Practical 9 – Linux Administration – Analysing Processes

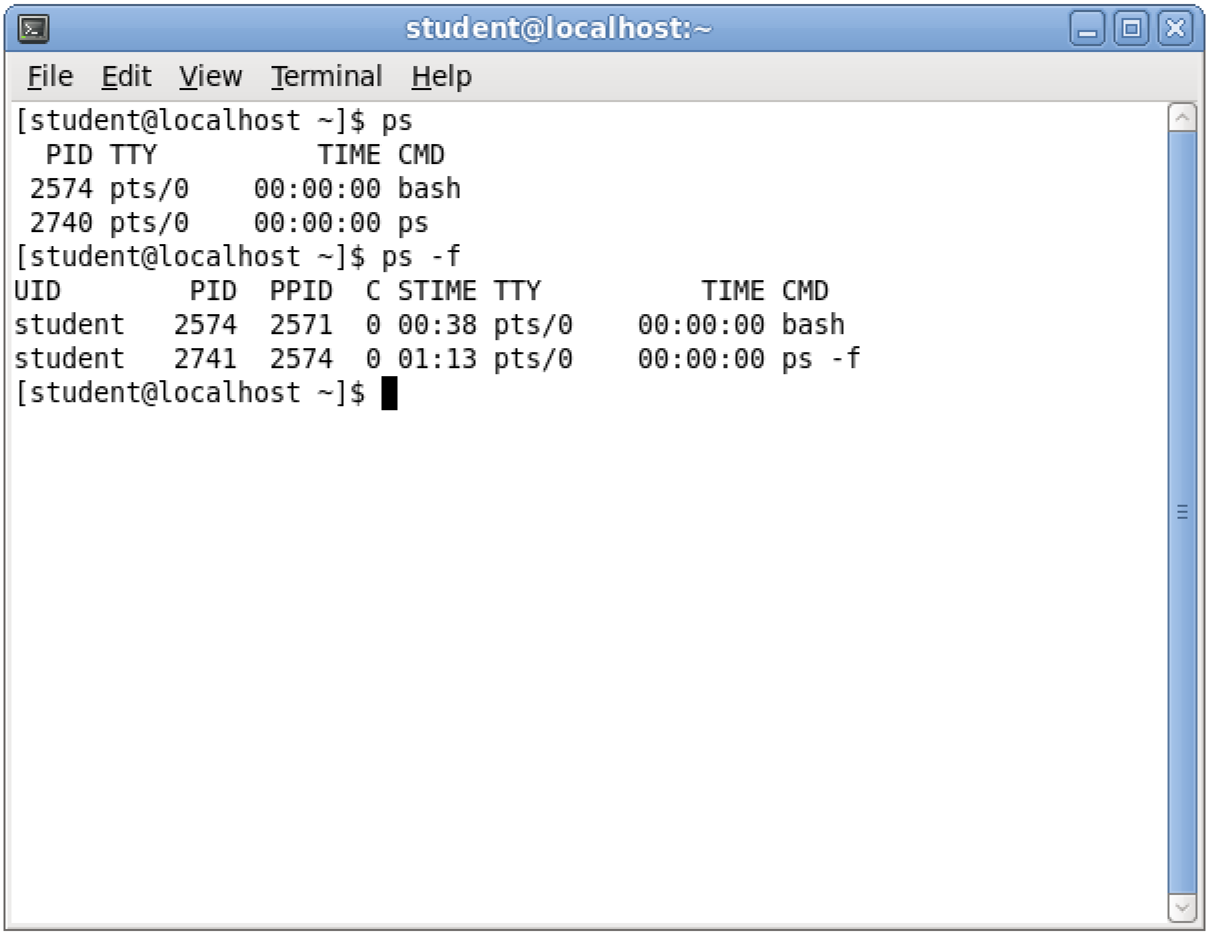
Pre-Requisites

Objectives

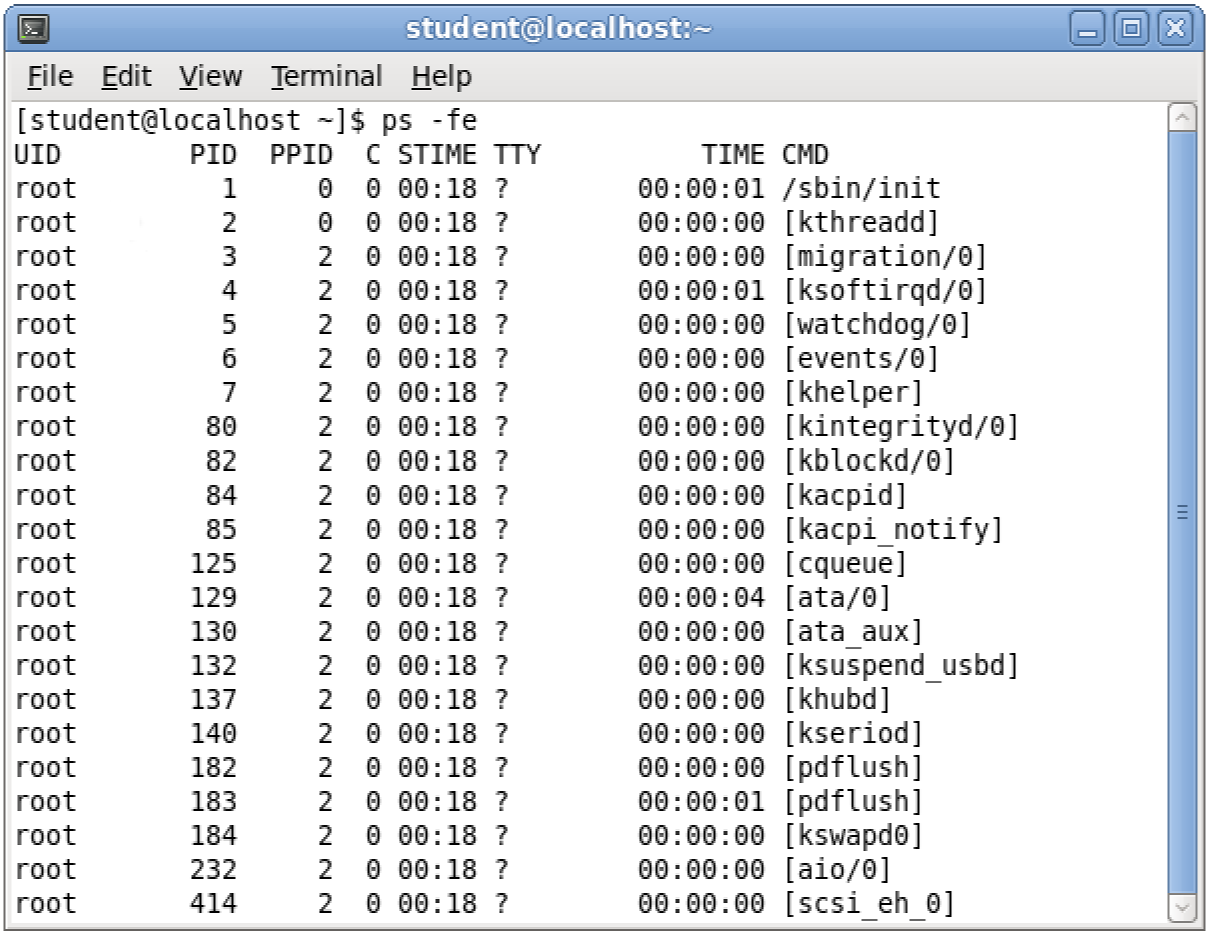
- Processes
- Jobs
- Starting background jobs
- Signals

Exercise 1 - Processes

1. Every program is runs as a process. A process is an instance of a running program.

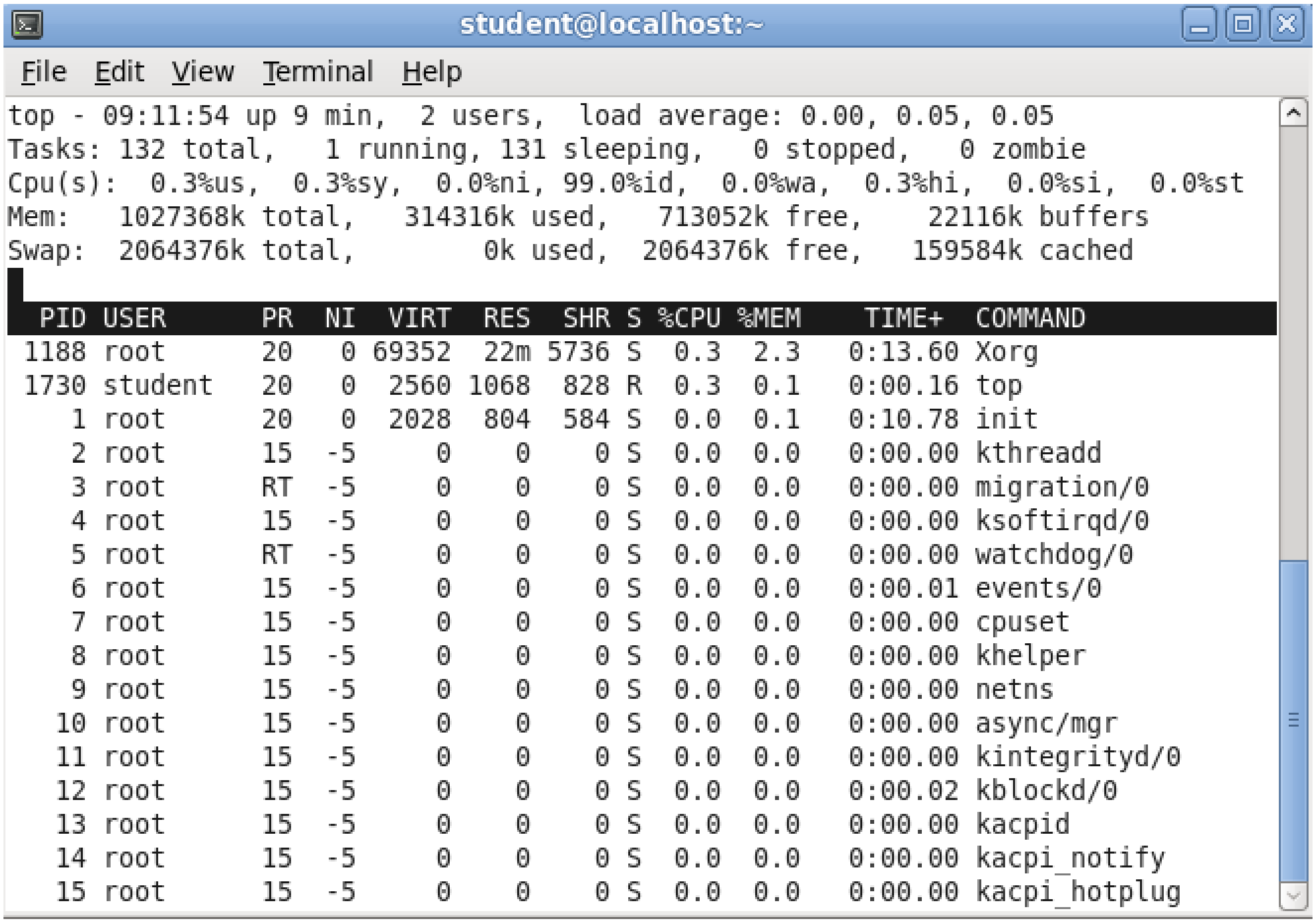


```
[student@localhost ~]$ ps
  PID TTY          TIME CMD
 2574 pts/0    00:00:00 bash
 2740 pts/0    00:00:00 ps
[student@localhost ~]$ ps -f
UID        PID  PPID  C STIME TTY          TIME CMD
student   2574  2571  0 00:38 pts/0    00:00:00 bash
student   2741  2574  0 01:13 pts/0    00:00:00 ps -f
[student@localhost ~]$ 
```



```
[student@localhost ~]$ ps -fe
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 00:18 ?        00:00:01 /sbin/init
root         2     0  0 00:18 ?        00:00:00 [kthreadd]
root         3     2  0 00:18 ?        00:00:00 [migration/0]
root         4     2  0 00:18 ?        00:00:01 [ksoftirqd/0]
root         5     2  0 00:18 ?        00:00:00 [watchdog/0]
root         6     2  0 00:18 ?        00:00:00 [events/0]
root         7     2  0 00:18 ?        00:00:00 [khelper]
root        80     2  0 00:18 ?        00:00:00 [kintegrityd/0]
root        82     2  0 00:18 ?        00:00:00 [kblockd/0]
root        84     2  0 00:18 ?        00:00:00 [kacpid]
root        85     2  0 00:18 ?        00:00:00 [kacpi_notify]
root       125     2  0 00:18 ?        00:00:00 [cqueue]
root       129     2  0 00:18 ?        00:00:04 [ata/0]
root       130     2  0 00:18 ?        00:00:00 [ata_aux]
root       132     2  0 00:18 ?        00:00:00 [ksuspend_usbd]
root       137     2  0 00:18 ?        00:00:00 [khubd]
root       140     2  0 00:18 ?        00:00:00 [kseriod]
root       182     2  0 00:18 ?        00:00:00 [pdflush]
root       183     2  0 00:18 ?        00:00:01 [pdflush]
root       184     2  0 00:18 ?        00:00:00 [kswapd0]
root       232     2  0 00:18 ?        00:00:00 [aio/0]
root       414     2  0 00:18 ?        00:00:00 [scsi_eh_0]
```

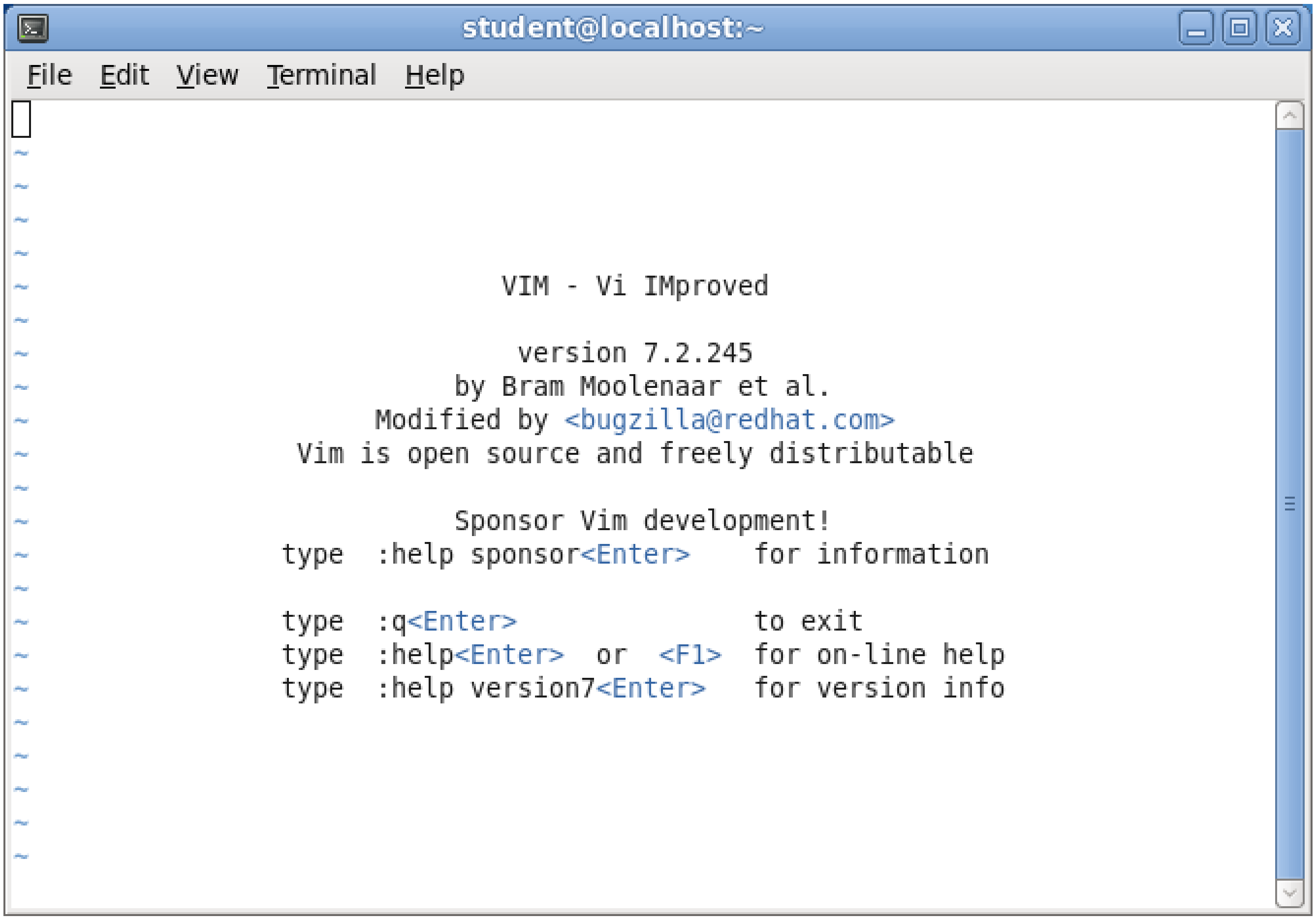
Another command to display a dynamic real-time view of processes and tasks is **top.**



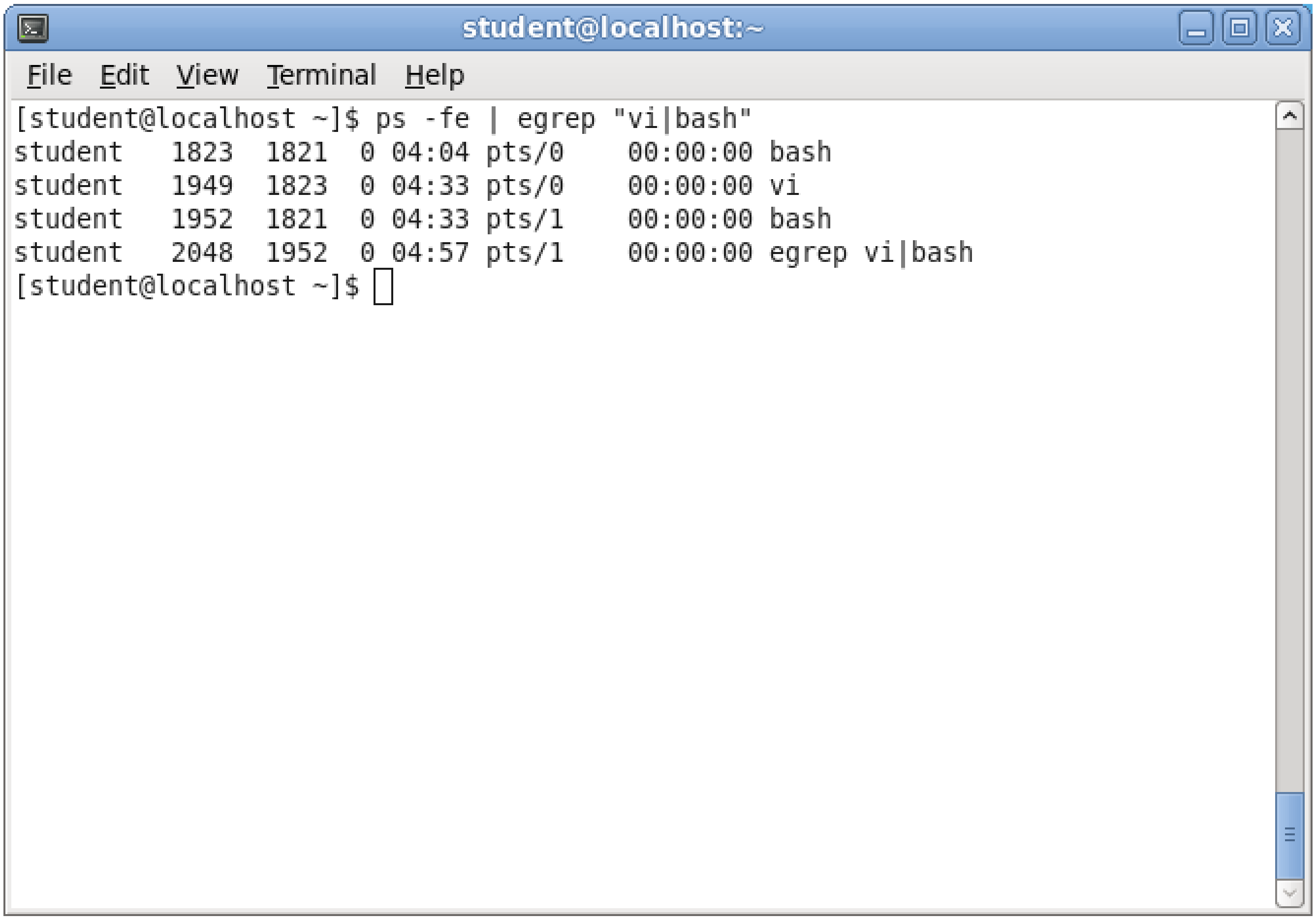
```
top - 09:11:54 up 9 min,  2 users,  load average: 0.00, 0.05, 0.05
Tasks: 132 total,   1 running, 131 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.3%us,  0.3%sy,  0.0%ni, 99.0%id,  0.0%wa,  0.3%hi,  0.0%si,  0.0%st
Mem:   1027368k total,   314316k used,   713052k free,    22116k buffers
Swap:  2064376k total,        0k used,  2064376k free,   159584k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 1188 root      20   0 69352  22m 5736 S  0.3  2.3   0:13.60 Xorg
 1730 student   20   0  2560 1068  828 R  0.3  0.1   0:00.16 top
    1 root      20   0  2028  804  584 S  0.0  0.1   0:10.78 init
    2 root      15  -5     0    0    0 S  0.0  0.0   0:00.00 kthreadd
    3 root      RT  -5     0    0    0 S  0.0  0.0   0:00.00 migration/0
    4 root      15  -5     0    0    0 S  0.0  0.0   0:00.00 ksoftirqd/0
    5 root      RT  -5     0    0    0 S  0.0  0.0   0:00.00 watchdog/0
    6 root      15  -5     0    0    0 S  0.0  0.0   0:00.01 events/0
    7 root      15  -5     0    0    0 S  0.0  0.0   0:00.00 cpuset
    8 root      15  -5     0    0    0 S  0.0  0.0   0:00.00 khelper
    9 root      15  -5     0    0    0 S  0.0  0.0   0:00.00 netns
   10 root      15  -5     0    0    0 S  0.0  0.0   0:00.00 async/mgr
   11 root      15  -5     0    0    0 S  0.0  0.0   0:00.00 kintegrityd/0
   12 root      15  -5     0    0    0 S  0.0  0.0   0:00.02 kblockd/0
   13 root      15  -5     0    0    0 S  0.0  0.0   0:00.00 kacpid
   14 root      15  -5     0    0    0 S  0.0  0.0   0:00.00 kacpi_notify
   15 root      15  -5     0    0    0 S  0.0  0.0   0:00.00 kacpi_hotplug
```

- **PID:** Shows task's unique process id.
- **PR:** Stands for priority of the task.
- **SHR:** Represents the amount of shared memory used by a task.
- **VIRT:** Total virtual memory used by the task.
- **USER:** User name of owner of task.
- **%CPU:** Represents the CPU usage.
- **TIME+:** CPU Time, the same as 'TIME', but reflecting more granularity through hundredths of a second.
- **SHR:** Represents the Shared Memory size (kb) used by a task.
- **NI:** Represents a Nice Value of task. A Negative nice value implies higher priority, and positive Nice value means lower priority.
- **%MEM:** Shows the Memory usage of task.

2. Run two terminals. Run the **vi** command in one of the terminal and run the **ps** command in the other terminal as shown.



```
~
~
~
~
~                           VIM - Vi IMproved
~
~                            version 7.2.245
~                        by Bram Moolenaar et al.
~                   Modified by <bugzilla@redhat.com>
~              Vim is open source and freely distributable
~
~                        Sponsor Vim development!
~             type  :help sponsor<Enter>    for information
~
~             type  :q<Enter>               to exit
~             type  :help<Enter>  or  <F1>  for on-line help
~             type  :help version7<Enter>   for version info
~
~
~
~
~
```



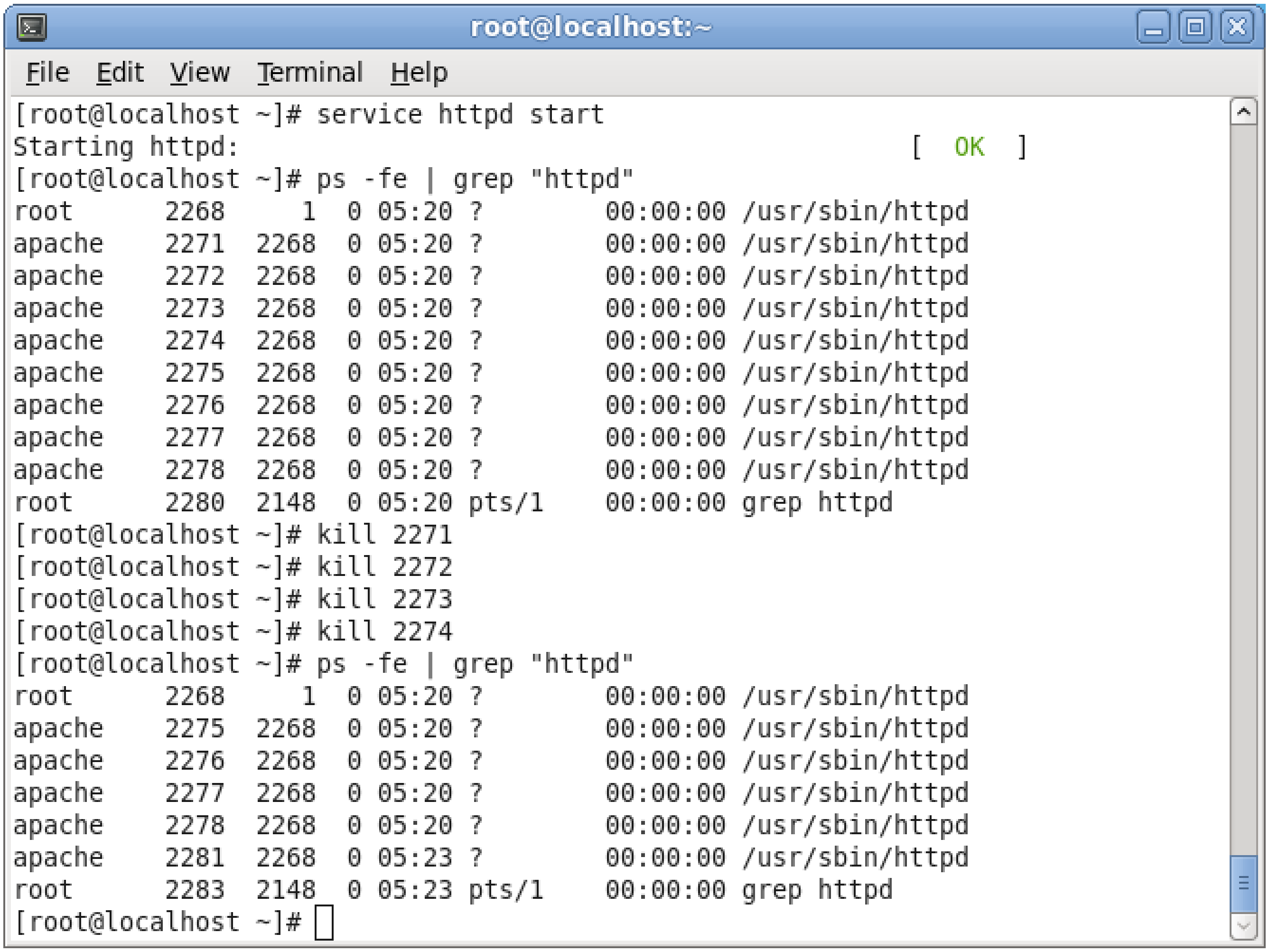
```
[student@localhost ~]$ ps -fe | egrep "vi|bash"
student   1823  1821  0 04:04 pts/0    00:00:00 bash
student   1949  1823  0 04:33 pts/0    00:00:00 vi
student   1952  1821  0 04:33 pts/1    00:00:00 bash
student   2048  1952  0 04:57 pts/1    00:00:00 egrep vi|bash
[student@localhost ~]$ 
```

Question

How can you identify conclusively the process ID (PID) for each of the bash shell?

3. Switch to the root user account. Run the commands shown to start the web server and identify all the httpd child processes.

   Run the **kill** command to terminate the child processes.

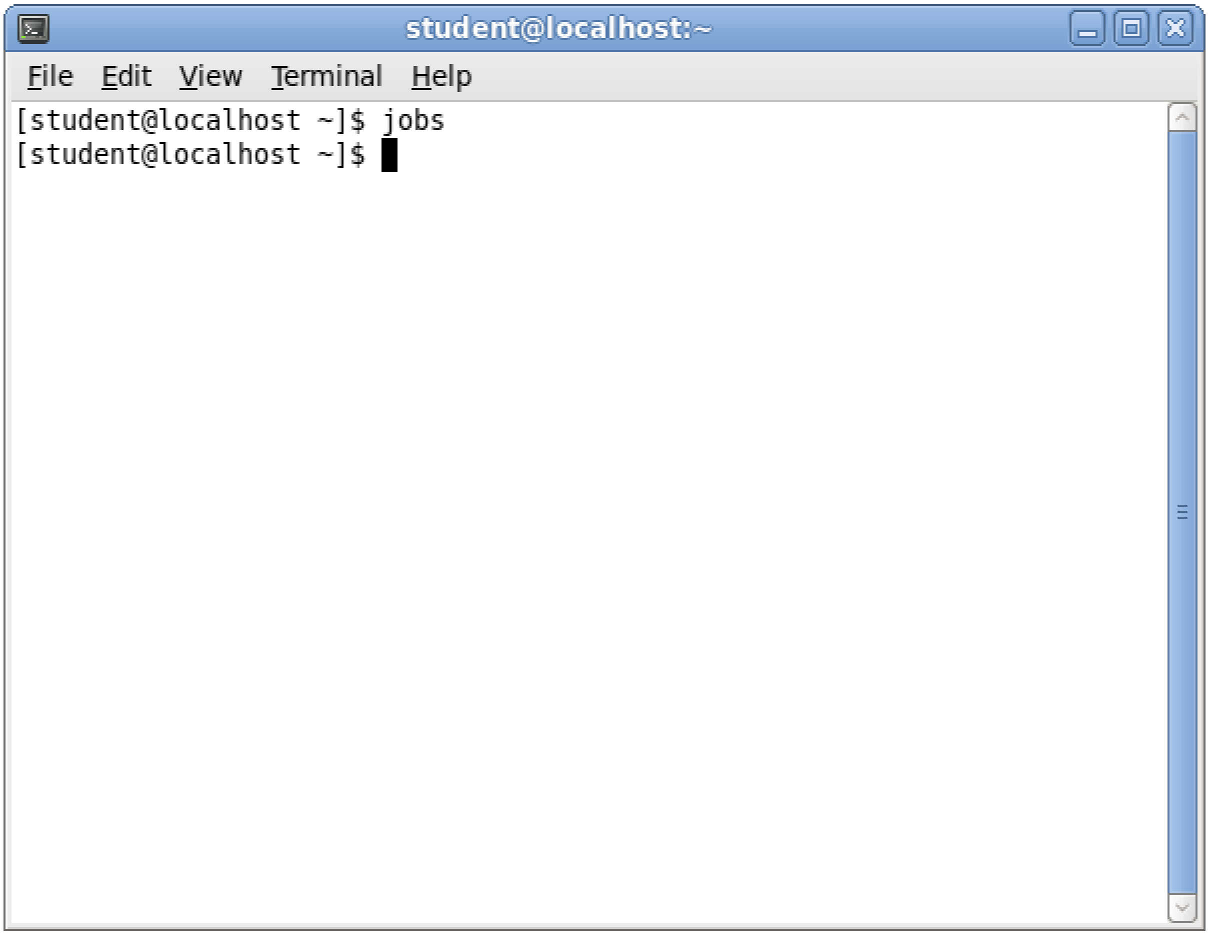


```
[root@localhost ~]# service httpd start
Starting httpd:                                            [  OK  ]
[root@localhost ~]# ps -fe | grep "httpd"
root      2268     1  0 05:20 ?        00:00:00 /usr/sbin/httpd
apache    2271  2268  0 05:20 ?        00:00:00 /usr/sbin/httpd
apache    2272  2268  0 05:20 ?        00:00:00 /usr/sbin/httpd
apache    2273  2268  0 05:20 ?        00:00:00 /usr/sbin/httpd
apache    2274  2268  0 05:20 ?        00:00:00 /usr/sbin/httpd
apache    2275  2268  0 05:20 ?        00:00:00 /usr/sbin/httpd
apache    2276  2268  0 05:20 ?        00:00:00 /usr/sbin/httpd
apache    2277  2268  0 05:20 ?        00:00:00 /usr/sbin/httpd
apache    2278  2268  0 05:20 ?        00:00:00 /usr/sbin/httpd
root      2280  2148  0 05:20 pts/1    00:00:00 grep httpd
[root@localhost ~]# kill 2271
[root@localhost ~]# kill 2272
[root@localhost ~]# kill 2273
[root@localhost ~]# kill 2274
[root@localhost ~]# ps -fe | grep "httpd"
root      2268     1  0 05:20 ?        00:00:00 /usr/sbin/httpd
apache    2275  2268  0 05:20 ?        00:00:00 /usr/sbin/httpd
apache    2276  2268  0 05:20 ?        00:00:00 /usr/sbin/httpd
apache    2277  2268  0 05:20 ?        00:00:00 /usr/sbin/httpd
apache    2278  2268  0 05:20 ?        00:00:00 /usr/sbin/httpd
apache    2281  2268  0 05:23 ?        00:00:00 /usr/sbin/httpd
root      2283  2148  0 05:23 pts/1    00:00:00 grep httpd
[root@localhost ~]# 
```

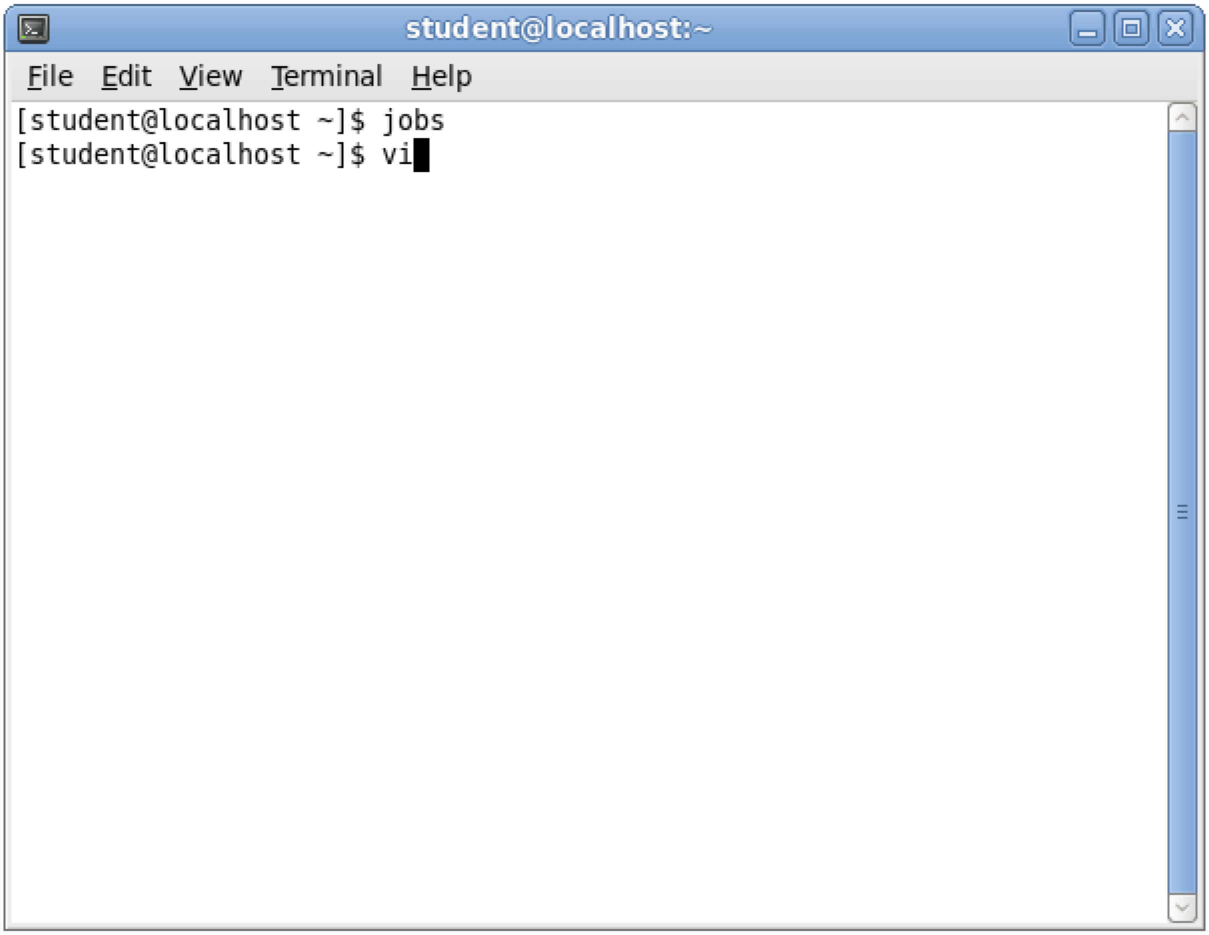
Exercise 2 - Jobs

1. In general, a job is a background execution initiated through the shell.

   Run the **jobs** to list all jobs.

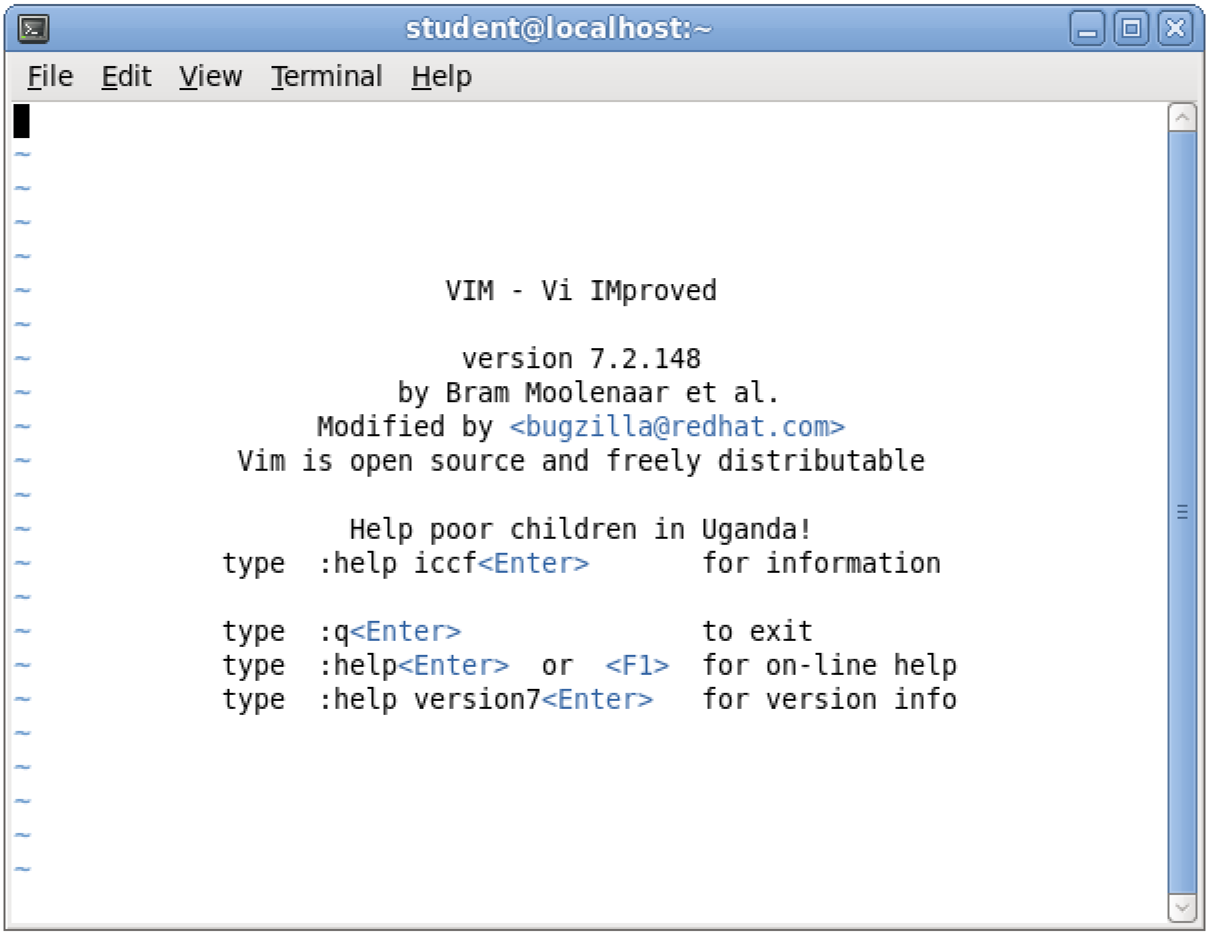


```
[student@localhost ~]$ jobs
[student@localhost ~]$ 
```

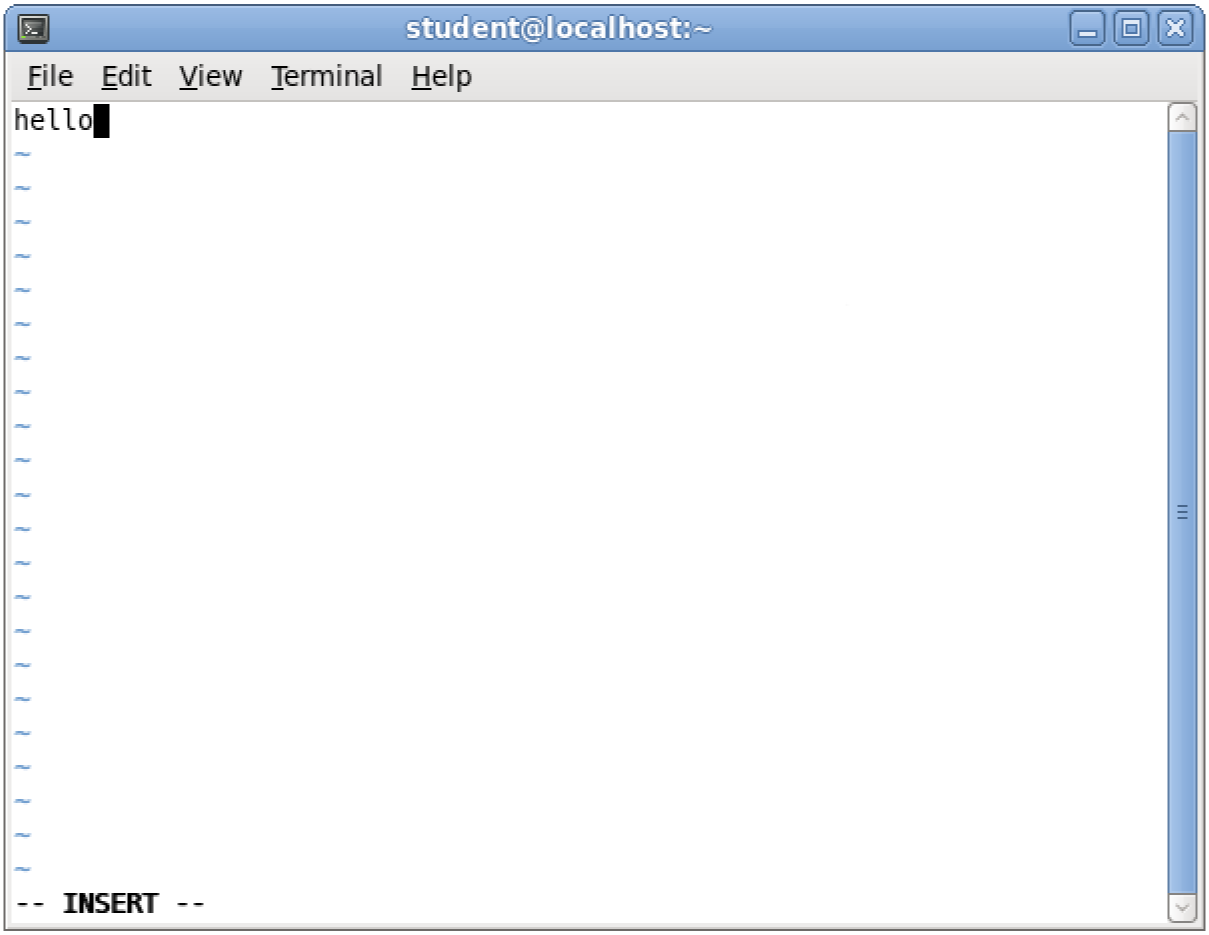


```
[student@localhost ~]$ jobs
[student@localhost ~]$ vi
```

Run **vi** and enter some text.

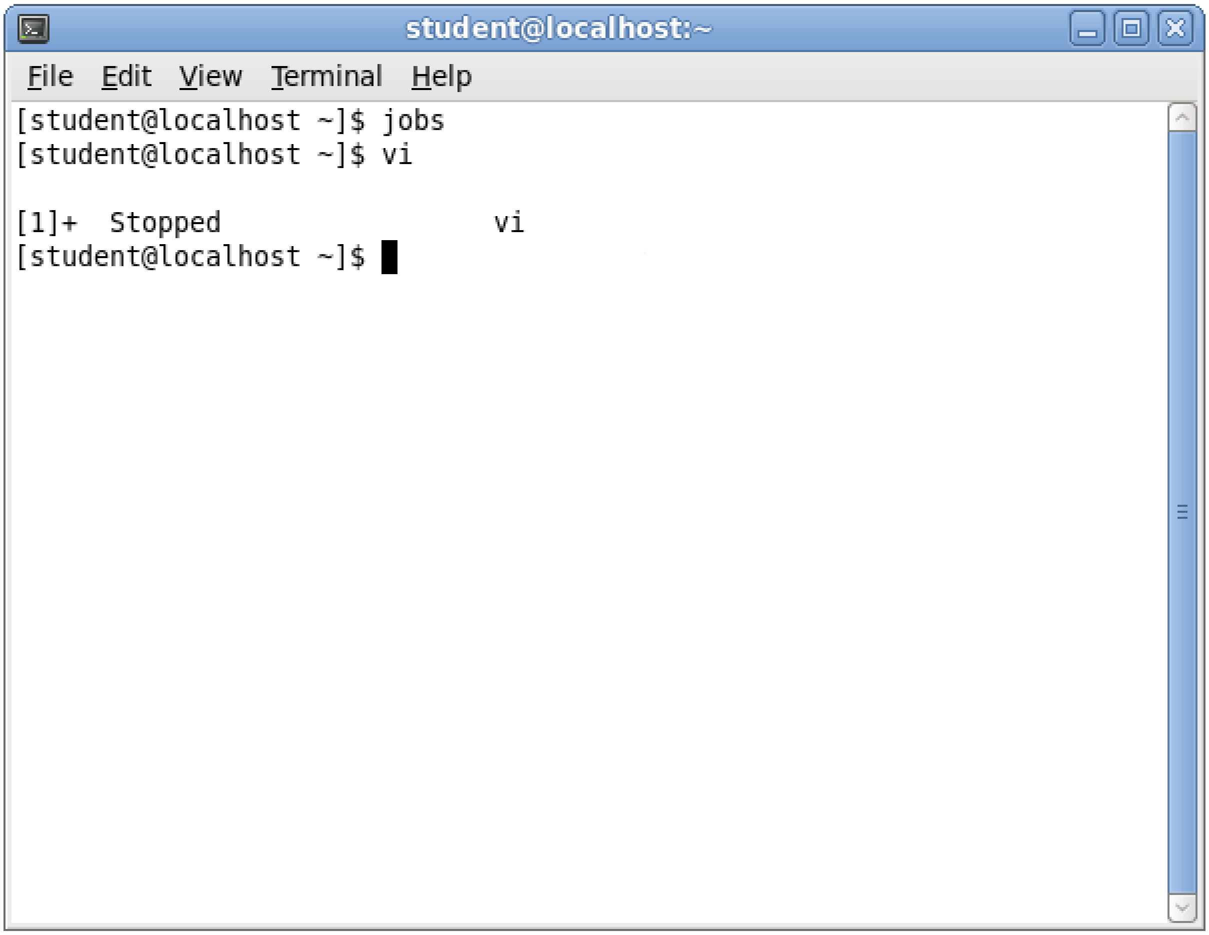


```

~
~
~
~
~                          VIM - Vi IMproved
~
~                           version 7.2.148
~                       by Bram Moolenaar et al.
~                  Modified by <bugzilla@redhat.com>
~             Vim is open source and freely distributable
~
~                    Help poor children in Uganda!
~            type  :help iccf<Enter>       for information
~
~            type  :q<Enter>               to exit
~            type  :help<Enter>  or  <F1>  for on-line help
~            type  :help version7<Enter>   for version info
~
~
~
~
~
```



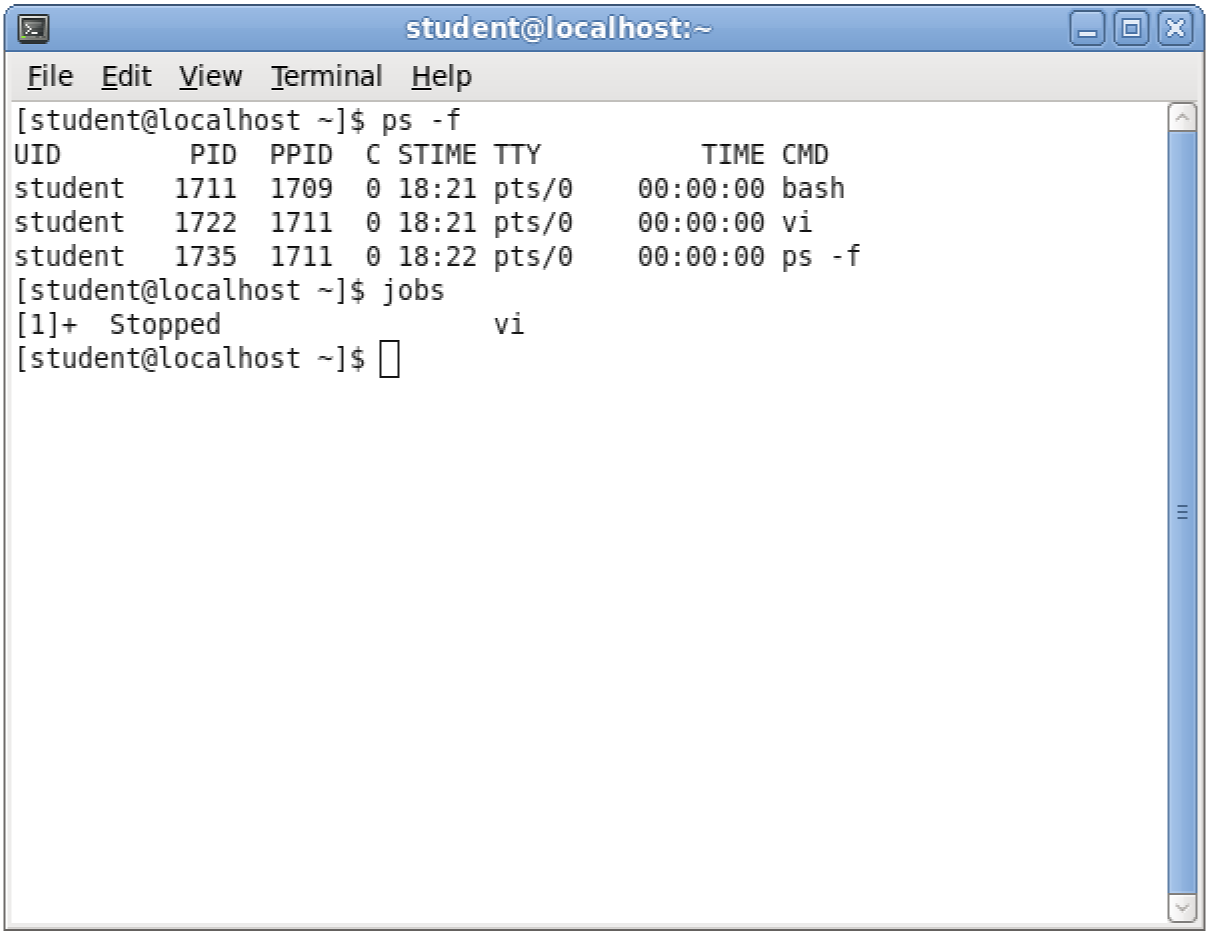
```
hello
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
-- INSERT --
```

Press Esc, then CTRL-Z to switch back to the prompt.



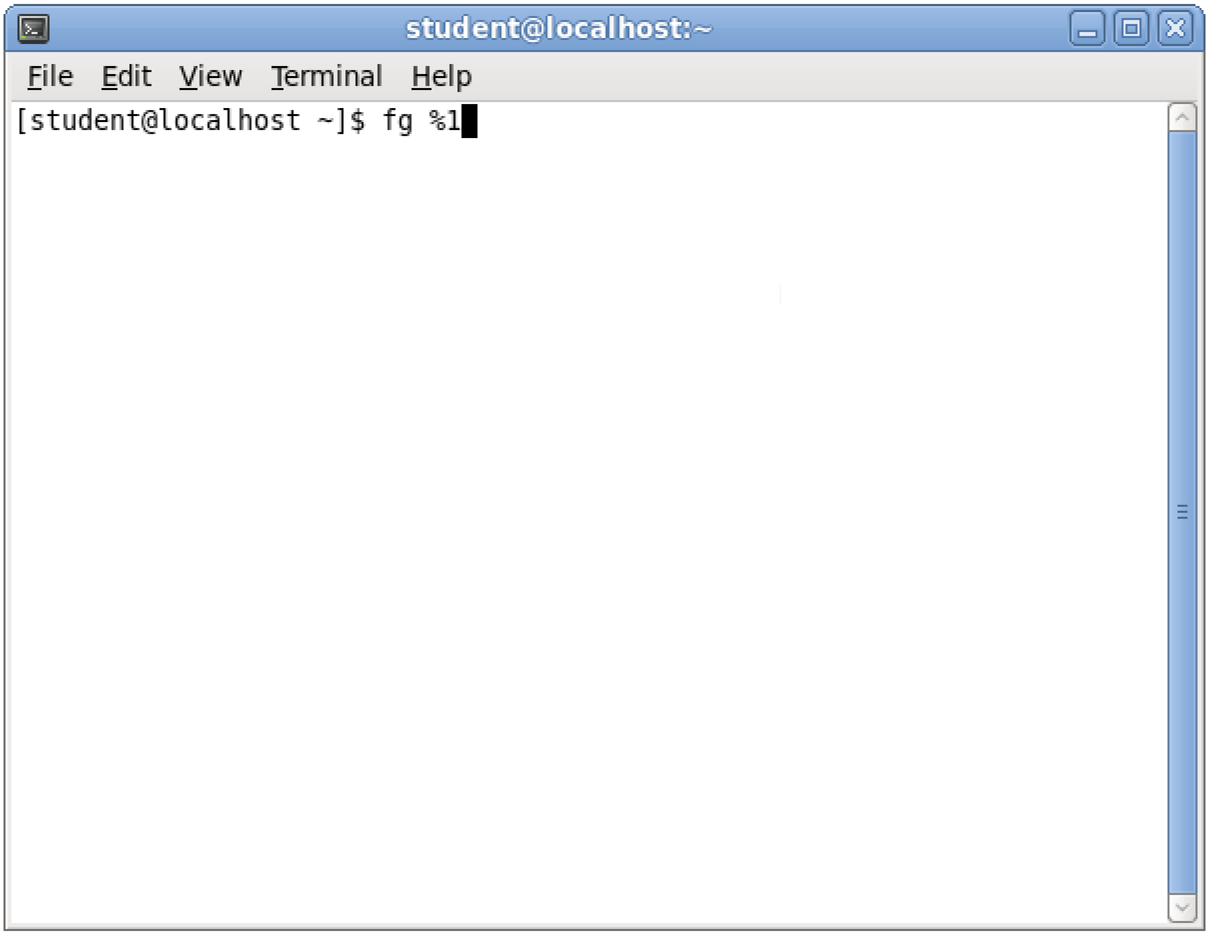
```
[student@localhost ~]$ jobs
[student@localhost ~]$ vi

[1]+  Stopped                 vi
[student@localhost ~]$ 
```

Run the **ps** and the **jobs** commands to check if **vi** is terminated.

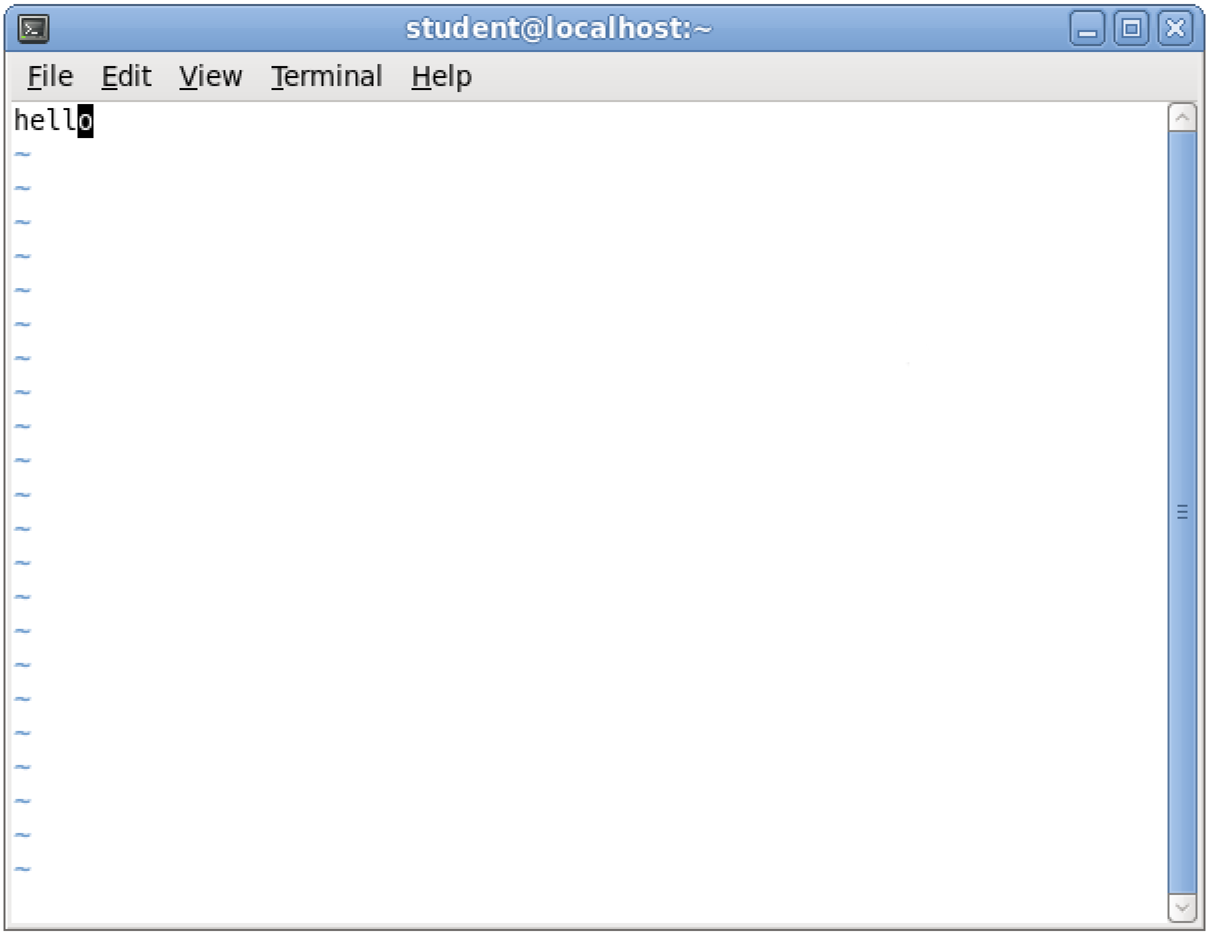


```
[student@localhost ~]$ ps -f
UID        PID  PPID  C STIME TTY          TIME CMD
student   1711  1709  0 18:21 pts/0    00:00:00 bash
student   1722  1711  0 18:21 pts/0    00:00:00 vi
student   1735  1711  0 18:22 pts/0    00:00:00 ps -f
[student@localhost ~]$ jobs
[1]+  Stopped                 vi
[student@localhost ~]$ 
```

Run the **fg** command to bring the process to the foreground. The parameter "%1" is the job number listed by the **jobs** command.

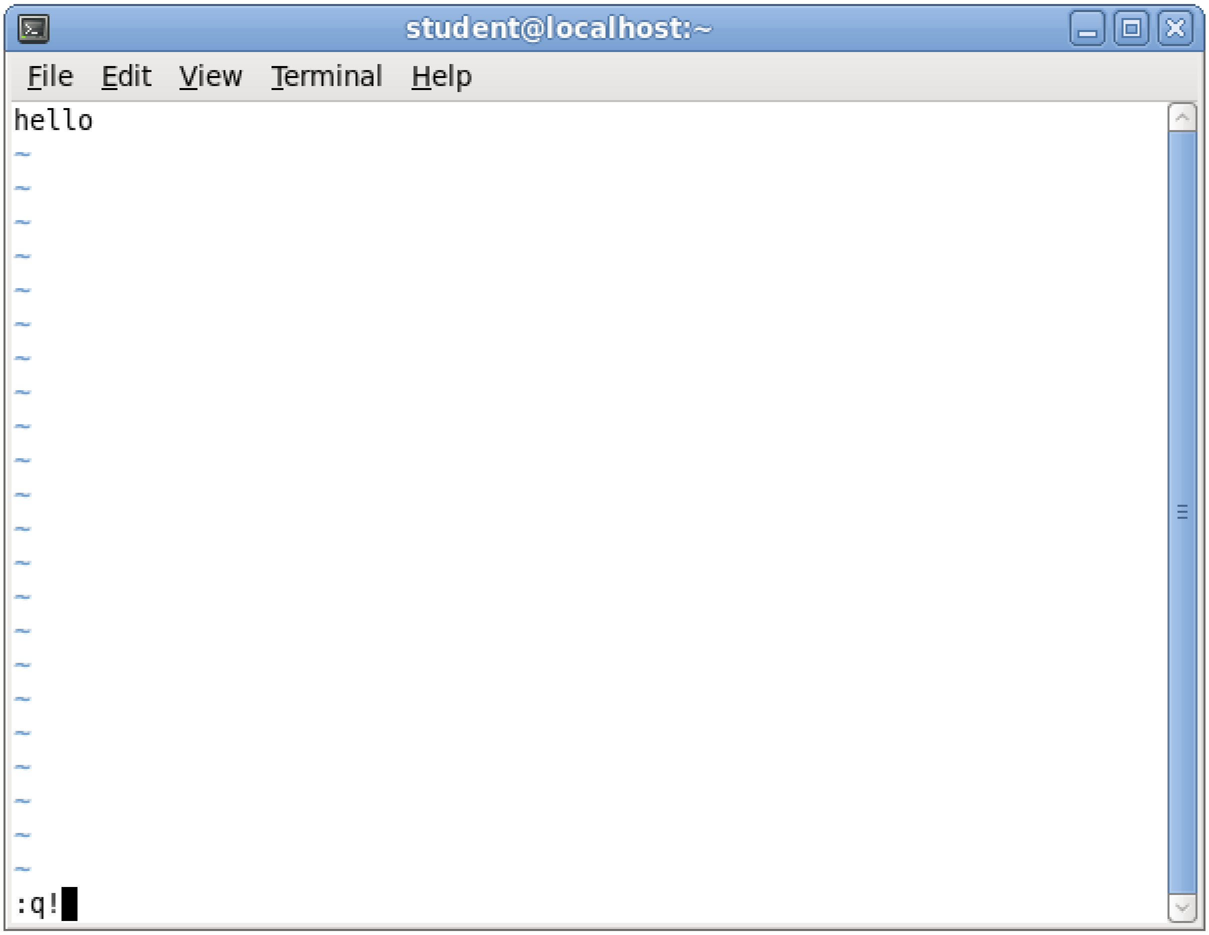


```
[student@localhost ~]$ fg %1
```

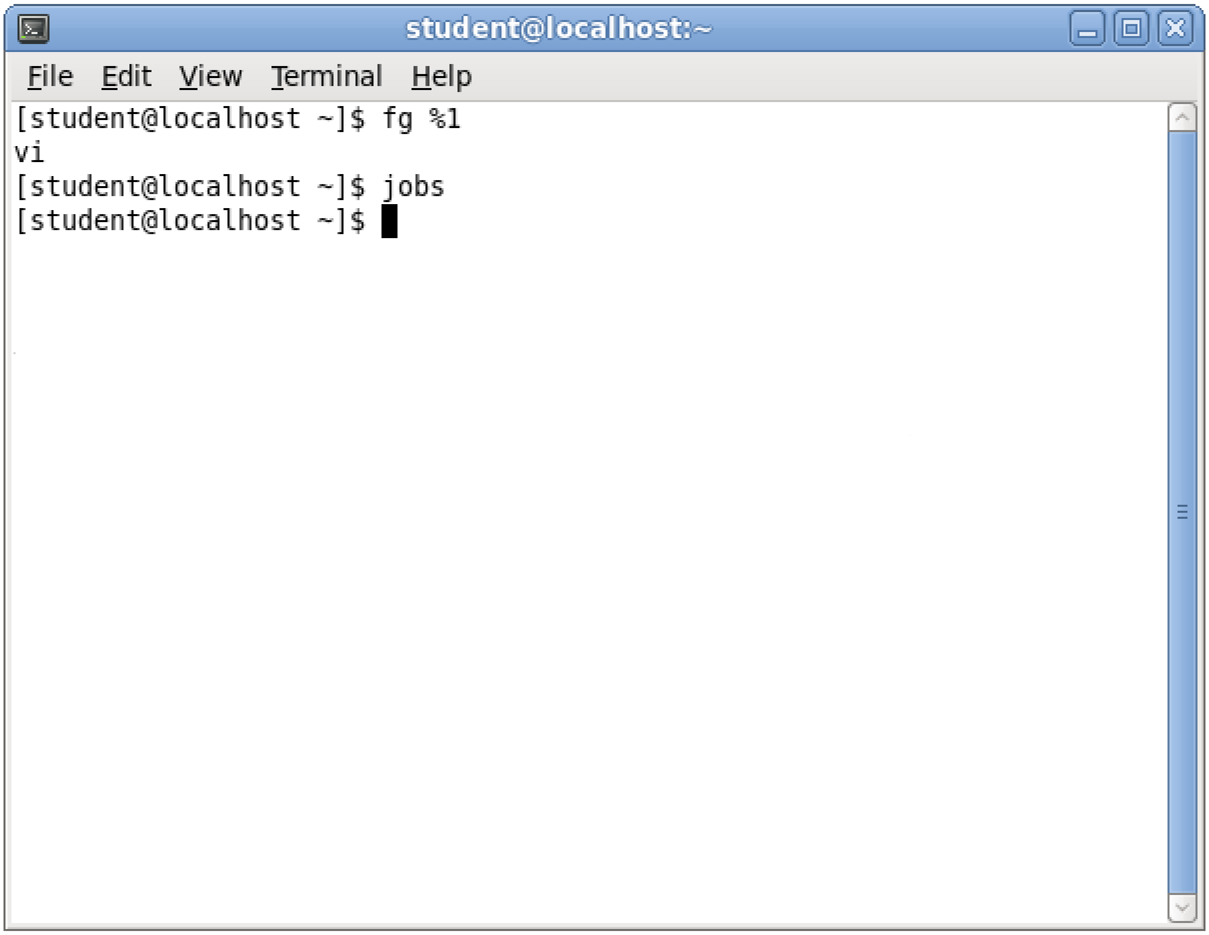


```
hello
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
```

Quit **vi** and run the **jobs** command list the current jobs.



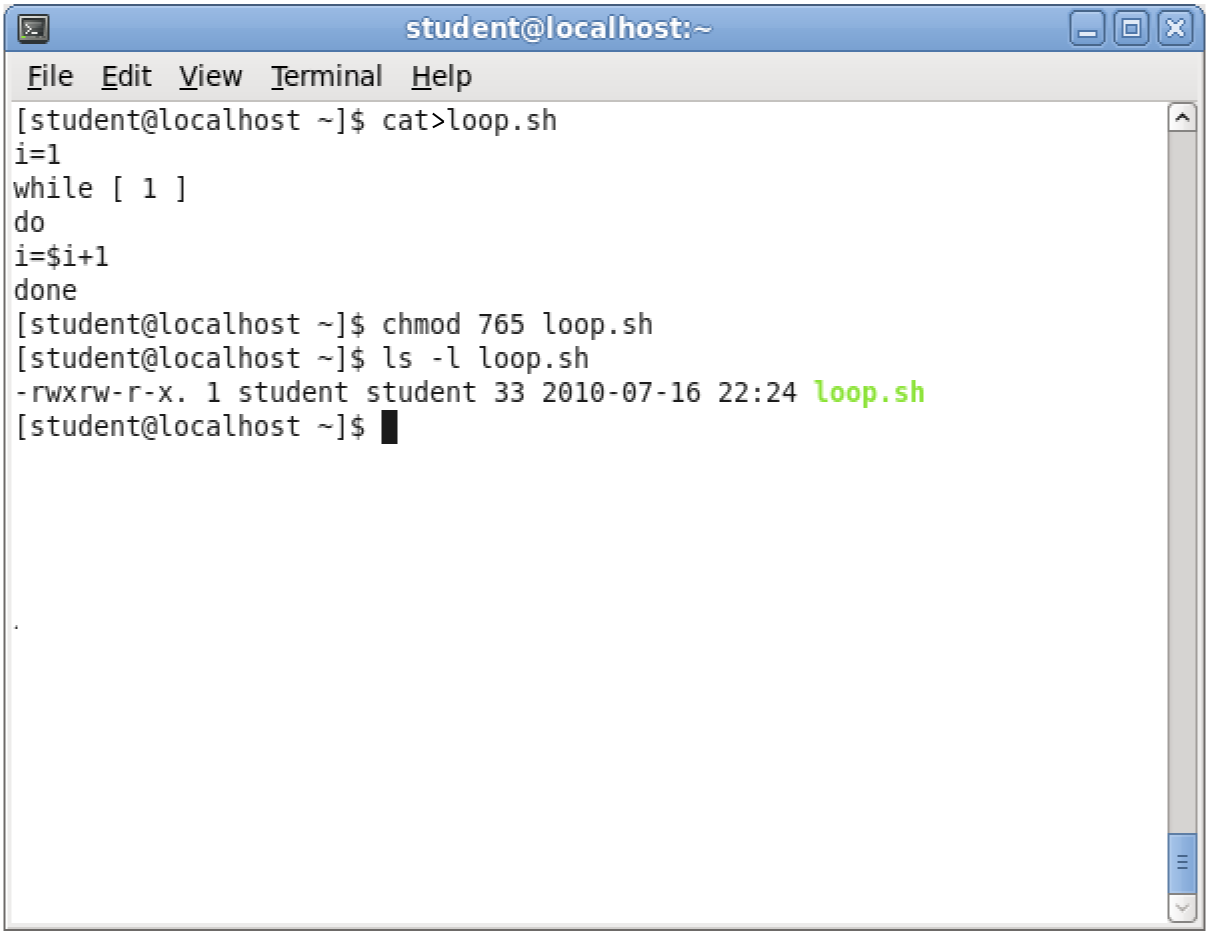
```
hello
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
:q!
```



```
[student@localhost ~]$ fg %1
vi
[student@localhost ~]$ jobs
[student@localhost ~]$ 
```

Exercise 3 - Starting background jobs

1. Create the "loop.sh" script with an editor and set the execution permission for the user.

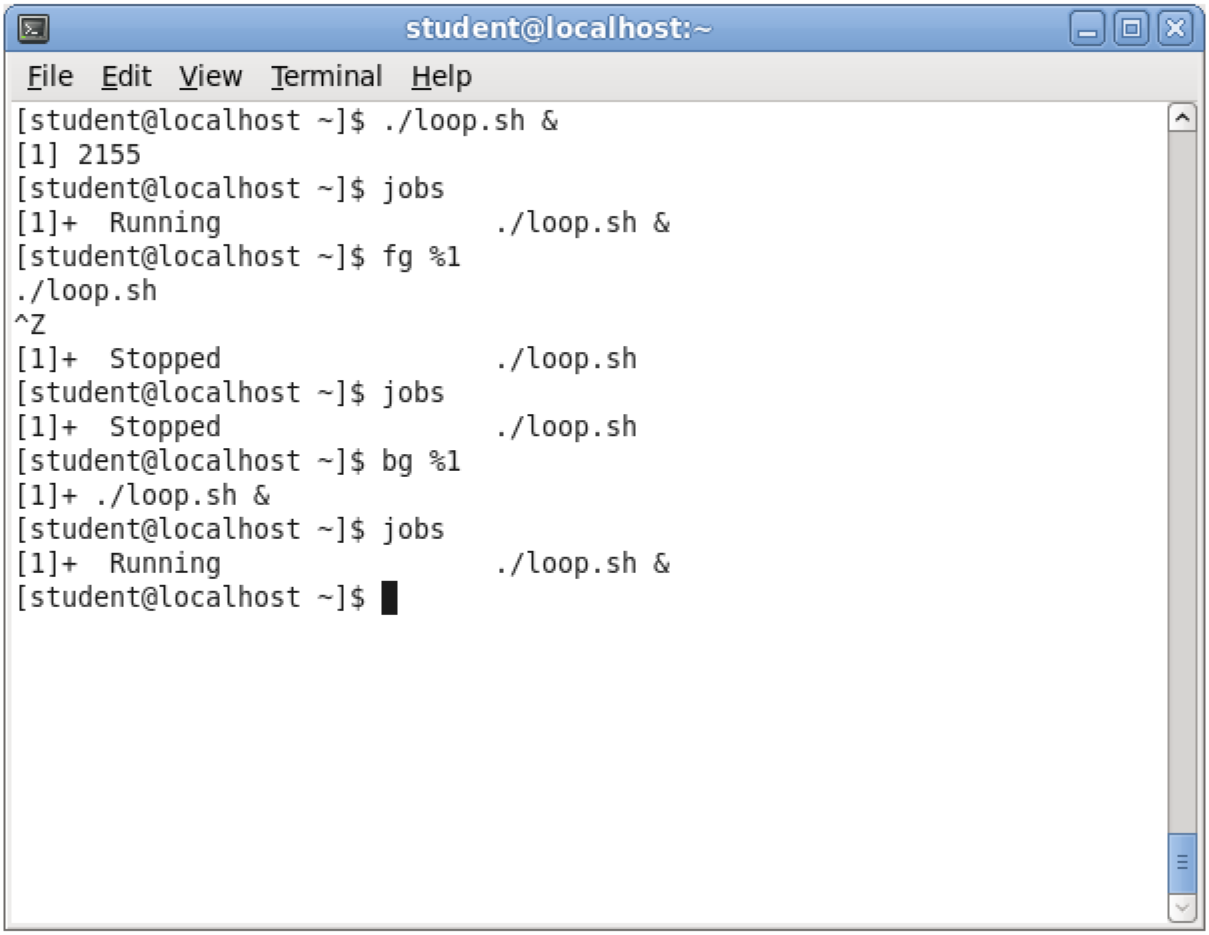


```
[student@localhost ~]$ cat>loop.sh
i=1
while [ 1 ]
do
i=$i+1
done
[student@localhost ~]$ chmod 765 loop.sh
[student@localhost ~]$ ls -l loop.sh
-rwxrw-r-x. 1 student student 33 2010-07-16 22:24 loop.sh
[student@localhost ~]$ 
```

You can start a job running in the background by adding the ampersand (&) to the end of any command.

Note

The CTRL-Z key cause the job to stop. You can restart the job by using the **bg[/bg] command.**

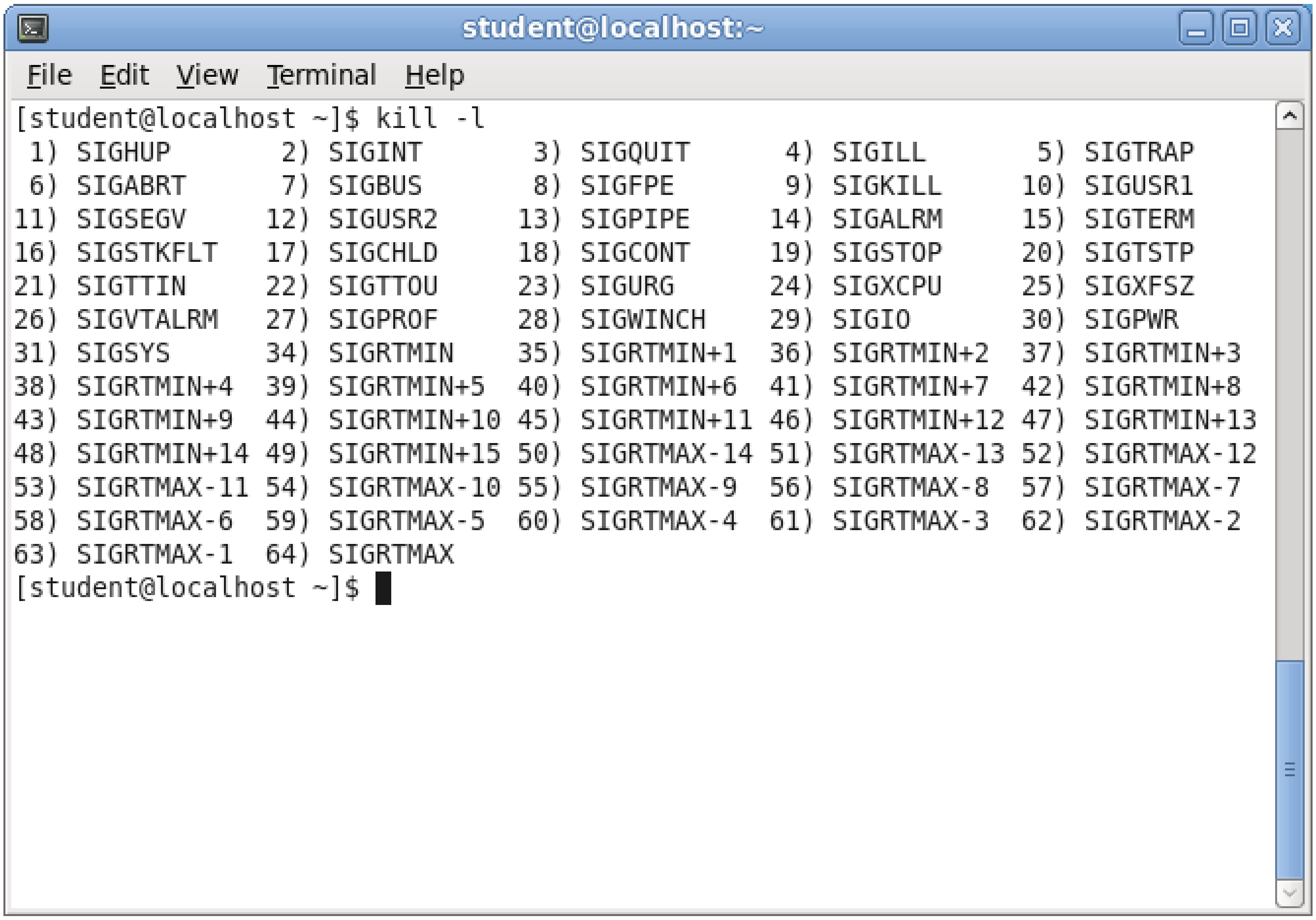


```
[student@localhost ~]$ ./loop.sh &
[1] 2155
[student@localhost ~]$ jobs
[1]+  Running                 ./loop.sh &
[student@localhost ~]$ fg %1
./loop.sh
^Z
[1]+  Stopped                 ./loop.sh
[student@localhost ~]$ jobs
[1]+  Stopped                 ./loop.sh
[student@localhost ~]$ bg %1
[1]+ ./loop.sh &
[student@localhost ~]$ jobs
[1]+  Running                 ./loop.sh &
[student@localhost ~]$ 
```

Exercise 4 - Signals

1. Signals are used to notify a process or thread of a particular event. Signals are software interrupts. When a signal is sent to a process or thread, it causes the processor to enter an "interrupt" handler, so subsequent processing can be done in the operating system based on the source and cause of the interrupt.

   Run "**kill** -l" command to list the signals to use with the kill command.



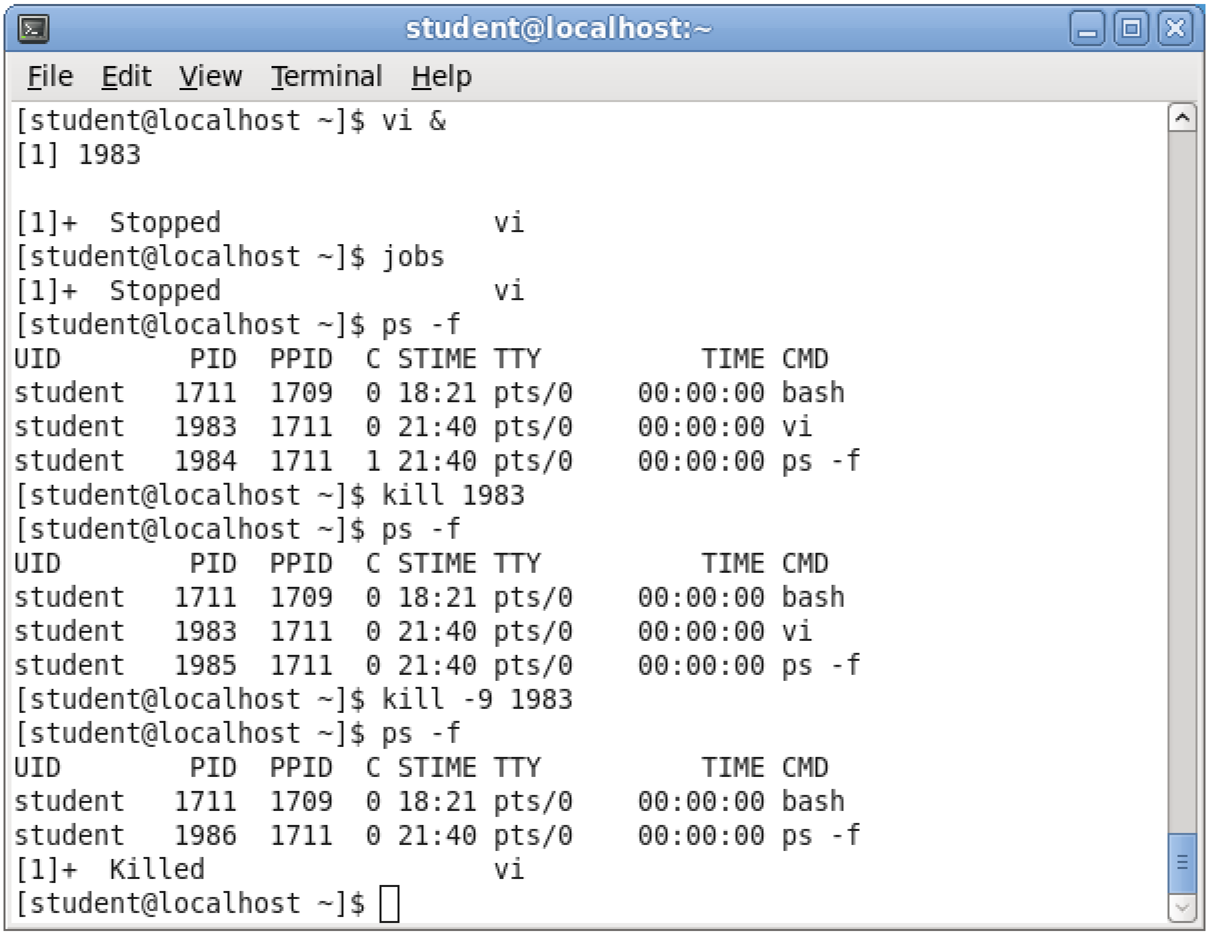
```
[student@localhost ~]$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
[student@localhost ~]$ 
```

   Every type of event is represented by a signal. Every signal has a unique signal name, and a corresponding signal number. The system defines a default action to take when a signal occurs.

   There are four types of default actions:

Action	Description
Exit	Forces the process to exit.
Core	Forces the process to exit and create a core file.
Stop	Stops the process.
Ignore	Ignores the signal and no action taken.

2.



```
[student@localhost ~]$ vi &
[1] 1983

[1]+  Stopped                 vi
[student@localhost ~]$ jobs
[1]+  Stopped                 vi
[student@localhost ~]$ ps -f
UID        PID  PPID  C STIME TTY          TIME CMD
student   1711  1709  0 18:21 pts/0    00:00:00 bash
student   1983  1711  0 21:40 pts/0    00:00:00 vi
student   1984  1711  1 21:40 pts/0    00:00:00 ps -f
[student@localhost ~]$ kill 1983
[student@localhost ~]$ ps -f
UID        PID  PPID  C STIME TTY          TIME CMD
student   1711  1709  0 18:21 pts/0    00:00:00 bash
student   1983  1711  0 21:40 pts/0    00:00:00 vi
student   1985  1711  0 21:40 pts/0    00:00:00 ps -f
[student@localhost ~]$ kill -9 1983
[student@localhost ~]$ ps -f
UID        PID  PPID  C STIME TTY          TIME CMD
student   1711  1709  0 18:21 pts/0    00:00:00 bash
student   1986  1711  0 21:40 pts/0    00:00:00 ps -f
[1]+  Killed                  vi
[student@localhost ~]$ 
```

Note

If the signal is not specified, by default, the **kill** command sends signal 15 to terminate a process.

Note

A signal value of 9 means that the signal cannot be caught by the application but is intercepted by operating system to terminate the process.