Project 1
Supplementary info + Q&A
Project 1 supplementary info

- Your shell must support the `cd` built-in command!
- However, `ls` is just a normal command
- No other built-ins than `cd` are required as of now
  - Tough, remember I hate `copy-pasting`
The Shell

Useful commands
Variables

% n=5
% echo $n
5
% h="Hello you"
% echo "The equivalent for $h in french is Salut toi"
The equivalent for Hello you in french is Salut toi
% echo "${n}Bits"
5Bits

Environment variables

% echo $HOME
/home/tom
% echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/...
% echo $[
   dis_stack
   HISTFILE
   LOGNAME
   prompt_themes
   dis_aliases
   history
   MACHTYPE
   PS1
   dis_builtins
   history_color
   MAIL
   PS2
   dis_functions
   history_prompt1
   MAILCHECK
   ...
Math

```
% echo $((5 * 10))
50
% n=5
% echo $((n * 10))
50
% echo $(( (n * 10) % 5))
0
```

Terminal

From stdin:

```
% echo "5 * 10" | bc
50
```

Terminal
seq, sort, shuf

% seq 5
1
2
3
4
5

% seq 5 | shuf
3
2
1
4
5

% seq 5 | shuf | sort
1
2
3
4
5

Terminal
uniq

% echo $(RANDOM % 5) >> rands.txt
% echo $(RANDOM % 5) >> rands.txt
% echo $(RANDOM % 5) >> rands.txt
% echo $(RANDOM % 5) >> rands.txt
% echo $(RANDOM % 5) >> rands.txt
% cat rands.txt

0
0
1
0
4

% cat rands.txt | uniq

0
1
0
4

% cat rands.txt | sort | uniq

0
1
4
wc, cut

% wc rands.txt
5 5 10 rands.txt
% wc -l rands.txt
5 rands.txt
% wc -l rands.txt | cut -f1 -d' '
5

% echo "hello:world:." | cut -f2 -d'.'
world

number of lines, number of words, number of characters and file

Terminal
More, Less, Most

- Scroll through the content of a file or a pipe
- Search in it using a pattern (vim-like, type / then the pattern)
head, tail

% seq 5 > seqs.txt
% head -n 2 seqs.txt
1
2
% tail -n 2 seqs.txt
4
5

Terminal

% sudo tail -n 50 -f /var/log/syslog
[50 last lines of syslog]
[does not return and watch for more content !]

Terminal
grep

- The most important of all
- By default, show the lines matching a certain pattern
- Useful parameters:
  - `grep -E "[a-z]+"` → Regex
  - `grep -oE "[a-z]+"` → Regex and show only the matching part
  - `grep -oiE "[a-z]+"` → Idem, case insensitive

```
% sudo cat /var/log/syslog | grep -oE ".*[0-9]{1,3}.*\[[0-9]{1,3}\]" | sort | uniq
109.88.217.64
188.189.92.184
192.168.0.1
192.168.0.2
```
Command substitution

% n=$(seq 5)
% echo $n
1
2
3
4
5

% ip=$(sudo cat /var/log/syslog | grep -oE "[0-9]{1,3}(\.[0-9]{1,3}){3}" | sort | uniq | head -n 1)
% ping $ip
PING 109.88.217.64 56(84) bytes of data.
64 bytes from ams16s30-in-f3.1e100.net (1109.88.217.64): icmp_seq=1 ttl=51 time=7.83 ms
Strings, filenames

% h="Hello you ."
% echo ${#h}
11
% echo ${h/you/him}
Hello him .
% echo ${h##*lo}
him .
% echo ${h%lo*}
Hel

Remove longest match from the beginning

Remove longest match from the end

% h="/home/tom/file.txt"
% basename $h
file.txt
% dirname $h
/home/tom
% echo ${h##*.}
txt
% echo {$(basename $h)%.*}
file
Find

% find .
[all objects in all subfolders]
% find . -type f
[all files in all subfolders]
% find . -maxdepth 2 -type f
[all files in this folder and one level deeper]
% find . -maxdepth 2 -type f | xargs echo
[all files in this folder and one level deeper on one line]
% find . -mindepth 1 -maxdepth 1 -type d | xargs ls
[display content of all directories in this folder]
% find . -maxdepth 1 -type f -exec wc {} \;
[launch wc on all files of the current folder]
Chain of commands

You can also put a program in background

% echo "hello" ; echo "you"
hello
you
% echo "hello" && echo "you"
hello
you
% echo "hello you" > file.txt
% grep "hello" file.txt &>/dev/null && echo "Found hello"
Found hello
% grep "hello" file.txt &>/dev/null || echo "Could not find hello in file.txt"
Could not find hello in file

% sudo tail -f /var/log/messages &
% fg

% sudo tail -f /var/log/messages
[CTRL-Z]
bg
Exercices

1. Write 6 random numbers between 0 and 9 in a file named `randoms` then display them in a sorted way on the screen.
2. Display only the unique numbers of the file on the screen.
3. Add the string: “This is a string” at the end of the file. What will return the command `wc {your file}`?
4. You have `h=“Hello you. Do you enjoyed this course ?”`. In one line display: I enjoyed this course.
You can make little scripts to avoid retyping all commands. Just create a file which starts by `#!/bin/bash` and simply puts your commands one by line.

Example:

```bash
#!/bin/sh

cd /target
make
make install

% chmod +x script.sh
% ./script.sh
...```
### Condition

<table>
<thead>
<tr>
<th>Script</th>
<th>Description</th>
</tr>
</thead>
</table>
| ```bash
#!/bin/sh
if grep "salut" file.txt ; then
echo "Salut is present!"
endif
``` | If uses the return code, not stdout! |
| ```bash
#!/bin/sh
grep "salut" file.txt 2>&1 /dev/null
if [ $? -eq 0 ] ; then
echo "Salut is present!"
endif
``` | `?` is the return code of the last cmd |
| ```bash
#!/bin/sh
number=7
if [ $number -lt 10 ] ; then
echo "$number is smaller than 10"
endif
``` | If uses the return code, not stdout! |
[thing] is just a shortcut for test thing

% man test
[...]

( EXPRESSION )
EXPRESSON is true

! EXPRESSION
EXPRESSON is false

EXPRESSION1 -a EXPRESSION2
both EXPRESSION1 and EXPRESSION2 are true

EXPRESSION1 -o EXPRESSION2
either EXPRESSION1 or EXPRESSION2 is true

-n STRING
the length of STRING is nonzero

STRING equivalent to -n STRING
...

Terminal
# Loops

<table>
<thead>
<tr>
<th>Shell Script</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>#!/bin/sh</code></td>
<td></td>
</tr>
<tr>
<td>n=0</td>
<td></td>
</tr>
<tr>
<td>while [ $n -lt 10 ]; do</td>
<td></td>
</tr>
<tr>
<td>echo &quot;$n&quot;</td>
<td></td>
</tr>
<tr>
<td>n=$((n + 1))</td>
<td></td>
</tr>
<tr>
<td>done</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Shell Script script.sh</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Shell Script</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>#!/bin/sh</code></td>
<td></td>
</tr>
<tr>
<td>for n in $(seq 10); do</td>
<td></td>
</tr>
<tr>
<td>echo &quot;$n&quot;</td>
<td></td>
</tr>
<tr>
<td>done</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Shell Script script.sh</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Shell Script</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>#!/bin/sh</code></td>
<td></td>
</tr>
<tr>
<td>for file in ./*; do</td>
<td></td>
</tr>
<tr>
<td>stat --printf=&quot;%s\n&quot; $file</td>
<td></td>
</tr>
<tr>
<td>done</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Shell Script script.sh</td>
</tr>
</tbody>
</table>
Supplementary exercises

1. Write a bash script which will print all the file that contains a specific string
2. Write a bash script that compute the result of the operation given in argument
Kernel Interaction
What is the Kernel

- A very big program, but still, a program
- Kernel vs OS
  - Linux is the Kernel of Linux distributions
  - XNU is Mac OS's kernel
  - Windows NT's kernel is kind of Windows NT's kernel, and makes the difference harder as the window manager is part of the kernel (hybrid kernel)
- So that explains a lot of problem in wordings (last's weeks "Linux fundamentals" slides is in fact the tools part while today we could say we'll speak about Linux fundamentals, the kernel part.)
Kernel Entry

- Interrupts
- System calls
  - Why not a simple library/function call?
  - Cost?
- (Memory mappings)

- How to read this "big program"'s state
/proc

A virtual filesystem that reports kernel's internal state

- `ls /proc` list most the general state.
- `cat /proc/cpuinfo` is all you need about your CPU
- `/proc/{number}` holds info about process `{number}`
/proc/1/status

Name: systemd
State: S (sleeping)
Tgid: 1
Ngid: 0
Pid: 1
PPid: 0
TracerPid: 0
Uid: 0 0 0 0
Gid: 0 0 0 0
FDSize: 256
Groups:
NStgid: 1
NSpid: 1
NSpgid: 1
NSsid: 1
VmPeak: 251112 kB
VmSize: 185788 kB
VmLck: 0 kB
VmPin: 0 kB
VmHWM: 6452 kB
VmRSS: 4912 kB
VmData: 149540 kB
VmStk: 136 kB
VmExe: 1392 kB

1475 struct task_struct {
1483 volatile long state; /* -1 unrunnable, 0 runnable, >0 stopped */
1484 void *stack;
1485 atomic_t usage;
1486 unsigned int flags; /* per process flags, defined below */
1487 unsigned int ptrace;
1488
1489 #ifdef CONFIG_SMP
1490 struct list_node wake_entry;
1491 int on_cpu;
1492 #ifdef CONFIG_THREAD_INFO_IN_TASK
1493 unsigned int cpu; /* current CPU */
1494 #endif
1495 unsigned int wakee_flips;
1496 unsigned long wakee_flip_decay_ts;
1497 struct task_struct *last_wakee;
1498 #endif
1499 #ifdef CONFIG_CGROUP_SCHED
1500 struct task_group *sched_task_group;
1501 #endif
1502 struct sched_dl_entity dl;
1503 #ifdef CONFIG_PREEMPT_NOTIFIERS
1504 /* list of struct preempt_notifier: */
1505 struct hlist_head preempt_notifiers;
1506 #endif
1507 #ifdef CONFIG_BLK_DEV_IO_TRACE
1508 unsigned int btrace_seq;
1509 #endif
1510 unsigned int policy;
1511 int nr_cpus_allowed;
1512 cpumask_t cpus_allowed;
1513 const struct sched_class *sched_class;
1514 struct sched_entity se;
/sys

- /proc is for process-related things
- /sys is for the whole system
- So why cpuinfo etc are in /proc?
  - Like everything else in Linux: it's a mess
- /sys/module : Modules parameters
- /sys/bus : Access devices per-bus

```
student $ ls /sys/bus/usb/devices
1-0:1.0  1-1.3  1-1.6:1.0  2-1  2-1.6.1  2-1.6.1:1.1  usb1  usb4
1-1  1-1.3:1.0  1-1.6:1.1  2-1:1.0  2-1.6:1.0  3-0:1.0  usb2
1-1:1.0  1-1.6  2-0:1.0  2-1.6  2-1.6.1:1.0  4-0:1.0  usb3
```
What's behind sys and proc?

- They are "fake" (virtual) file systems.
- When read is called, it finds some handlers registered by kernel parts, eg:
  - "hey I'm a usb device handler, call fnt_usb_read() when a read is made on usb/deviceXXX/status"
- So at the end, it relies on the open/read system calls.
What's behind network configuration

- Set IP address
  - Manual
    - Set the IP
      - `ifconfig eth0 IP netmask MASK`
      - `ip addr add IP/CIDR dev eth0`
    - Set the DNS
      - `vi /etc/resolv.conf`
  - dhcp
    - `dhclient eth0`

- Nearly just a syscall wrapper
- Some userspace parts
- Does not directly involves the kernel
System call example: set eth0 ip address

```c
int socket(int domain, int type, int protocol);
→ Creates a socket
→ Return a File Descriptor (FD). man 2 socket for more informations

int ioctl(unsigned int fd, unsigned int cmd, unsigned long arg);
→ Call some command with some arguments on a file descriptor (here, we pass the socket fd)
```
Create partitions
- fdisk /dev/sda
- gparted /dev/sda

Format partitions
- mkfs.ext4 /dev/sda1

Mount partition
- mount /dev/sda1 /mnt/disk

Unmount partition
- umount /mnt/disk

Nearly just a syscall wrapper

Some userspace parts

Does not directly involves the kernel
Practical Course

Part 2
You will add a "sys" built-in to your shell

- `sys hostname` → Gives the hostname without using a system call
- `sys cpu model` → Gives the CPU model
- `sys cpu freq N` → Gives the CPU number N frequency
- `sys cpu freq N X` → Set the frequency of the CPU N to X (in HZ)
  - Prints nothing
- `sys ip addr DEV` → Get the ip and mask of the interface DEV
  - `a.b.c.d e.f.g.h`
- `sys ip addr DEV IP MASK` → Set the ip of the interface DEV to IP/MASK

Built-in must return error code like real software
Support variables along with $? and $! replacement
Project 2 advices

- You are (or should be) good programmers
  - No copy-pasting, use functions!
  - Pay attention to style and indentation
  - Be clean and efficient
  - Don't forget previous courses
  - C → beware of memory leaks!

- Project will have different importance, though P1 and P2 are of equivalent importance (next projects won't)