Overview

- Introduction – Linux philosophy
- Command line basics – getting help
- The shell revisited: some features
- Navigating the file system
- File manipulation
- Text editing
- Various commands
- Archiving
- Groups, users, security
- Process control

1

Process control

2

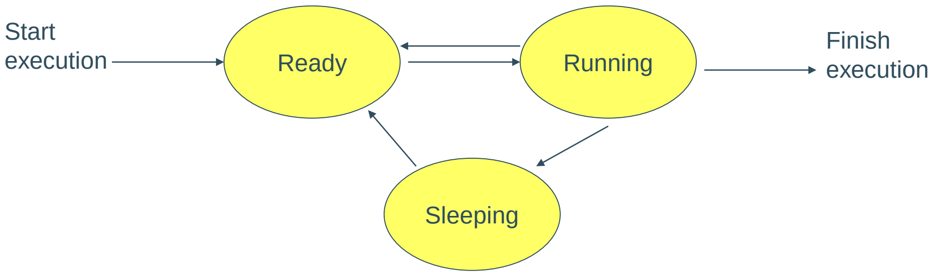
Process

- “Everything in linux is a file. Everything in linux that is not a file is a process”
- Processes
  - Instances of a running programs
  - Several instances of the same program can run at the same time
  - Processes are assigned a unique identifier which is used to monitor and control the process (PID)

3

Process

- A program that is claimed to be executing is called a process
- For a multitasking system, a process has at least the following three states:



4

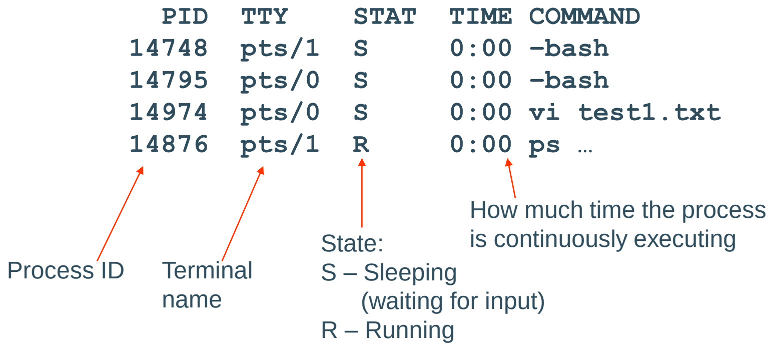
Process

- Ready state
  - All processes that are ready to execute but without the CPU are at the ready state
  - If there is only 1 CPU in the system, all processes except one are at the ready state
- Running state
  - The process that actually possesses the CPU is at the running state
  - If there is only 1 CPU in the system, there is only one process that is at the running state
- Sleeping state
  - The process that is waiting for other resources, e.g. I/O

ps

- Display running processes
  (cfr. Windows Task Manager ctrl-shift-esc)
- `$ ps` Display the current user's processes
- `$ ps -e` Display all processes running on the system
- `$ ps -ef` Display detailed information about running processes
- `$ ps -u [USER]` Display processes owned by the specified user
- `$ ps a` Display extra info (running state)

Process



```
  PID  TTY    STAT  TIME COMMAND
14748  pts/1  S     0:00 -bash
14795  pts/0  S     0:00 -bash
14974  pts/0  S     0:00 vi test1.txt
14876  pts/1  R     0:00 ps …
```

Process ID
Terminal name
State:
S – Sleeping (waiting for input)
R – Running
How much time the process is continuously executing

- For the example above, both bash processes, which are the shell of both terminals, are waiting for the input of user. They must be in the sleeping state
- The vi process, which is an editor, is also waiting for the input of user. Hence it is also in sleeping state
- When ps reporting the processes in the system, it is the only process that is running. Hence it is in running state

7

Process state codes

- Different values possible :
  - D uninterruptible sleep (usually IO)
  - R running or runnable (on run queue)
  - S interruptible sleep (waiting for an event to complete)
  - T stopped, either by a job control signal or because it is being traced.
  - X dead (should never be seen)
  - Z defunct ("zombie") process, terminated but not reaped by its parent.

8

kill

- Sends an abort signal to the given processes. Lets processes save data and exit by themselves. Should be used first.
- `$ kill <pid>`

  Example:

  ```
  $ kill 3039 3134 3190 3416
  ```
- `$ kill -9 <pid>`

  Sends an immediate termination signal. The system itself terminates the processes. Useful when a process is really stuck.

killall

- The `killall` command terminates all processes that match the specified name
- `$ killall [-<signal>] <command>`

  Example:

  ```
  $ killall bash
  ```

xkill

`xkill`

Lets you kill a graphical application by clicking on it!

Very quick! Convenient when you don't know the application command name.

11

&

- **&** is a command line operator that instructs the shell to start the specified program in the background.
- This allows you to have more than one program running at the same time without having to start multiple terminal sessions.
- Starting a process in background: add & at the end of your line:

  ```
  $ gedit &
  ```

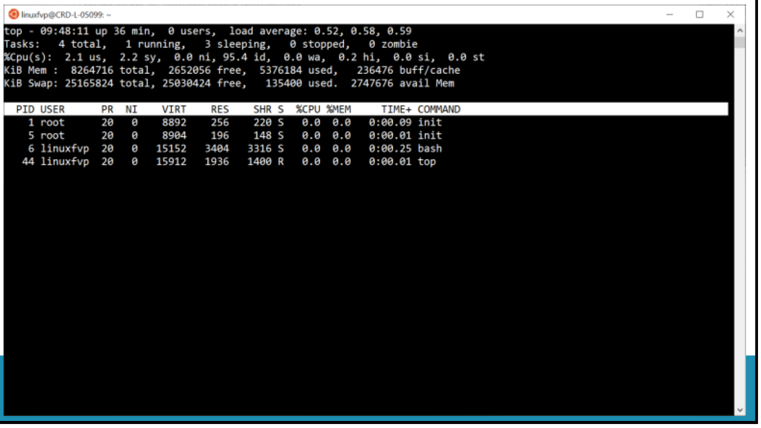
  check with `ps`

12

top

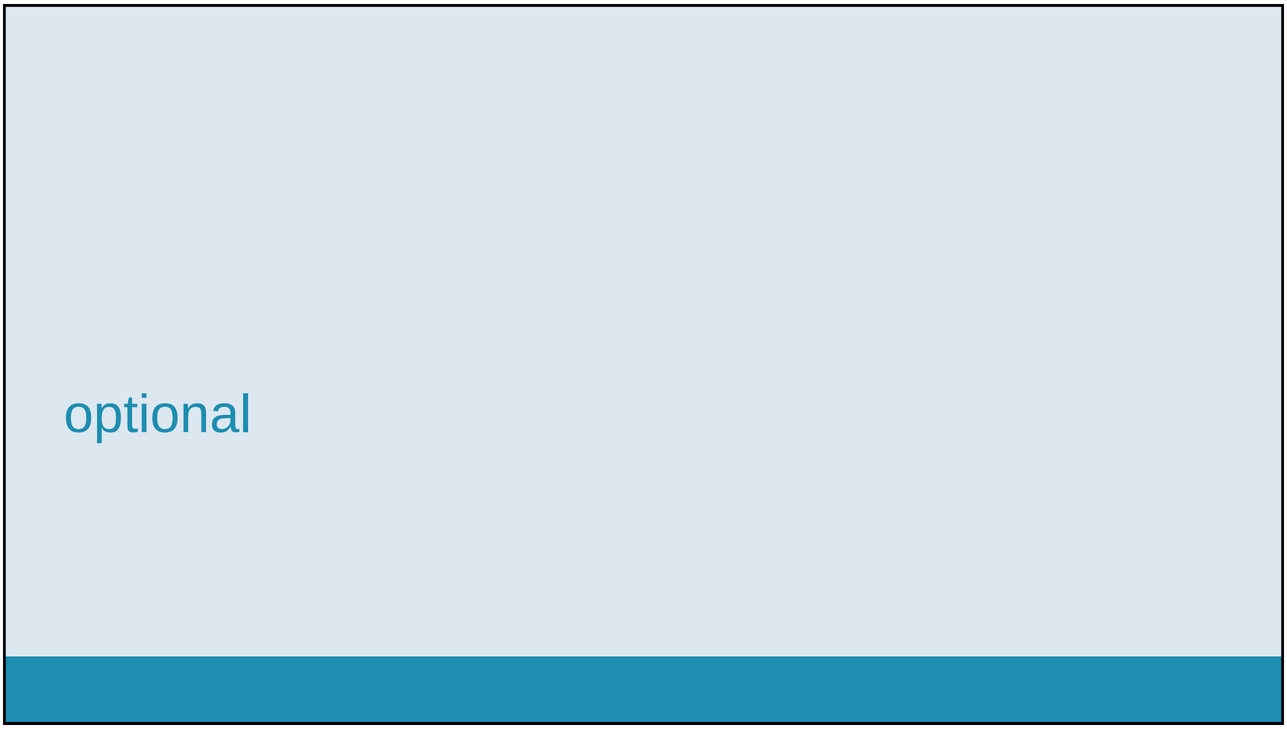
- Displays most important processes, sorted by cpu percentage (use > or < to change the order)

  http://www.thegeekstuff.com/2010/01/15-practical-unix-linux-top-command-examples/



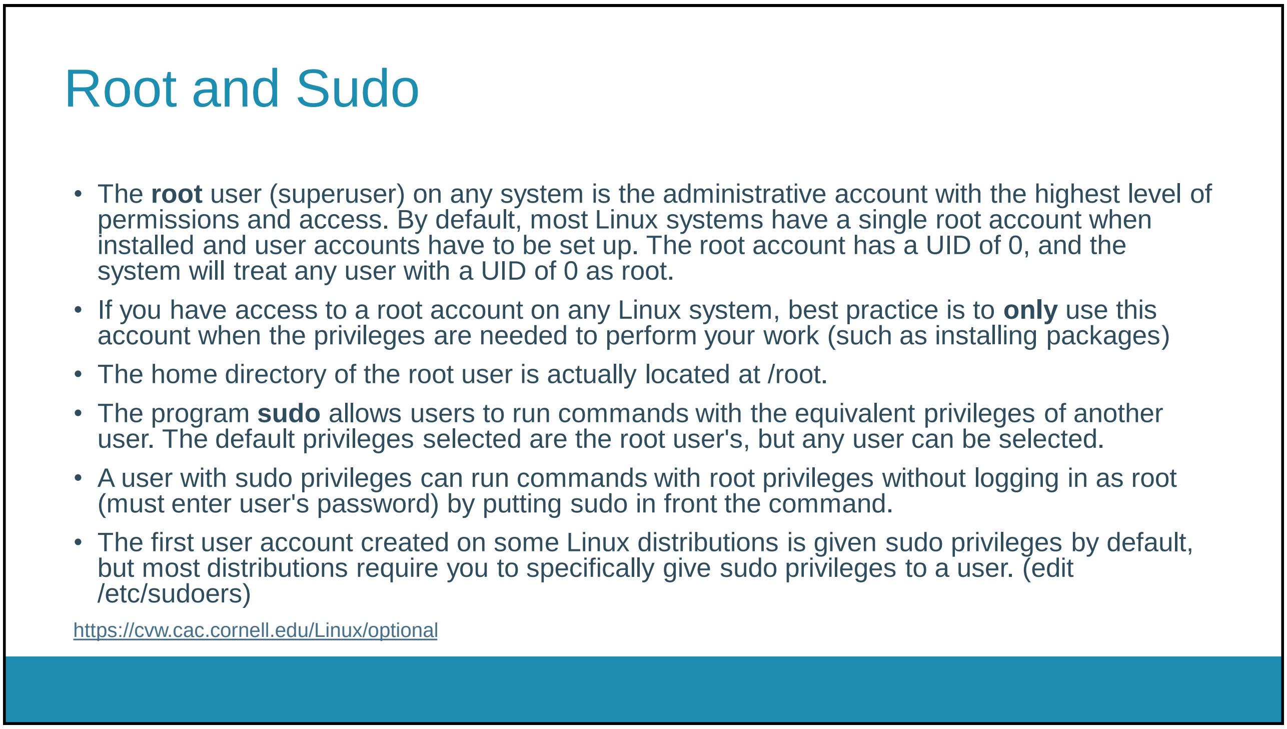
```
top - 09:48:11 up 36 min,  0 users,  load average: 0.52, 0.58, 0.59
Tasks:   4 total,   1 running,   3 sleeping,   0 stopped,   0 zombie
%Cpu(s):  2.1 us,  2.2 sy,  0.0 ni, 95.4 id,  0.0 wa,  0.2 hi,  0.0 si,  0.0 st
KiB Mem :  8264716 total,  2652056 free,  5376184 used,   236476 buff/cache
KiB Swap: 25165824 total, 25030424 free,   135400 used.  2747676 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
    1 root      20   0    8892    256    220 S   0.0  0.0   0:00.09 init
    5 root      20   0    8904    196    148 S   0.0  0.0   0:00.01 init
    6 linuxfvp  20   0   15152   3404   3316 S   0.0  0.0   0:00.25 bash
   44 linuxfvp  20   0   15912   1936   1400 R   0.0  0.0   0:00.01 top
```

- `uptime`: show the system load and how long the system has been up



optional

15

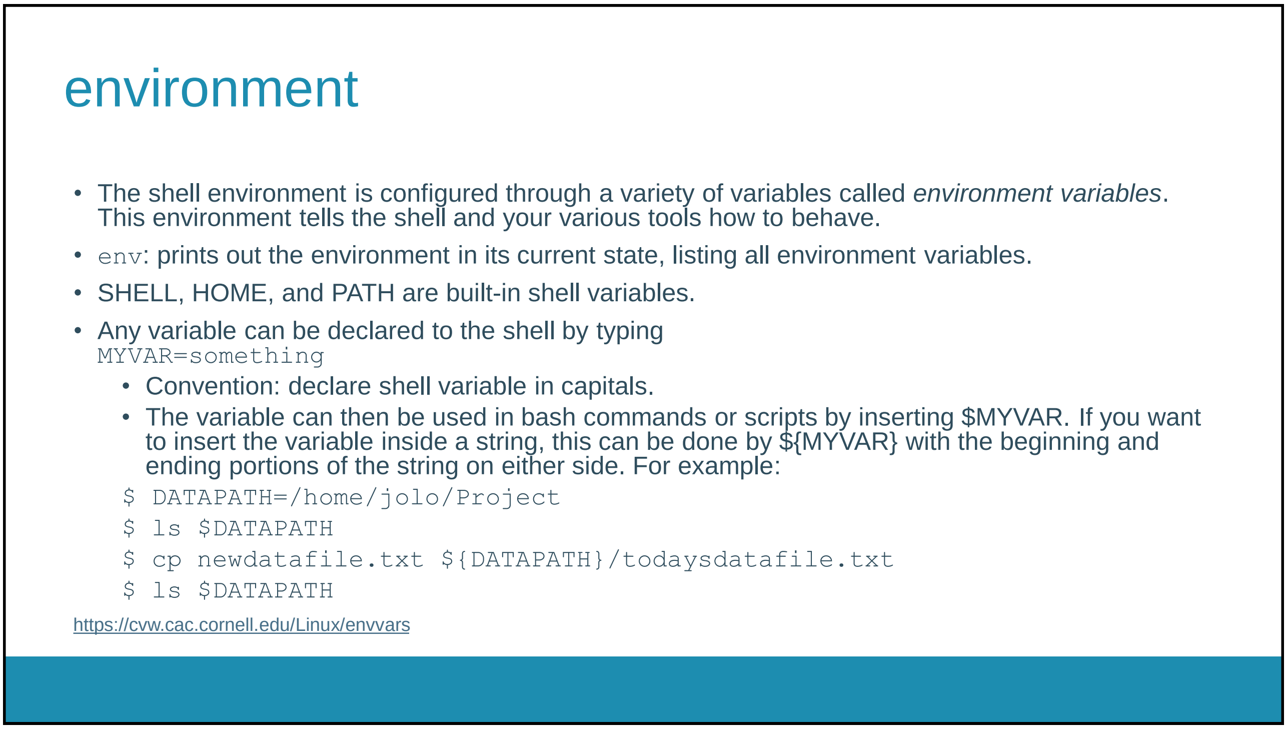


Root and Sudo

- The **root** user (superuser) on any system is the administrative account with the highest level of permissions and access. By default, most Linux systems have a single root account when installed and user accounts have to be set up. The root account has a UID of 0, and the system will treat any user with a UID of 0 as root.
- If you have access to a root account on any Linux system, best practice is to **only** use this account when the privileges are needed to perform your work (such as installing packages)
- The home directory of the root user is actually located at /root.
- The program **sudo** allows users to run commands with the equivalent privileges of another user. The default privileges selected are the root user's, but any user can be selected.
- A user with sudo privileges can run commands with root privileges without logging in as root (must enter user's password) by putting sudo in front the command.
- The first user account created on some Linux distributions is given sudo privileges by default, but most distributions require you to specifically give sudo privileges to a user. (edit /etc/sudoers)

https://cvw.cac.cornell.edu/Linux/optional

16



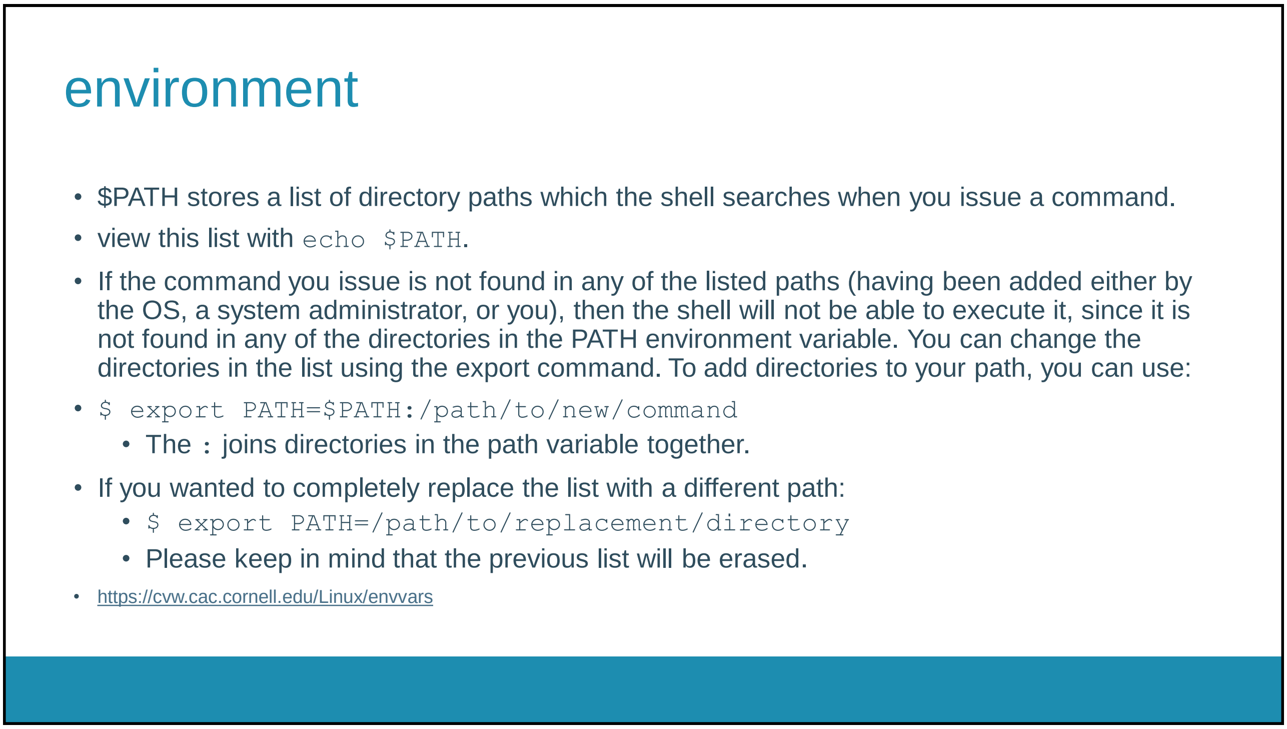
environment

- The shell environment is configured through a variety of variables called *environment variables*. This environment tells the shell and your various tools how to behave.
- `env`: prints out the environment in its current state, listing all environment variables.
- SHELL, HOME, and PATH are built-in shell variables.
- Any variable can be declared to the shell by typing `MYVAR=something`
  - Convention: declare shell variable in capitals.
  - The variable can then be used in bash commands or scripts by inserting $MYVAR. If you want to insert the variable inside a string, this can be done by ${MYVAR} with the beginning and ending portions of the string on either side. For example:

```
$ DATAPATH=/home/jolo/Project
$ ls $DATAPATH
$ cp newdatafile.txt ${DATAPATH}/todaysdatafile.txt
$ ls $DATAPATH
```

https://cvw.cac.cornell.edu/Linux/envvars

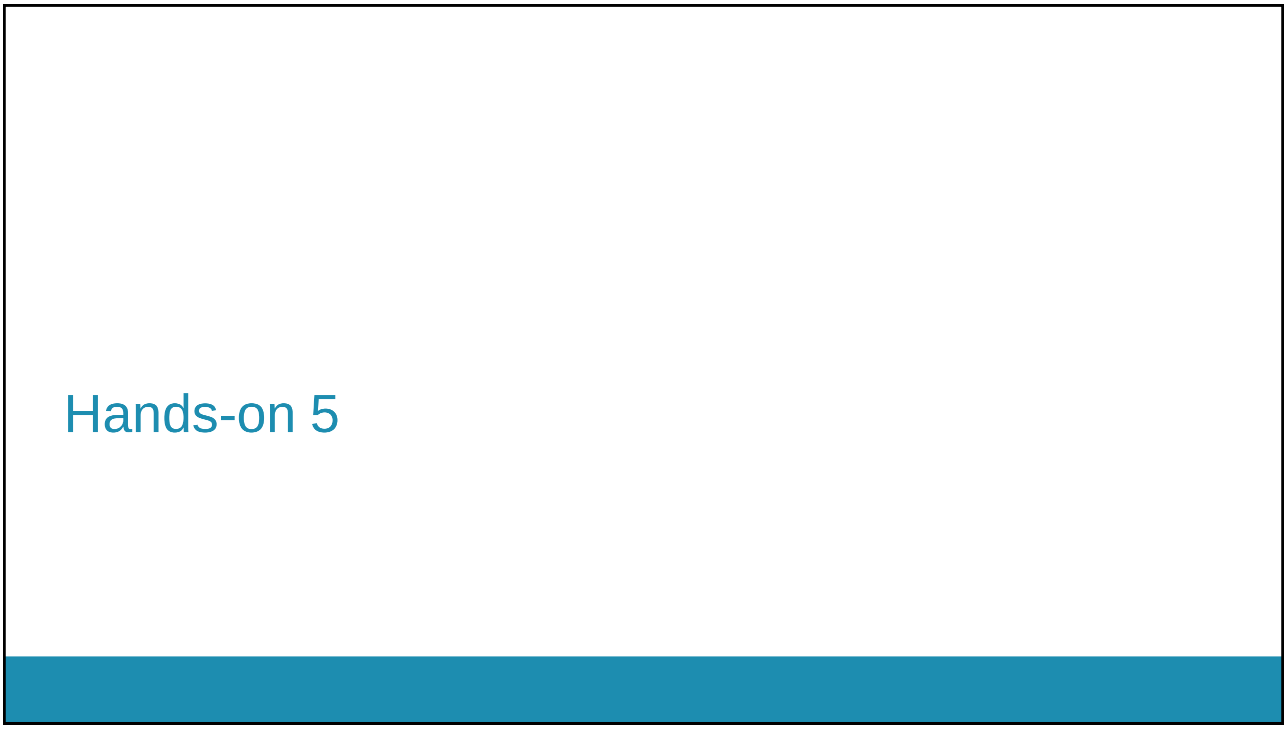
17



environment

- $PATH stores a list of directory paths which the shell searches when you issue a command.
- view this list with `echo $PATH`.
- If the command you issue is not found in any of the listed paths (having been added either by the OS, a system administrator, or you), then the shell will not be able to execute it, since it is not found in any of the directories in the PATH environment variable. You can change the directories in the list using the export command. To add directories to your path, you can use:
- `$ export PATH=$PATH:/path/to/new/command`
  - The `:` joins directories in the path variable together.
- If you wanted to completely replace the list with a different path:
  - `$ export PATH=/path/to/replacement/directory`
  - Please keep in mind that the previous list will be erased.
- https://cvw.cac.cornell.edu/Linux/envvars

18



Hands-on 5

19