

# Namespaces

Ameet Deshpande (CS15B001)

November 6, 2017

## 1 Namespaces

A namespace wraps a global system resource in an abstraction that makes it appear to the processes within the namespace that they have their own isolated instance of the global resource. *unshare* is a command that can be used to run a program with some namespaces unshared from the parent.

The *-m* flag is used to unshare the mountspace. The aim of this assignment is to create a different mount point in namespace. This mount point is not visible in the other session. The following is some example code.

---

```
1 # unshare --mount /bin/bash
2 # mount /dev/sda1 /mnt/test
3 # grep test /proc/mounts
```

---

This makes sure that there is a new mount point for root. When the following command is run in the same session, we get an output like *total 164*. When the code is run in a different terminal, we get an output like *total 0*. This shows that the mount is visible only in that session.

## 2 Types of Namespaces

### 2.1 Mount Namespaces

Mount namespaces provide isolation of the list of mount points seen by the processes in each namespace instance. Thus, the processes in each of the mount namespace instances will see distinct single- directory hierarchies.

### 2.2 Process ID

PID namespaces isolate the process ID number space, meaning that processes in different PID namespaces can have the same PID. PID namespaces allow containers to provide functionality. The different PID's will now be in different namespaces.

## 2.3 User Namespaces

User namespaces are a feature to provide both privilege isolation and user identification segregation across multiple sets of processes. The same process can have different privileges in different namespaces. User namespaces can be nested, that is, each user namespace other than "root" has a parent user namespace and can have zero or more child user namespaces. The *clone* or the *unshare* command can create a new user namespace. Unlike the *fork* command, namespaces can be such that the whole hierarchy tree is copied. There can thus be multiple roots

## 2.4 Network Namespaces

A network namespace allows each of these processes to see an entirely different set of networking interfaces. This is interesting because each network namespace will have a different iptable, routing table etc. The environment for the processes will be like there are multiple network interfaces.

## 3 Experiment

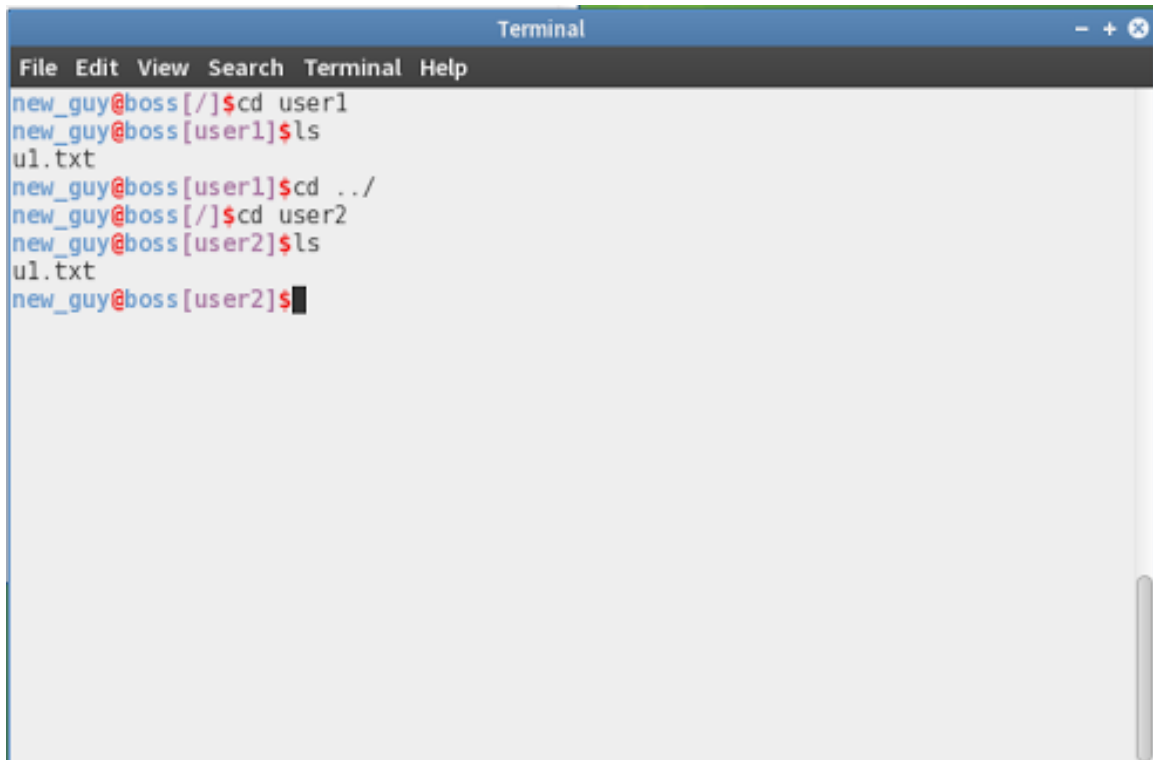
As discussed previously, the main commands that are to be used are *unshare*, *mount* and *umount*. The first command basically unshares the process from the parent and executes it in a new namespace. The next two commands are actually used for mounting the files to different points. This will be useful when we want to change the mount point for different users upon login. The following are the steps that are followed to check how we can mount at different points and yet have the same files beneath.

- We first create a new username and password. We obviously need *sudo* privileges for the same. This can be done by executing the command *sudo unshare -m login*. This will prompt the user to enter a new username and password.
- To check how the mount points work, we create 2 folders in the root directory. Lets call it *user1* and *user2*. We create two files in them, one in each. The first is called *u1.txt* and the second is called *u2.txt*.
- We then run the mount command. *sudo mount - -bind user1 user2*. This mounts changes the mount point for the newly created namespaces.
- We then open a new terminal. This is clearly not in the current namespace. Note that the namespace lasts for that particular session alone. We change the directory to *user2* in both cases and check which file is present inside.

## 4 Results

As expected, for the new user, because they are in a new namespace, they see the context of directory *user1*. But since this mount is visible only within the new

namespace, the new session for the terminal shows the normal output itself. This clearly shows the fact that the mount points created in user namespaces are not visible to the other namespaces.

A terminal window titled "Terminal" with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows a sequence of commands and their outputs. The prompt is "new\_guy@boss". The commands and outputs are: "cd user1", "ls" (output: "u1.txt"), "cd ../", "cd user2", "ls" (output: "u1.txt"), and finally "new\_guy@boss[user2]\$".

```
new_guy@boss[/]$cd user1
new_guy@boss[user1]$ls
u1.txt
new_guy@boss[user1]$cd ../
new_guy@boss[/]$cd user2
new_guy@boss[user2]$ls
u1.txt
new_guy@boss[user2]$
```

Figure 1: Terminal Output

As can be seen in this terminal. The same file is seen from both the folders. This is because the mount point has been changed.

The following figure shows the output of both the terminals.

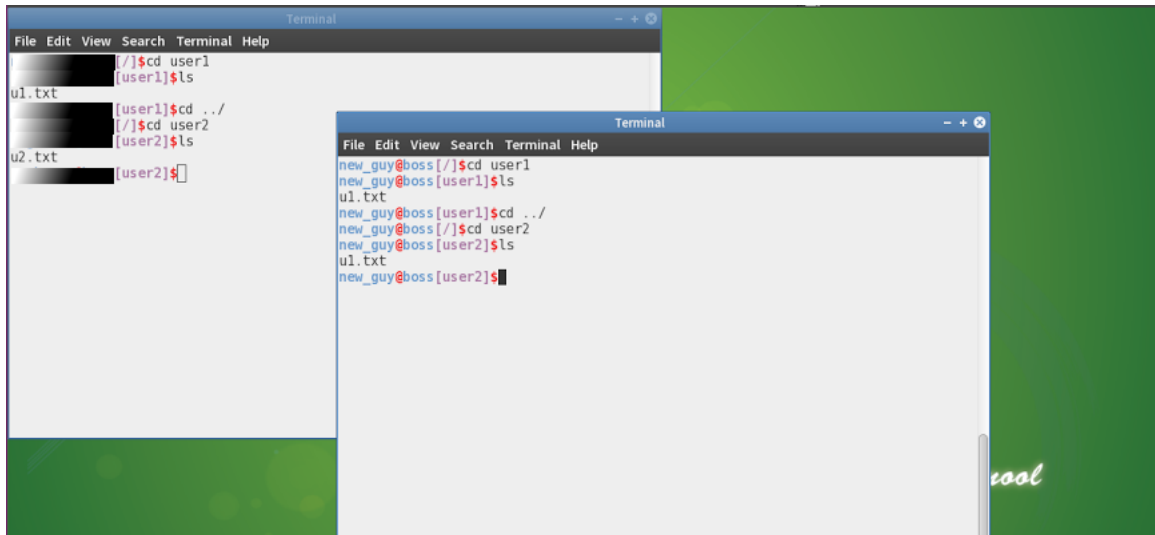


Figure 2: Terminal Output

As can be seen from the same, the other terminal that is not using the namespace shows the output correctly for both the files. This means that it is not able to see the mount point.