Introduction to the command line

April 13, 2017

Navigating the shell

Why use the command line?

- powerful and fast way of interacting with a computer
- graphical user interfaces tend to slow you down
- many powerful tools available

What is the shell

- command interpreter Read-Eval-Print-Loop (REPL)
- interface between user and computer/operating system
- is itself a programming language (shell scripts)
  - requires input
  - produces output
  - has variables and a state

Popular shells

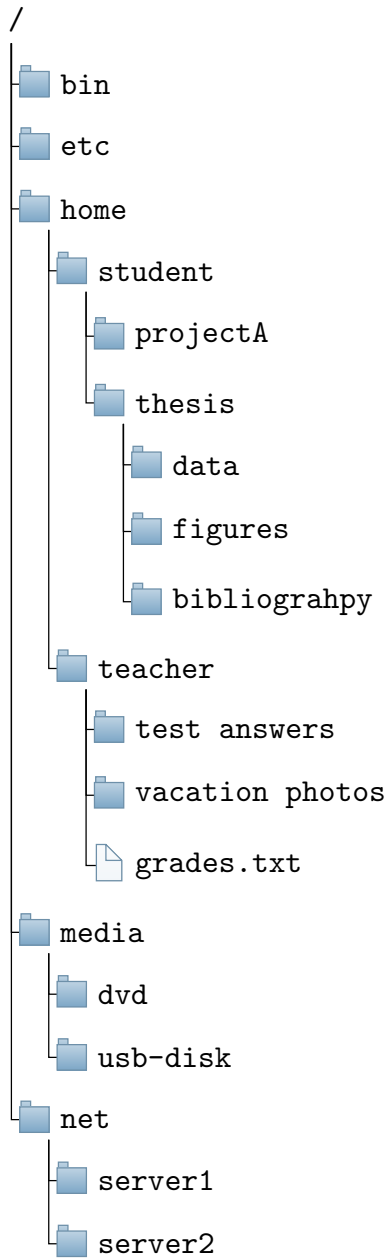
Table: Overview of most used shells

exe name	name	description
`sh`	Bourne shell	the original from 1977
`csh`	C shell	inspired by C programming language, improvement to `sh`
`ksh`	Korn shell	further extensions to `sh`, compatible with `sh`
`bash`	Bourne again shell	free replacement for `sh`, modern version, many improvements, default on most systems
`tcsh`	Tenex C shell	updated and extended C shell
`zsh`	Z shell	very powerful modern `bash` alternative (somewhat compatible)

Path and directories I

- folders are called directories
- directory can hold files and other directories
- files and directories can be addressed by a path
- there are absolute paths and relative paths
- absolute paths begin at the top of the tree
  - / is the root directory
  - reachable from everywhere
  - example: `/home/student/thesis/data`
- relative paths
  - realtive to the current working directory
  - current directory denoted by .
  - parent directory denoted by ..

Path and directories II



Home directory (~)

- new terminal/shell usually starts in your home directory
- abbreviated with the tilde character ~
  - ~ → `/home/student` (for user `student`)
  - ~ → `/home/teacher` (for user `teacher`)
- because of this extension `~/thesis/figures` and `/home/student/thesis/figures` are both absolute paths

Listing directory content (`ls`)

- the command `ls` lists all files and directories in the current directory
- `ls [path]` list the content of the given path, without leaving the current directory

e.g. `ls /bin`, `ls thesis/figures`, `ls ~/vacation_photos`

Changing directories (cd)

- no matter where you are, you can always just type `cd` and get back to your home directory
- `cd` is equivalent to `cd ~`
- `pwd` (print working directory) shows your current "location"
- `cd [path]` changes to the given directory
  - has to be a directory, cannot be a file!
  - works with relative and absolute paths

Table: Path shortcuts

~	home directory
.	current directory
..	parent directory
../..	parent of parent

- e.g. `cd ./figures` is equivalent to `cd figures`

Working with files and directories I

copying files (`cp`)

- `cp <source> <destination>`
  - `source` and `destination` have to be relative or absolute paths

moving or renaming files (`mv`)

- `mv <source> <destination>`
  - can be used to move `source` file to new `destination` or to rename a file

Working with files and directories II

creating directories (`mkdir`)

- `mkdir <name>`
  - creates a directory in the current one (relative path)
- `mkdir -p <name>`
  - can create a tree of paths, takes also absolute paths as argument
  - e.g. `mkdir -p ./homework/series1/exercise2`

deleting files and directories (`rm` / `rmdir`)

- `rm <file>`
  - deletes file irrecoverably
- `rmdir <dir>`
  - deletes directory
- `rm -r <path>`
  - deletes path recursively (files and directories)

Remark: The `-r` flag can also be used with the `cp` and `mv`

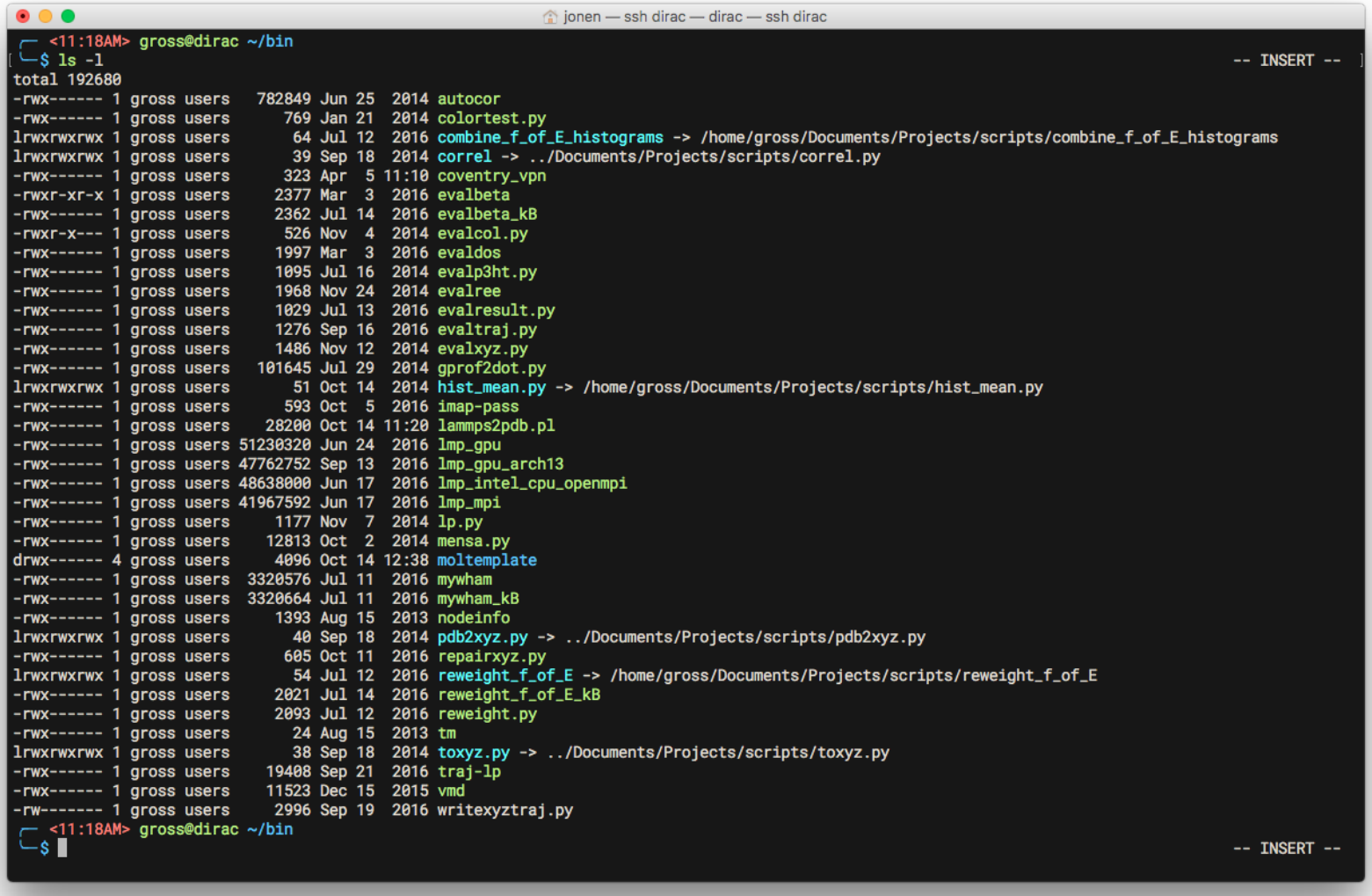
File permissions I

In UNIX files and directories have flags, that determine who was permission the access them.

- three types of people are distinguished:
  1. the User
  2. the Group a user is member of
  3. all Others
- there are 3 different types of access
  1. permission to Read
  2. permission to Write
  3. permission to eXecute
- change file permission with `chmod`, `chown`, and `chgrp`
- `chmod o+rx myProgram.py`
- `chown -R user:group [dir]`
- `chgrp newgroup [file]`

eXecutable flag needed for programs and scripts to be runable.

File permissions II



Displaying file content I

beginning of a file (`head`)

- usage: `head [options] [file]`
- standard output first 10 lines of the file
- usually enough to get a good idea about the contents of the file
- option `-n x` or `--lines=x` would display `x` lines

end of a file (`tail`)

- usage: `tail [options] [file]`
- standard output last 10 lines of the file
- interesting feature: follow mode
  - monitor a file that is still written too
  - `tail -f [filename]`
    - will update as soon an new data is written to the file and display the data on the screen

Displaying file content II

print whole file (`cat`)

- usage: `cat [filename]`
- will print the whole content of the file to the screen

print whole file (`less`)

- usage: `less [filename]`
- displays first "page" of the file
- scroll through file using up and down arrows
- `PageUp` and `PageDown` move by "page"
- search inside file with `/`
- `n` next occurrence of search term
- `q` quits back to the shell

Redirecting output (>, », and |) I

- in UNIX everything is a file
- the contents of your display is a special file `/dev/stdout` (the standard output)
- your keyboard is your standard input `/dev/stdin`
- you can redirect input and output from one file to another
- chain commands together to pass input and output between them
- `run_simulation.sh > measurement.dat`
  - will redirect output of simulation from `/dev/stdout` to data file
  - creates file if it does not exist
  - CAUTION: will overwrite file if it exists

Redirecting output (>, », and |) II

- `evaluate_data.py » results.dat`
  - redirects output from `/dev/stdout` to file
  - creates if non-existent
  - append new data to end of file if it already exists
- `run_simulation.sh | evaluate_data.py » result.dat`
  - chain commands together
  - take output from `run_simulation.sh` as input for `evaluate_data.py`
- redirect `/dev/stdout` and `/dev/stderr`
  - `debug_run.py 1> output.dat 2> debug_info.dat`
  - separate data from error or debug messages

Manipulating columns of data (`awk`) I

- powerful scripting language to work on lines and columns of a data file
- e.g. print only 2nd and 4th column of a data file
  - `awk '{print $2, $4}' data.txt`
  - can be used to reorder columns
  - extract certain data form a bigger file (redirect to different file)
- built-in variables like line number `NR`
  - `awk '{print NR}' file` (will print line number starting at 1 till end of file)
  - `awk 'END {print NR}' file` (will print last line number only)

Manipulating columns of data (awk) II

- define custom variables and perform calculations
  - e.g. `awk '{sum += $1+$2} END {print sum}' histogram.dat`
  - adds all values of row 1 and row2 and prints the sum at the end
- search for strings in a file
  - `awk '/energy/' measurements.dat`
  - will print all lines where at least one row contains the string "energy"
  - string in the quotes can be a regular expression

Manual pages (`man`)

- maybe the most important command line application
- every command line tool comes with a manual page
- usage: `man [command]`, e.g. `man awk`
- displays syntax, description and all options in great detail
- shows examples

REMARK: Get used to look up information in manual pages. You are already in the terminal, getting information there is way faster than searching google!

Connecting to other computers (`ssh` and `scp`)

- high performance computing is done on clusters (networks of computers)
- you can work on multiple computers at once using your terminal as well
- log in to another computer (`ssh`)
  - `ssh user@host`
  - `ssh gross@dirac.physik.uni-leipzig.de`
    - connect from anywhere in the world to my local workstation
    - work as if I'm sitting in my office in front of it
- copying files remotely (`scp`)
  - `scp [source] [destination]`
  - same syntax as "normal" `cp`
  - e.g. `scp gross@wagner:~/simulations/data.txt ./`