi/p.py .
   ```

4. copy all files ending with `.py` from `/home/kurtz/pbi` to `/home/stud/meier/pbi`:

   ```
   $ cp /home/kurtz/pbi/*.py /home/stud/meier/pbi
   ```

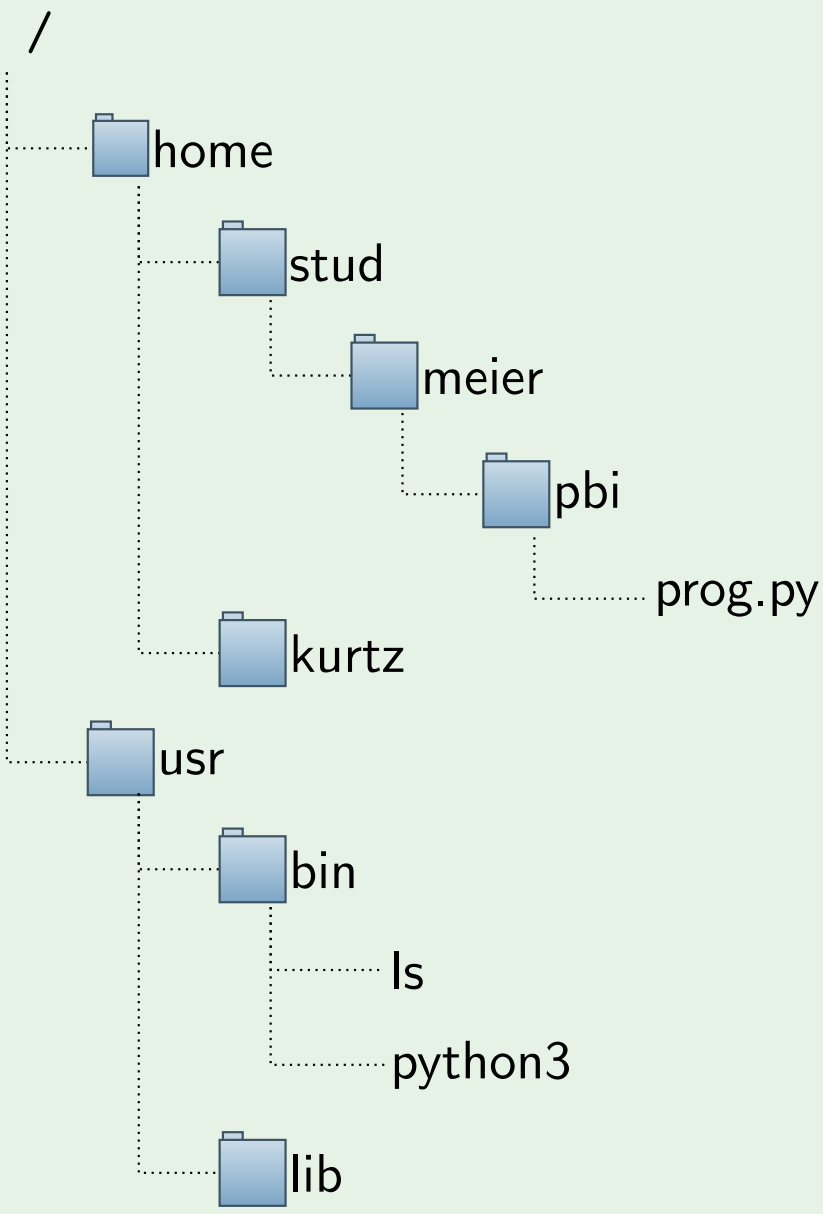
5. copy an existing directory pbi from a co-student mueller with all files and subdirectories to own home directory

   ```
   $ cp -R ../mueller/pbi ~
   ```

Directory Commands

name	meaning	example
`cd`	change directory	`$ cd ../..`
`pwd`	print current/working directory	`$ pwd`
`mkdir`	make a directory	`$ mkdir pbi`
`rmdir`	remove a directory	`$ rmdir -f pbi`
`ls`	list files in current directory	`$ ls -l`

Example (Navigating in the directory tree)



1. use absolute path to change to `pbi`-dir of `meier`:

   ```
   $ cd /home/stud/meier/pbi
   ```

2. use relative path to change from `/usr/bin` to `kurtz`:

   ```
   $ cd ../../home/kurtz
   ```

3. use relative path to change from `meier/pbi` to `kurtz`:

   ```
   $ cd ../../../kurtz
   ```

4. as user `meier` change from anywhere to subdirectory `pbi`

   ```
   $ cd ~/pbi
   ```

   or

   ```
   $ cd ${HOME}/pbi
   ```

Types of files

- directories are structured collections of files of different types

text files

as ASCII- or UTF-8 files:

- plain text
- LaTeX sources
- Python scripts
- web-pages in HTML
- C-source code

binary files

in formats comprehensible only to the machine or specific programs; not human-readable

- executable files
- PDF files (.pdf)
- compressed files (.gz, .zip, .bz2)

- specific file suffix for certain file types is not mandatory
- the program `file` classifies files

```
$ file /usr/bin/ls
/usr/bin/ls: ELF 64-bit LSB executable, ...
$ file countwords.py
countwords.py: Python script, ASCII text executable
```

Where do files come from?

- they are created by various programs, e.g.,
  - text editors
  - file compression tools
  - system commands (`cp`, `mv`, ...)
  - screen output redirection
  - remote copy tools (`scp`, `sftp`, `wget`, `firefox`)
  - application software

Rules for naming files

- names are case-sensitive
  - `MyFile`, `myfile`, `myFile` are all different
- may not be true on Mac OS X
- use only letters (upper-and lower-case), digits, a dot (.), and an underscore (_):

  e.g. `first_python_script.py`
- avoid other characters, like $, &, #, or white spaces in file names

  e.g. ~~`python script #42.py`~~ ~~`my-module.py`~~
- these characters have special meaning to either Linux, or to the application you are trying to run
- file with names including these character will cause problems when you or others access your files or directories

Input/Output Redirection & Pipes

concept of Linux programs

- programs do one thing and they do it well
- programs handle text streams, i.e. they read from/write to the standard input/output (usually, the terminal window)

input/output *streams*:

- `STDIN`: standard input
- `STDOUT`: standard output
- `STDERR`: standard error output

redirection operators for streams

`>`	send output to file	`$ cat file1 file2 > file3`
`<`	use file as input	`$ wc < file1`
`>>`	append output to file	`$ cat file1 >> file3`
`>!`	force creation of existing file	`$ cat *.tex >! file3`
`>&`	send `STDOUT` and `STDERR` to file	`$ python3 p.py >& err.txt`

Pipes (1/2)

- for more complex task one needs to combine several programs

Example (Solving a task by sequence of commands)

Task: display the 1 000 largest numbers (without duplicates) in the first 100 000 lines of the unordered file `data.txt` (with one number per line):
Solution:

```
$ head -n 100000 data.txt > tmp1.txt
$ sort -u -n tmp1.txt > tmp2.txt
$ tail -n 1000 tmp2.txt
```

- handling of temporary files is slow and inconvenient
- the pipe operator | allows to specify that output of step i becomes input of step $i + 1 \Rightarrow$ no temporary files necessary

```
$ head -n 100000 data.txt | sort -u -n | tail -n 1000
```

Pipes (2/2)

more examples using pipes:

- count the number of files and sub-directories in current directory:
  `$ ls | wc -l`
- list all names of files and directories in current dir containing the string `tex` in upper or lower case
  `$ ls | grep -i tex`
- count number of lines in `dnafile.fna` beginning with the symbol >
  `$ grep '^>' dnafile.fna | wc -l`
- make a copy of a file with DNA sequences in upper case notation
  `$ cat dnafile-lower | tr acgt ACGT > dnafile-upper`
- display list of files with suffix `.pdb` in current dir sorted in ascending order of their number of lines
  `$ wc -l *.pdb | sort -n`
- Count number of characters excluding newline in file1
  `$ cat file1 | tr -d '\n' | wc -c`

Time Sharing

- Linux is a multi-user access, multi-tasking system:
  - multiple users may be logged in and
  - run multiple tasks on one machine at the same time

  $\Rightarrow$ need to share resources (CPUs, memory, disk space)
- Linux (as most modern operating systems) usually runs more than one process possibly for different users at a time

Multi-User Environment

- multiple users on Linux system share the computing resources
- authentication of users is required
  - each user has a unique account
  - users can be parts of groups to share files not accessible by others
  - all files have individual permissions to read/write/execute

Example (Long listing)

```
$ ls -l
total 596
-rw-r--r--  1 kurtz gi  1709 2017-05-24 18:12 ascii.c
drwxr-x---  2 kurtz gi   680 2017-10-10 22:30 LinuxLN/
-rwxr-xr-x  1 kurtz gi  1575 2017-05-24 18:12 aufgaben.rb*
-rw-r-----  1 kurtz gi 11540 2017-10-18 23:09 unixprog.tex
-rwxrw-r--  1 kurtz gi  1269 2017-05-24 18:12 p.py
```

File permissions (first column of listing)

1. are for user, group, others (`u`, `g`, `o` in this order)
2. specify permissions to read, write, execute (`r`, `w`, `x` in this order)

-rwxrw-r–	1	kurtz	gi	1269	2017-05-24	18:12	p.py
permission user group others	#links	owner of file (user)	group	size in bytes	last changed		name

r: readable
w: writable
x: executable
-: no permission.

user has all permissions
group has read & write permissions
others have only read permissions
initial character: d for directories or else -

chmod allows user to change permissions

- remove write permissions for group

  ```
  $ chmod g-w p.py
  ```

- give user permission to execute

  ```
  $ chmod u+x p.py
  ```

- give others permission to read

  ```
  $ chmod o+r p.py
  ```

- remove executable permissions for all users

  ```
  $ chmod -x p.py
  ```

Processes in Linux (1/2)

A process ...

- carries out a computing task in the operating system
- comprises and manages all relevant information for a running program
- processes can run in foreground or background
- foreground process (the usual case): you type a command and view the result in the terminal
- background process: commands that require a longer time or do not display there results on the terminal (e.g. a PDF-viewer) should be run as a background process: add suffix & after command
- shell does not have to wait for background process

Processes in Linux (2/2)

Managing a process

Ctrl-c	interrupt foreground process
Ctrl-d	declare end of file/input
`$ ps`	display list of processes with their status
`$` *command* `&`	start process in background, return to shell immediately
`$ fg`	bring background process to foreground
`$ jobs`	display processes in background
`$ kill`	kill a process

Example (display processes matching a regular expression)

```
ps -ef | egrep 'kurtz.*bash'
kurtz    25309 25243  0 Oct01 pts/0    00:00:00 /bin/bash
kurtz    25354 25243  0 Oct01 pts/2    00:00:00 /bin/bash
```

More useful commands

name	meaning	example
`find`	find and manipulate files recursively	`find ~ -name 'blatt1.*'`
`uname`	display specification of machine (name, OS, etc)	`uname -a`
`man`	display manual entry	`man find`
`du`	display disk space	`du -h .`
`gzip`	compress a file	`gzip -9 bigfile`
`gunzip`	uncompress a file	`gunzip bigfile.gz`
`tar`	manipulate tape-archives	`tar xvf archive.tar`
`cut`	extract columns from a file	`cut -d , -f 2 data.txt`
`paste`	merge lines of files	`paste file1 file2`
`diff`	show differences in files	`diff file1 file2`
`wget`	copy file specified by URL	`wget http://web.org/f1.pdf`
`touch`	create new file or change date stamp of existing file	`touch pbi`

Environment variables (1/2)

- to set values available to all commands, use environment variables
- e.g. many programs need to know the home-directory of current user
- this is stored in the environment variable `HOME` (set at login)
- to display the value of this variable, type

  ```
  $ echo ${HOME}
  /home/stud/meier
  ```

- another important environment variable is `PATH`
- stores paths (separated by `:`) in which the shell looks for executable programs you type (usually without a path) on the command line

  ```
  $ echo ${PATH}
  /bin:/usr/bin:/usr/local/bin
  ```

Environment variables (2/2)

Example

When you execute

```
$ ls
```

- shell checks the 3 directories, `/bin`, `/usr/bin`, `/usr/local/bin` (in this order, as stored in `PATH`) whether any stores the program `ls`
- it finds `/bin/ls` and executes this file

- often you want own executable files to be run from anywhere in the directory tree
- store these in a directory `bin` below your home directory
- then add this line to the file `~/.bashrc`:

```
export PATH=${PATH}:${HOME}/bin
```

- this line appends `${HOME}/bin` to your path list
- becomes effective after a new login or after executing

```
$ .  ~/.bashrc
```

Text Editors

- there are many possible text editors available
- initially you probably want to start with a simple one, like e.g.
  - nano
  - joe
  - gedit
  - kate
- once you have more experience and write more code you may at some point switch to a more powerful editor, like
  - vim
  - emacs
- if you want to edit you files over remote access (see below), then you should use vim

Logging in to a computer in ZBH-Pool (1/3)

Local access in the pool

- you need an account name with a password
- type these into the login window of one of the computers

Remote access from a computer running Linux/Mac: step 1

- open a terminal and execute

```
$ ssh -p 7373 -X <account>@bari.zbh.uni-hamburg.de
```

Remote access from a computer running MS Windows: step 1

- use PuTTY, see `https://www.chiark.greenend.org.uk/~sgtatham/putty/`
- use port 7373, hostname `bari.zbh.uni-hamburg.de` and your account

Logging in to a computer in ZBH-Pool (2/3)

Remote access: step2

- to login to bari, type in your password

```
Last login: Mon Oct  9 20:03:16 2017 from xyz.dyn.telefonica.de
Welcome to gateway.zbh.uni-hamburg.de
Please note that you share this gateway with other interactive
users, so avoid large jobs. X11 will be very slow.
```

- in the terminal execute

```
$ ssh -X mondsee
```

- instead of `mondsee` use any other of the computers in the ZBH pool, see next slide

Logging in to a computer in ZBH-Pool (3/3)

Name of computers in the ZBH student pool

- barmsee
- bannwaldsee
- brombachsee
- eibsee
- freibergsee
- hartsee
- hopfensee
- kochelsee
- mondsee
- obersee
- ostersee
- pilsensee
- plansee
- riegsee
- simssee
- weissensee
- weitsee
- rottachsee

Logging out of a computer in ZBH-Pool

- while in terminal window, type `exit` or Ctrl-d: this will close the current terminal window
- before leaving, always logout, using the corresponding menu of your desktop (can be very different depending on the Linux-distribution)

Copying files from and to the ZBH-computers

- your home directory on the computers in the ZBH is directly accessible for reading an writing (with your password, of course)
- assume you have
  account name: `jdoe`
  home directory: `/home/stud/jdoe`

  and have stored your solution for exercise sheet 5 in file `Blatt05.Nm1.Nm2.tar.gz` in your home directory.

  ```
  scp -P 7373 jdoe@bari.zbh.uni-hamburg.de:/home/stud/jdoe/Blatt05.Nm1.Nm2.tar.gz .
  ```

  will copy the file to your remote computer.
- do not forget dot at end (denotes target file, same a source file)
- the next command will copy the file from your remote computer to your home directory on the ZBH-computers:

  ```
  scp -P 7373 Blatt05.Nm1.Nm2.tar.gz jdoe@bari.zbh.uni-hamburg.de:/home/stud/jdoe/.
  ```

- both commands can be abbreviated by the shell script `myscp.sh`:
  `myscp.sh jdoe get Blatt05.Nm1.Nm2.tar.gz`
  `myscp.sh jdoe put Blatt05.Nm1.Nm2.tar.gz`

Useful tricks (1/2)

Avoid excessive command typing in shell by

- using copy/paste: any text in the terminal or elsewhere can be selected with the mouse

  while holding the left mouse button it is copied to clipboard.

  it may then be pasted, e.g., into a command, by clicking the right mouse button
- using up/down arrow keys: this will cycle through recently executed commands.
- using the TAB key: this will often present you with a list of choices after typing a part of a command

Useful tricks (2/2)

Avoid excessive command typing by

- using the history of commands stored by the terminal:
- e.g. to display all previous commands with their number on screen

```
$ history
..
41  20:57  okular intro-linux.pdf &
42  21:01  pdflatex intro-linux
..
```

- to display all previous commands containing the string `pdflatex`

```
$ history | grep pdflatex
```

- to execute command number 42

```
$ !42
```