

David Nguyen

CMPE150

Prelab Questions:

1. [5] What command will show you which groups you are a member of?
The groups command will show which groups the user belongs to on Linux or unix-like operating systems
2. [5] What does the environmental variable "\$?" hold? (Hint: the command 'echo \$?' will should you this on your screen)
"echo \$?" Will return the number 0 meaning the variable \$? Holds the value 0
3. [5] What key combination will suspend a currently running process and place it as a background process?
The command ctrl-z will suspend a currently running process and bg will put the most recently suspended process as a background process
4. [5] With what command (and arguments) can you find out your kernel version and the "nodename"? [The output should not include any other information]
uname -n -r
5. [5] What is the difference between the paths ".", "..", and "~"? What does the path "/" refer to when not preceded by anything?
. represents the current directory
.. represents the parent directory
~ represents the home directory
/ refers to the root directory
6. [10] Which command would you use to find the ID (pid) for a running process? which command you will use to kill a running process using its ID?
pidof (running process)
for example: pidof Photos
kill (PID)
for example: kill 18723
7. [20] Write a single command that will return every user's default shell. [You may chain commands using piping and redirects] (Hint: See 'Chapter 19: filters' of linux-training.be as well as the man page for the /etc/passwd file: <https://linux.die.net/man/5/passwd>)
getent passwd | cut -d: -f1,7
<https://unix.stackexchange.com/questions/313928/return-every-user-s-default-shell>
8. [5] What is the difference between "sudo" and "su root"?
The sudo command allows the user to run a command as a root from the current user

The su root switchers to the user named root requiring the user to login in as root, this means the user will not need to enter sudo before a command to run as root

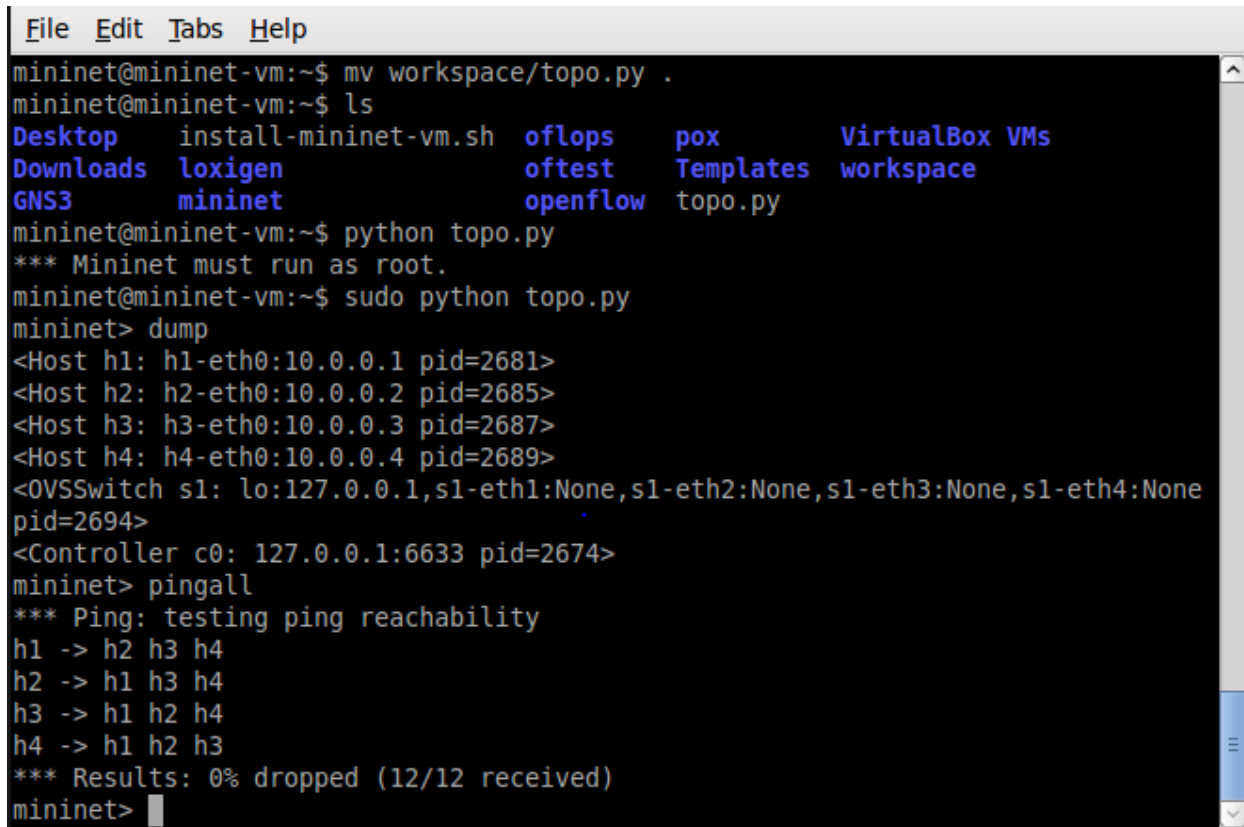
9. [10] How would you make a program or script execute on a schedule or set interval? E.g. Run this program once every 30 minutes or every day at midnight.

`watch -n 1800 script/command`

for a command to run on a schedule just set the watch command at the correct time and then put in the seconds that would run every day or for how often the script should be ran

for example: `watch -n 1800 script.sh`

2.[30 pts] Save a screenshot of dump and pingall output. Explain what is being shown in the screenshot.

A screenshot of a terminal window with a menu bar (File, Edit, Tabs, Help) and a title bar (mininet@mininet-vm). The terminal shows the following commands and output:

```
mininet@mininet-vm:~$ mv workspace/topo.py .
mininet@mininet-vm:~$ ls
Desktop      install-mininet-vm.sh  oflops      pox          VirtualBox  VMs
Downloads    loxigen                oftest      Templates    workspace
GNS3         mininet                openflow    topo.py

mininet@mininet-vm:~$ python topo.py
*** Mininet must run as root.
mininet@mininet-vm:~$ sudo python topo.py
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=2681>
<Host h2: h2-eth0:10.0.0.2 pid=2685>
<Host h3: h3-eth0:10.0.0.3 pid=2687>
<Host h4: h4-eth0:10.0.0.4 pid=2689>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None,s1-eth3:None,s1-eth4:None
pid=2694>
<Controller c0: 127.0.0.1:6633 pid=2674>
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
mininet>
```

Dump displays all information about nodes and switches such as IP address, process ID, and ports. For example H1 has port eth0, IP address 10.0.0.1, and process ID 2681

Pingall command does an all pair ping which checks for connections between all peers. For example h1 pings all other host, then the cycle repeats from h1-h4. The results were show how many and percentage of packets loss.

3. [10 pts] Run the iperf command as well, and screenshot the output, how fast is the connect?

```
File Edit Tabs Help
<Controller c0: 127.0.0.1:6633 pid=2674>
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h4
*** Results: ['13.6 Gbits/sec', '13.6 Gbits/sec']
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h4
*** Results: ['15.2 Gbits/sec', '15.2 Gbits/sec']
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h4
*** Results: ['17.9 Gbits/sec', '17.9 Gbits/sec']
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h4
*** Results: ['13.6 Gbits/sec', '13.6 Gbits/sec']
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h4
*** Results: ['13.6 Gbits/sec', '13.6 Gbits/sec']
mininet> 
```

The connect ranges from 13.6 Gb/s to 17.9Gb/s

4. Run wireshark, and using the display filter, filter for “of”. Note: When you run wireshark you should do so as “sudo wireshark”. When you choose an interface to capture on, you should select “any”.
- a. [20 pts] Run ping from a host to any other host using hX ping -c 5 hY. How many of_packet_in messages show up? Take a screenshot of your results

```

mininet@mininet-vm: ~
File Edit Tabs Help
[-w deadline] [-W timeout] [hop1 ...] destination
mininet> h1 ping -c 5 h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=1.56 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=1.20 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.098 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=0.032 ms
64 bytes from 10.0.0.3: icmp_seq=5 ttl=64 time=0.036 ms

--- 10.0.0.3 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4002ms
rtt min/avg/max/mdev = 0.032/0.585/1.562/0.660 ms
mininet> h2 ping -c 5 h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=1.82 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.366 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.027 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=0.037 ms
64 bytes from 10.0.0.3: icmp_seq=5 ttl=64 time=0.031 ms

--- 10.0.0.3 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4002ms
rtt min/avg/max/mdev = 0.027/0.456/1.821/0.694 ms
mininet>

```

2	0.000904000	127.0.0.1	127.0.0.1	OF 1.0	76 of_echo_reply
12	4.999986000	127.0.0.1	127.0.0.1	OF 1.0	76 of_echo_request
13	5.000728000	127.0.0.1	127.0.0.1	OF 1.0	76 of_echo_reply
16	7.438607000	10.0.0.1	10.0.0.3	OF 1.0	184 of_packet_in
17	7.439402000	127.0.0.1	127.0.0.1	OF 1.0	92 of_packet_out
24	7.439687000	10.0.0.3	10.0.0.1	OF 1.0	184 of_packet_in
25	7.440172000	127.0.0.1	127.0.0.1	OF 1.0	148 of_flow_add
29	8.439428000	10.0.0.1	10.0.0.3	OF 1.0	184 of_packet_in
30	8.439987000	127.0.0.1	127.0.0.1	OF 1.0	148 of_flow_add
50	12.451091000	96:a5:1e:64:7f:18	6e:7e:37:57:38:de	OF 1.0	128 of_packet_in
51	12.451607000	127.0.0.1	127.0.0.1	OF 1.0	148 of_flow_add
55	12.452223000	6e:7e:37:57:38:de	96:a5:1e:64:7f:18	OF 1.0	128 of_packet_in
56	12.452780000	127.0.0.1	127.0.0.1	OF 1.0	148 of_flow_add
59	16.999995000	127.0.0.1	127.0.0.1	OF 1.0	76 of_echo_request
60	17.000447000	127.0.0.1	127.0.0.1	OF 1.0	76 of_echo_reply

```

> Frame 1: 76 bytes on wire (608 bits), 76 bytes captured (608 bits) on interface 0
> Linux cooked capture
> Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
> Transmission Control Protocol, Src Port: 60647 (60647), Dst Port: 6633 (6633), Seq: 1, Ack: 1, Len: 8
> OpenFlow

```

There is a total of 5 of_packet_in messages that show up in wireshark

Frame	Source IP	Destination IP
16	10.0.0.1	10.0.0.3
24	10.0.0.3	10.0.0.1
29	10.0.0.1	10.0.0.3
50	96:a5:1e:64:7f:18	6e:73:37:57:38:de
55	6e:73:37:57:38:de	96:a5:1e:64:7f:18

[20 pts] What is the source and destination IP addresses for these entries? Find another packet that matches the “of” filter with the OpenFlow typefield set to OFPT_PACKET_OUT. What is the source and destination IP address for this entry? Take screenshots showing your results

487	369.44846200	10.0.0.3	10.0.0.4	OF 1.0	184 of_packet_in
488	369.44907900	127.0.0.1	127.0.0.1	OF 1.0	92 of_packet_out
495	369.44935100	10.0.0.4	10.0.0.3	OF 1.0	184 of_packet_in
496	369.44985800	127.0.0.1	127.0.0.1	OF 1.0	148 of_flow_add
500	370.46069400	10.0.0.3	10.0.0.4	OF 1.0	184 of_packet_in
501	370.46130900	127.0.0.1	127.0.0.1	OF 1.0	148 of_flow_add
519	374.45502600	b2:2d:cd:1a:9d:77	96:a5:1e:64:7f:18	OF 1.0	128 of_packet_in
520	374.45559500	127.0.0.1	127.0.0.1	OF 1.0	148 of_flow_add
524	374.45609600	96:a5:1e:64:7f:18	b2:2d:cd:1a:9d:77	OF 1.0	128 of_packet_in
525	374.45651400	127.0.0.1	127.0.0.1	OF 1.0	148 of_flow_add

penFlow

version: 1

type: OFPT_PACKET_OUT (13)

OFPT_PACKET_OUT

Frame	Source IP	Destination IP
17	127.0.0.1	127.0.0.1

[20 pts] Replace the display filter for “of” to “icmp && not of”. Run pingall again, how many entries are generated in wireshark? What types of icmp entries show up? Take a screenshot of your results.

Filter: icmp && not of Expression... Clear Apply Save						
No.	Time	Source	Destination	Protocol	Length	Info
1283	600.9655640	10.0.0.3	10.0.0.4	ICMP	100	Echo (ping) reply id=0x0f13, seq=1/256, ttl=64 (request in 1202)
1284	600.9657550	10.0.0.3	10.0.0.4	ICMP	100	Echo (ping) reply id=0x0f13, seq=1/256, ttl=64
1248	609.9794160	10.0.0.1	10.0.0.2	ICMP	100	Echo (ping) request id=0x0f16, seq=1/256, ttl=64
1249	609.9795050	10.0.0.1	10.0.0.2	ICMP	100	Echo (ping) request id=0x0f16, seq=1/256, ttl=64 (reply in 1250)
1250	609.9795150	10.0.0.2	10.0.0.1	ICMP	100	Echo (ping) reply id=0x0f16, seq=1/256, ttl=64 (request in 1249)
1251	609.9797660	10.0.0.2	10.0.0.1	ICMP	100	Echo (ping) reply id=0x0f16, seq=1/256, ttl=64
1252	609.9816670	10.0.0.1	10.0.0.3	ICMP	100	Echo (ping) request id=0x0f17, seq=1/256, ttl=64
1253	609.9817370	10.0.0.1	10.0.0.3	ICMP	100	Echo (ping) request id=0x0f17, seq=1/256, ttl=64 (reply in 1254)
1254	609.9817480	10.0.0.3	10.0.0.1	ICMP	100	Echo (ping) reply id=0x0f17, seq=1/256, ttl=64 (request in 1253)
1255	609.9819220	10.0.0.3	10.0.0.1	ICMP	100	Echo (ping) reply id=0x0f17, seq=1/256, ttl=64
1256	609.9838540	10.0.0.1	10.0.0.4	ICMP	100	Echo (ping) request id=0x0f18, seq=1/256, ttl=64
1257	609.9839260	10.0.0.1	10.0.0.4	ICMP	100	Echo (ping) request id=0x0f18, seq=1/256, ttl=64 (reply in 1258)
1258	609.9839370	10.0.0.4	10.0.0.1	ICMP	100	Echo (ping) reply id=0x0f18, seq=1/256, ttl=64 (request in 1257)
1259	609.9841130	10.0.0.4	10.0.0.1	ICMP	100	Echo (ping) reply id=0x0f18, seq=1/256, ttl=64
1260	609.9864960	10.0.0.2	10.0.0.1	ICMP	100	Echo (ping) request id=0x0f19, seq=1/256, ttl=64
1261	609.9865810	10.0.0.2	10.0.0.1	ICMP	100	Echo (ping) request id=0x0f19, seq=1/256, ttl=64 (reply in 1262)
1262	609.9865950	10.0.0.1	10.0.0.2	ICMP	100	Echo (ping) reply id=0x0f19, seq=1/256, ttl=64 (request in 1261)
1263	609.9867850	10.0.0.1	10.0.0.2	ICMP	100	Echo (ping) reply id=0x0f19, seq=1/256, ttl=64
1264	609.9901740	10.0.0.2	10.0.0.3	ICMP	100	Echo (ping) request id=0x0f1a, seq=1/256, ttl=64
1265	609.9903170	10.0.0.2	10.0.0.3	ICMP	100	Echo (ping) request id=0x0f1a, seq=1/256, ttl=64 (reply in 1266)
1266	609.9903330	10.0.0.3	10.0.0.2	ICMP	100	Echo (ping) reply id=0x0f1a, seq=1/256, ttl=64 (request in 1265)
1267	609.9905110	10.0.0.3	10.0.0.2	ICMP	100	Echo (ping) reply id=0x0f1a, seq=1/256, ttl=64
1268	609.9931960	10.0.0.2	10.0.0.4	ICMP	100	Echo (ping) request id=0x0f1b, seq=1/256, ttl=64
1269	609.9933170	10.0.0.2	10.0.0.4	ICMP	100	Echo (ping) request id=0x0f1b, seq=1/256, ttl=64 (reply in 1270)
1270	609.9933280	10.0.0.4	10.0.0.2	ICMP	100	Echo (ping) reply id=0x0f1b, seq=1/256, ttl=64 (request in 1269)
1271	609.9935100	10.0.0.4	10.0.0.2	ICMP	100	Echo (ping) reply id=0x0f1b, seq=1/256, ttl=64
1272	609.9961020	10.0.0.3	10.0.0.1	ICMP	100	Echo (ping) request id=0x0f1c, seq=1/256, ttl=64
1273	609.9962110	10.0.0.3	10.0.0.1	ICMP	100	Echo (ping) request id=0x0f1c, seq=1/256, ttl=64 (reply in 1274)
1274	609.9962200	10.0.0.1	10.0.0.3	ICMP	100	Echo (ping) reply id=0x0f1c, seq=1/256, ttl=64 (request in 1273)
1275	609.9964250	10.0.0.1	10.0.0.3	ICMP	100	Echo (ping) reply id=0x0f1c, seq=1/256, ttl=64
1276	609.9987430	10.0.0.3	10.0.0.2	ICMP	100	Echo (ping) request id=0x0f1d, seq=1/256, ttl=64
1277	609.9988800	10.0.0.3	10.0.0.2	ICMP	100	Echo (ping) request id=0x0f1d, seq=1/256, ttl=64 (reply in 1278)
1278	609.9988900	10.0.0.2	10.0.0.3	ICMP	100	Echo (ping) reply id=0x0f1d, seq=1/256, ttl=64 (request in 1277)
1280	609.9991680	10.0.0.2	10.0.0.3	ICMP	100	Echo (ping) reply id=0x0f1d, seq=1/256, ttl=64
1283	610.0027040	10.0.0.3	10.0.0.4	ICMP	100	Echo (ping) request id=0x0f1e, seq=1/256, ttl=64
1284	610.0028110	10.0.0.3	10.0.0.4	ICMP	100	Echo (ping) request id=0x0f1e, seq=1/256, ttl=64 (reply in 1285)
1285	610.0028970	10.0.0.4	10.0.0.3	ICMP	100	Echo (ping) reply id=0x0f1e, seq=1/256, ttl=64 (request in 1284)
1286	610.0032150	10.0.0.4	10.0.0.3	ICMP	100	Echo (ping) reply id=0x0f1e, seq=1/256, ttl=64
1287	610.0096550	10.0.0.4	10.0.0.1	ICMP	100	Echo (ping) request id=0x0f1f, seq=1/256, ttl=64
1288	610.0097390	10.0.0.4	10.0.0.1	ICMP	100	Echo (ping) request id=0x0f1f, seq=1/256, ttl=64 (reply in 1289)
1289	610.0097490	10.0.0.1	10.0.0.4	ICMP	100	Echo (ping) reply id=0x0f1f, seq=1/256, ttl=64 (request in 1288)
1290	610.0099440	10.0.0.1	10.0.0.4	ICMP	100	Echo (ping) reply id=0x0f1f, seq=1/256, ttl=64
1291	610.0131090	10.0.0.4	10.0.0.2	ICMP	100	Echo (ping) request id=0x0f20, seq=1/256, ttl=64
1292	610.0131950	10.0.0.4	10.0.0.2	ICMP	100	Echo (ping) request id=0x0f20, seq=1/256, ttl=64 (reply in 1293)
1293	610.0132080	10.0.0.2	10.0.0.4	ICMP	100	Echo (ping) reply id=0x0f20, seq=1/256, ttl=64 (request in 1292)
1294	610.0134190	10.0.0.2	10.0.0.4	ICMP	100	Echo (ping) reply id=0x0f20, seq=1/256, ttl=64
1295	610.0202800	10.0.0.4	10.0.0.3	ICMP	100	Echo (ping) request id=0x0f21, seq=1/256, ttl=64
1296	610.0204130	10.0.0.4	10.0.0.3	ICMP	100	Echo (ping) request id=0x0f21, seq=1/256, ttl=64 (reply in 1297)
1297	610.0204250	10.0.0.3	10.0.0.4	ICMP	100	Echo (ping) reply id=0x0f21, seq=1/256, ttl=64 (request in 1296)
1298	610.0206630	10.0.0.3	10.0.0.4	ICMP	100	Echo (ping) reply id=0x0f21, seq=1/256, ttl=64

49 entries are generated in wireshark when running pingall

There are echo reply (0) and echo requests (8) ICMP entries