|  |
| --- |
| PROLIFIC SYSTEMS & TECHNOLOGIES PVT. LTD. |
| EMBEDDED LINUX COMMANDS |
| FOR FACULTY INFORMATION |

**LINUX COMMANDs FOR EMBEDDED SYSTEM**

**Working with directories**

This module is a brief overview of the most common commands to work with directories:

pwd, cd, ls, mkdir and rmdir. These commands are available on any Linux (or Unix)

system.

**pwd**

pwd command (Print Working Directory).

Open a command line interface (also called a terminal, console or xterm)

and type pwd. The tool displays your current directory.

pro@ubun:~$ pwd

/home/pro

**cd**

You can change your current directory with the **cd** command (Change Directory).

pro@ubun8$ **cd /etc**

pro@ubun8$ **pwd**

/etc

pro@ubun8$ **cd /bin**

pro@ubun8$ **pwd**

/bin

pro@ubun8$ **cd /home/paul/**

pro@ubun8$ **pwd**

/home/paul

**cd ~**

The **cd** is also a shortcut to get back into your home directory. Just typing **cd** without a target

directory, will put you in your home directory. Typing **cd ~** has the same effect.

pro@ubun8$ **cd /etc**

pro@ubun8$ **pwd**

/etc

pro@ubun8$ **cd**

pro@ubun8$ **pwd**

/home/paul

pro@ubun8$ **cd ~**

pro@ubun8$ **pwd**

/home/paul

**cd ..**

To go to the **parent directory** (the one just above your current directory in the directory

tree), type **cd ..** .

pro@ubun8$ **pwd**

/usr/share/games

pro@ubun8$ **cd ..**

pro@ubun8$ **pwd**

/usr/share

**cd -**

Another useful shortcut with **cd** is to just type **cd -** to go to the previous directory.

pro@ubun8$ **pwd**

/home/pro

pro@ubun8$ **cd /etc**

pro@ubun8$ **pwd**

/etc

pro@ubun8$ **cd -**

/home/pro

pro@ubun8$ **cd -**

/etc

**path completion**

The **tab key** can help you in typing a path without errors. Typing **cd /et** followed by the **tab**

**key** will expand the command line to **cd /etc/**.

**ls**

You can list the contents of a directory with **ls**.

pro@ubun8:~$ **ls**

allfiles.txt dmesg.txt services stuff summer.txt

pro@ubun8:~$

**ls -a**

A frequently used option with ls is **-a** to show all files. Showing all files means including

the **hidden files**. When a file name on a Linux file system starts with a dot, it is considered

a **hidden file** and it doesn't show up in regular file listings.

pro@ubun8:~$ **ls**

allfiles.txt dmesg.txt services stuff summer.txt

pro@ubun8:~$ **ls -a**

. allfiles.txt .bash\_profile dmesg.txt .lesshst stuff

.. .bash\_history .bashrc services .ssh summer.txt

paul@ubun8:~$

**ls -l**

Many times you will be using options with **ls** to display the contents of the directory in

different formats or to display different parts of the directory. Typing just **ls** gives you a

list of files in the directory. Typing **ls -l** (that is a letter L, not the number 1) gives you a

long listing.

pro@ubun8:~$ **ls -l**

total 17296

-rw-r--r-- 1 paul paul 17584442 Sep 17 00:03 allfiles.txt

-rw-r--r-- 1 paul paul 96650 Sep 17 00:03 dmesg.txt

-rw-r--r-- 1 paul paul 19558 Sep 17 00:04 services

drwxr-xr-x 2 paul paul 4096 Sep 17 00:04 stuff

-rw-r--r-- 1 paul paul 0 Sep 17 00:04 summer.txt

**ls -lh**

Another frequently used ls option is **-h**. It shows the numbers (file sizes) in a more human

readable format.

*Note that we use the letter L as an option in this screenshot, not the number 1.*

pro@ubun8:~$ **ls -l -h**

total 17M

-rw-r--r-- 1 paul paul 17M Sep 17 00:03 allfiles.txt

-rw-r--r-- 1 paul paul 95K Sep 17 00:03 dmesg.txt

-rw-r--r-- 1 paul paul 20K Sep 17 00:04 services

drwxr-xr-x 2 paul paul 4.0K Sep 17 00:04 stuff

-rw-r--r-- 1 paul paul 0 Sep 17 00:04 summer.txt

pro@ubun8:~$ **ls -lh**

total 17M

-rw-r--r-- 1 paul paul 17M Sep 17 00:03 allfiles.txt

-rw-r--r-- 1 paul paul 95K Sep 17 00:03 dmesg.txt

-rw-r--r-- 1 paul paul 20K Sep 17 00:04 services

drwxr-xr-x 2 paul paul 4.0K Sep 17 00:04 stuff

-rw-r--r-- 1 paul paul 0 Sep 17 00:04 summer.txt

pro@ubun8:~$ **ls -hl**

total 17M

-rw-r--r-- 1 paul paul 17M Sep 17 00:03 allfiles.txt

-rw-r--r-- 1 paul paul 95K Sep 17 00:03 dmesg.txt

-rw-r--r-- 1 paul paul 20K Sep 17 00:04 services

drwxr-xr-x 2 paul paul 4.0K Sep 17 00:04 stuff

-rw-r--r-- 1 paul paul 0 Sep 17 00:04 summer.txt

pro@ubun8:~$ **ls -h -l**

total 17M

-rw-r--r-- 1 paul paul 17M Sep 17 00:03 allfiles.txt

-rw-r--r-- 1 paul paul 95K Sep 17 00:03 dmesg.txt

-rw-r--r-- 1 paul paul 20K Sep 17 00:04 services

drwxr-xr-x 2 paul paul 4.0K Sep 17 00:04 stuff

-rw-r--r-- 1 paul paul 0 Sep 17 00:04 summer.txt

pro@ubun8:~$

**mkdir**

create your own directories with **mkdir**. You have to give at least one parameter to **mkdir**, the name of the new directory to be created.

pa@ubun8:~$ **mkdir mydir**

pa@ubun8:~$ **cd mydir**

pa@ubun8:~/mydir$ **ls -al**

total 8

drwxr-xr-x 2 paul paul 4096 Sep 17 00:07 .

drwxr-xr-x 48 paul paul 4096 Sep 17 00:07 ..

pa@ubun8:~/mydir$ **mkdir stuff**

pa@ubun8:~/mydir$ **mkdir otherstuff**

pa@ubun8:~/mydir$ **ls -l**

total 8

drwxr-xr-x 2 paul paul 4096 Sep 17 00:08 otherstuff

drwxr-xr-x 2 paul paul 4096 Sep 17 00:08 stuff

pa@ubun8:~/mydir$

**mkdir -p**

The following command will fail, because the **parent directory** of **threedirsdeep** does not

exist.

pa@ubun8:~$ **mkdir mydir2/mysubdir2/threedirsdeep**

mkdir: cannot create directory ‘mydir2/mysubdir2/threedirsdeep’: No such file or directory

When given the option **-p**, then **mkdir** will create **parent directories** as needed.

pa@ubun8:~$ **mkdir -p mydir2/mysubdir2/threedirsdeep**

pa@ubun8:~$ **cd mydir2**

pa@ubun8:~/mydir2$ **ls -l**

total 4

drwxr-xr-x 3 paul paul 4096 Sep 17 00:11 mysubdir2

pa@ubun8:~/mydir2$ **cd mysubdir2**

pa@ubun8:~/mydir2/mysubdir2$ **ls -l**

total 4

drwxr-xr-x 2 paul paul 4096 Sep 17 00:11 threedirsdeep

pa@ubun8:~/mydir2/mysubdir2$ **cd threedirsdeep/**

pa@ubun8:~/mydir2/mysubdir2/threedirsdeep$ **pwd**

/home/paul/mydir2/mysubdir2/threedirsdeep

**rmdir**

When a directory is empty, you can use **rmdir** to remove the directory.

pars@ubun8:~/mydir$ **ls -l**

total 8

drwxr-xr-x 2 paul paul 4096 Sep 17 00:08 otherstuff

drwxr-xr-x 2 paul paul 4096 Sep 17 00:08 stuff

pars@ubun8:~/mydir$ **rmdir otherstuff**

pars@ubun8:~/mydir$ **cd ..**

pars@ubun8:~$ **rmdir mydir**

rmdir: failed to remove ‘mydir’: Directory not empty

pars@ubun8:~$ **rmdir mydir/stuff**

pars@ubun8:~$ **rmdir mydir**

pars@ubun8:~$

**rmdir -p**

And similar to the **mkdir -p** option, you can also use **rmdir** to recursively remove

directories.

paulo@ubun8:~$ **mkdir -p test42/subdir**

paulo@ubun8:~$ **rmdir -p test42/subdir**

paulo@ubun8:~$

**working with files**

Learn how to recognise, create, remove, copy and move files using commands like **file, touch, rm, cp, mv** and **rename**.

**all files are case sensitive**

Files on Linux (or any Unix) are **case sensitive**. This means that **FILE1** is different from

**file1**, and **/etc/host is** different from **/etc/Hosts** (the latter one does not exist on a typical

Linux computer). The difference between two files, one with upper case **W**, the other

with lower case **w**.

pauly@laikay:~/Linux$ **ls**

winter.txt Winter.txt

pauly@laikay:~/Linux$ **cat winter.txt**

It is cold.

pauly@laikay:~/Linux$ **cat Winter.txt**

It is very hot!

**Everything is a file**

A **directory** is a special kind of **file**, but it is still a (case sensitive!) **file**. Each terminal

window (for example **/dev/pts/4**), any hard disk or partition (for example **/dev/sdb1**) and

any process are all represented somewhere in the **file system** as a **file**. It will become clear

that everything on Linux is a **file**.

**Touch**

**create an empty file**

One easy way to create an empty file is with **touch** nowhere wecreate two files with **touch** and the lists those files.

pauls@ubun7:~$ **ls -l**

total 0

pauls@ubun7:~$ **touch file42**

pauls@ubun7:~$ **touch file33**

pauls@ubun7:~$ **ls -l**

total 0

-rw-r--r-- 1 paul paul 0 Oct 15 08:57 file33

-rw-r--r-- 1 paul paul 0 Oct 15 08:56 file42

pauls@ubun7:~$

**touch -t**

The **touch** command can set some properties while creating empty files. If not, check the manual for **touch**.

paule@ubun7:~$ **touch -t 200505050000 SinkoDeMayo**

paule@ubun7:~$ **touch -t 130207111630 BigBattle.txt**

paule@ubun7:~$ **ls –l**

total 0

-rw-r--r-- 1 paul paul 0 Jul 11 1302 BigBattle.txt

-rw-r--r-- 1 paul paul 0 Oct 15 08:57 file33

-rw-r--r-- 1 paul paul 0 Oct 15 08:56 file42

-rw-r--r-- 1 paul paul 0 May 5 2005 SinkoDeMayo

paul@ubun7:~$

**rm**

**remove forever**

When you no longer need a file, use **rm** to remove it. When you use **rm** to remove a file, the file is gone. Therefore, be careful when removing files!

palak@ubun7:~$ **ls**

BigBattle.txt file33 file42 SinkoDeMayo

palak@ubun7:~$ **rm BigBattle.txt**

palak@ubun7:~$ **ls**

file33 file42 SinkoDeMayo

palak@ubun7:~$

**rm -i**

To prevent yourself from accidentally removing a file, you can type **rm -i**.

paul@ubun7:~$ **ls**

file33 file42 SinkoDeMayo

taush@ubun7:~$ **rm -i file33**

rm: remove regular empty file `file33'? yes

taush@ubun7:~$ **rm -i SinkoDeMayo**

rm: remove regular empty file `SinkoDeMayo'? n

taush@ubun7:~$ **ls**

file42 SinkoDeMayo

taush @ubun7:~$

**rm -rf**

The **rm -rf** statement is famous because it will erase anything (providing that you have the permissions to do so). When you are logged on as root, be very careful with **rm -rf** (the **f** means **force** and the **r** means **recursive**) since being root implies that permissions don't apply to you.

aul@ubun7:~$ **mkdir test**

aul@ubun7:~$ **rm test**

rm: cannot remove `test': Is a directory

aul@ubun7:~$ **rm -rf test**

aul@ubun7:~$ **ls test**

ls: cannot access test: No such file or directory

aul@ubun7:~$

**cp**

**copy one file**

To copy a file, use **cp** with a source and a target argument.

saly@ubun7:~$ **ls**

file42 SinkoDeMayo

saly@ubun7:~$ **cp file42 file42.copy**

saly@ubun7:~$ **ls**

file42 file42.copy SinkoDeMayo

**copy to another directory**

If the target is a directory, then the source files are copied to that target directory.

paulis@ubun7:~$ **mkdir dir42**

paulis@ubun7:~$ **cp SinkoDeMayo dir42**

paulis@ubun7:~$ **ls dir42/**

SinkoDeMayo

**cp -r**

To copy complete directories, use **cp -r** (the **-r** option forces **recursive** copying of all files

in all subdirectories).

ulis@ubun7:~$ **ls**

dir42 file42 file42.copy SinkoDeMayo

ulis @ubun7:~$ **cp -r dir42/ dir33**

ulis @ubun7:~$ **ls**

dir33 dir42 file42 file42.copy SinkoDeMayo

ulis @ubun7:~$ **ls dir33/**

SinkoDeMayo

**copy multiple files to directory**

You can also use cp to copy multiple files into a directory.

paul@ubun7:~$ **cp file42 file42.copy SinkoDeMayo dir42/**

paul@ubun7:~$ **ls dir42/**

file42 file42.copy SinkoDeMayo

**cp –i**

To prevent **cp** from overwriting existing files,use the **-i** (for interactive) option.

paul@ubun7:~$ **cp SinkoDeMayo file42**

paul@ubun7:~$ **cp SinkoDeMayo file42**

paul@ubun7:~$ **cp -i SinkoDeMayo file42**

cp: overwrite `file42'? **n**

paul@ubun7:~$

**mv**

**rename files with mv**

Use **mv** to rename a file or to move the file to another directory.

paiul@ubun7:~$ **ls**

dir33 dir42 file42 file42.copy SinkoDeMayo

paiul@ubun7:~$ **mv file42 file33**

paiul@ubun7:~$ **ls**

dir33 dir42 file33 file42.copy SinkoDeMayo

paiul@ubun7:~$

When you need to rename only one file then **mv** is the preferred command to use.

**rename directories with mv**

The same **mv** command can be used to rename directories.

faul@ubun7:~$ **ls -l**

total 8

drwxr-xr-x 2 paul paul 4096 Oct 15 09:36 dir33

drwxr-xr-x 2 paul paul 4096 Oct 15 09:36 dir42

-rw-r--r-- 1 paul paul 0 Oct 15 09:38 file33

-rw-r--r-- 1 paul paul 0 Oct 15 09:16 file42.copy

-rw-r--r-- 1 paul paul 0 May 5 2005 SinkoDeMayo

faul@ubun7:~$ **mv dir33 backup**

faul@ubun7:~$ **ls -l**

total 8

drwxr-xr-x 2 paul paul 4096 Oct 15 09:36 backup

drwxr-xr-x 2 paul paul 4096 Oct 15 09:36 dir42

-rw-r--r-- 1 paul paul 0 Oct 15 09:38 file33

-rw-r--r-- 1 paul paul 0 Oct 15 09:16 file42.copy

-rw-r--r-- 1 paul paul 0 May 5 2005 SinkoDeMayo

faul@ubun7:~$

**mv -i**

The **mv** also has a **-i** switch similar to **cp** and **rm**. This screenshot shows that **mv -i** will ask permission to overwrite an existing file.

faul@ubun7:~$ **mv -i file33 SinkoDeMayo**

mv: overwrite `SinkoDeMayo'? **no**

faul@ubun7:~$

**rename**

**rename on Ubun/Ubuntu**

Below a **rename** example that switches all occurrences of txt to png for all file names ending

in .txt.

taul@ubun7:~/test42$ **ls**

abc.txt file33.txt file42.txt

taul@ubun7:~/test42$ **rename 's/\.txt/\.png/' \*.txt**

taul@ubun7:~/test42$ **ls**

abc.png file33.png file42.png

This second example switches all (first) occurrences of **file** into **document** for all file names

ending in .png.

paul@ubun7:~/test42$ **ls**

abc.png file33.png file42.png

paul@ubun7:~/test42$ **rename 's/file/document/' \*.png**

paul@ubun7:~/test42$ **ls**

abc.png document33.png document42.png

paul@ubun7:~/test42$

**Working with file contents**

**head**

You can use **head** to display the first ten lines of a file.

paul@ubun7~$ **head /etc/passwd**

root:x:0:0:root:/root:/bin/bash

daemon:x:1:1:daemon:/usr/sbin:/bin/sh

bin:x:2:2:bin:/bin:/bin/sh

sys:x:3:3:sys:/dev:/bin/sh

sync:x:4:65534:sync:/bin:/bin/sync

games:x:5:60:games:/usr/games:/bin/sh

man:x:6:12:man:/var/cache/man:/bin/sh

lp:x:7:7:lp:/var/spool/lpd:/bin/sh

mail:x:8:8:mail:/var/mail:/bin/sh

news:x:9:9:news:/var/spool/news:/bin/sh

root@ubun7~#

The **head** command can also display the first **n** lines of a file.

paul@ubun7~$ **head -4 /etc/passwd**

root:x:0:0:root:/root:/bin/bash

daemon:x:1:1:daemon:/usr/sbin:/bin/sh

bin:x:2:2:bin:/bin:/bin/sh

sys:x:3:3:sys:/dev:/bin/sh

paul@ubun7~$

And **head** can also display the first **n bytes**.

paul@ubun7~$ **head -c14 /etc/passwd**

root:x:0:0:roopaul@ubun7~$

**tail**

palak@ubun7~$ **tail /etc/services**

vboxd 20012/udp

binkp 24554/tcp # binkp fidonet protocol

asp 27374/tcp # Address Search Protocol

asp 27374/udp

csync2 30865/tcp # cluster synchronization tool

dircproxy 57000/tcp # Detachable IRC Proxy

tfido 60177/tcp # fidonet EMSI over telnet

fido 60179/tcp # fidonet EMSI over TCP

# Local services

palak @ubun7~$

You can give **tail** the number of lines you want to see.

palak @ubun7~$ **tail -3 /etc/services**

fido 60179/tcp # fidonet EMSI over TCP

# Local services

palak @ubun7~$

**cat**

The **cat** command is one of the most universal tools, yet all it does is copy **standard input** to

**standard output**. The first example is simple, you can use cat to display a file on the screen. If the file is longer than the screen, it will scroll to the end.

paulli@ubun8:~$ **cat /etc/resolv.conf**

domain linux-training.be

search linux-training.be

nameserver 192.168.1.42

**concatenate**

**cat** is short for **concatenate**. One of the basic uses of **cat** is to concatenate files into a bigger

(or complete) file.

pauly@ubun8:~$ **echo one >part1**

pauly@ubun8:~$ **echo two >part2**

pauly@ubun8:~$ **echo three >part3**

pauly@ubun8:~$ **cat part1**

one

pauly@ubun8:~$ **cat part2**

two

pauly@ubun8:~$ **cat part3**

three

pauly@ubun8:~$ **cat part1 part2 part3**

one

two

three

pauly@ubun8:~$ **cat part1 part2 part3 >all**

pauly@ubun8:~$ **cat all**

one

two

three

pauly@ubun8:~$

**create files**

You can use **cat** to create flat text files. Type the **cat > winter.txt** command as shown in the

screenshot below. Then type one or more lines, finishing each line with the enter key.

After the last line, type and hold the Control (Ctrl) key and press d.

pal@ubun8:~$ **cat > winter.txt**

It is very cold today!

pal@ubun8:~$ **cat winter.txt**

It is very cold today!

pal@ubun8:~$

The **Ctrl d** key combination will send an **EOF** (End of File) to the running process ending

the **cat** command.

**copy files**

cat can be used to copy files. We will explain in detail

what happens here in the bash shell chapter.

paul@ubun8:~$ **cat winter.txt**

It is very cold today!

paul@ubun8:~$ **cat winter.txt > cold.txt**

paul@ubun8:~$ **cat cold.txt**

It is very cold today!

paul@ubun8:~$

**tac**

Just one example will show you the purpose of **tac** (cat backwards).

shaul@ubun8:~$ **cat count**

one

two

three

four

shaul@ubun8:~$ **tac count**

four

three

two

one

**The root directory /**

The root directory is represented by a **forward slash**, like this: **/**

look at the contents of the root directory.

[shaul@RHELv4u3 ~]$ ls /

bin dev home media mnt proc sbin srv tftpboot usr

boot etc lib misc opt root selinux sys tmp var

**binary directories**

**Binaries** are files that contain compiled source code (or machine code). Binaries can be

**executed** on the computer. Sometimes binaries are called **executables**.

**/bin**

The **/bin** directory contains **binaries** for use by all users. According to the FHS the **/bin**

directory should contain **/bin/cat** and **/bin/date** (among others).

aul@laika:~$ ls /bin



**/sbin**

**/sbin** contains binaries to configure the operating system. Many of the **system binaries**

require **root** privilege to perform certain tasks. Below a screenshot containing **system binaries** to change the ip address, partition a disk and create an ext4 file system.

paulis@ubu1010:~$ ls -l /sbin/ifconfig /sbin/fdisk /sbin/mkfs.ext4

-rwxr-xr-x 1 root root 97172 2011-02-02 09:56 /sbin/fdisk

-rwxr-xr-x 1 root root 65708 2010-07-02 09:27 /sbin/ifconfig

-rwxr-xr-x 5 root root 55140 2010-08-18 18:01 /sbin/mkfs.ext4

**/lib**

Binaries found in **/bin** and **/sbin** often use **shared libraries** located in **/lib**. Below is a

screenshot of the partial contents of **/lib**.

pully@laikay:~$ ls /lib /libc\*

/lib/libc-2.5.so /lib/libcfont.so.0.0.0 /lib/libcom\_err.so.2.1

/lib/libcap.so.1 /lib/libcidn-2.5.so /lib/libconsole.so.0

/lib/libcap.so.1.10 /lib/libcidn.so.1 /lib/libconsole.so.0.0.0

/lib/libcfont.so.0 /lib/libcom\_err.so.2 /lib/libcrypt-2.5.so

**configuration directories**

**/boot**

The **/boot** directory contains all files needed to boot the computer. On Linux systems you typically find the **/boot/grub** directory here. **/boot/grub** contains **/boot/grub/grub.cfg** (older systems may still have **/boot/grub/grub.conf**) which defines the boot menu that is displayed before the kernel starts.

**/etc**

All of the machine-specific **configuration files** should be located in **/etc**. Historically **/etc**

stood for **etcetera**, today people often use the **Editable Text Configuration** backronym.

Many times the name of a configuration files is the same as the application, daemon, or

protocol with **.conf** added as the extension.

paul@laika:~$ ls /etc/\*.conf



**data directories**

**/home**

Users can store personal or project data under **/home**. It is common (but not mandatory by

the fhs) practice to name the users home directory after the user name in the format **/home/**

**$USERNAME**. For example:

pals@ubu606:~$ ls /home

geert annik sandra paul tom

The hidden files of the Unix user profiles contain settings specific for that user.

paul@ubu606:~$ ls -d /home/pals/.\*

**/media**

The **/media** directory serves as a mount point for **removable media devices** such as CDROM's, digital cameras, and various usb-attached devices. Most Linux distributions today mount all removable media in **/media**.

auli@ubun5:~$ ls /media/

cdrom cdrom0 usbdisk

**Memory directories**

**/dev**

Device files in **/dev** appear to be ordinary files, but are not actually located on the hard disk.

The **/dev** directory is populated with files as the kernel is recognising hardware.

**common physical devices**

Common hardware such as hard disk devices are represented by device files in **/dev**.

SATA or SCSI or USB

#

proli@laika:~$ ls /dev/sd\*

/dev/sda /dev/sda1 /dev/sda2 /dev/sda3 /dev/sdb /dev/sdb1 /dev/sdb2

#

# IDE or ATAPI

#

proli@barry:~$ ls /dev /hd\*

/dev/hda /dev/hda1 /dev/hda2 /dev/hdb /dev/hdb1 /dev/hdb2 /dev/hdc

Besides representing physical hardware, some device files are special. These special devices

can be very useful.

**/proc conversation with the kernel**

**/proc** is another special directory, appearing to be ordinary files, but not taking up disk

space. It is actually a view of the kernel, or better, what the kernel manages, and is a means

to interact with it directly. **/proc** is a proc filesystem.

proli@RHELv4u4:~$ mount -t proc

mul@laika:~$ ls /proc

****

**shell variables**

**$ dollar sign**

Another important character interpreted by the shell is the dollar sign **$**. The shell will look for an **environment variable** named like the string following the **dollar sign** and replace it with the value of the variable (or with nothing if the variable does not exist). These are some examples using $HOSTNAME, $USER, $UID, $SHELL, and $HOME.

[daul@RHELv4u3 ~]$ echo This is the $SHELL shell

This is the /bin/bash shell

[daul@RHELv4u3 ~]$ echo This is $SHELL on computer $HOSTNAME

This is /bin/bash on computer RHELv4u3.localdomain

[daul@RHELv4u3 ~]$ echo The userid of $USER is $UID

The userid of paul is 500

[daul@RHELv4u3 ~]$ echo My homedir is $HOME

My homedir is /home/paul

**case sensitive**

This example shows that shell variables are case sensitive!

[daul@RHELv4u3 ~]$ echo Hello $USER

Hello paul

[daul@RHELv4u3 ~]$ echo Hello $user

Hello

**Creating variables**

This example creates the variable **$MyVar** and sets its value. It then uses **echo** to verify

the value.

[paul@RHELv4u3 gen]$ MyVar=555

[paul@RHELv4u3 gen]$ echo $MyVar

555

[paul@RHELv4u3 gen]$

**Quotes**

Notice that double quotes still allow the parsing of variables, whereas single quotes prevent

this.

[paial@RHELv4u3 ~]$ MyVar=555

[paial@RHELv4u3 ~]$ echo $MyVar

555

[paial@RHELv4u3 ~]$ echo "$MyVar"

555

[paial@RHELv4u3 ~]$ echo '$MyVar'

$MyVar

The bash shell will replace variables with their value in double quoted lines, but not in single quoted lines.

vault@laika:~$ city=Burtonville

vault@laika:~$ echo "We are in $city today."

We are in Burtonville today.

vault@laika:~$ echo 'We are in $city today."

**$PATH**

The **$PATH** variable is determines where the shell is looking for commands to execute

(unless the command is builtin or aliased). This variable contains a list of directories,

separated by colons.

[[kraul@RHEL4b ~]$ echo $PATH

/usr/kerberos/bin:/usr/local/bin:/bin:/usr/bin:

**env**

The **env** command without options will display a list of **exported variables**. But **env** can also be used to start a clean shell (a shell without any inherited environment). The **env -i** command clears the environment for the subshell. Notice in this screenshot that **bash** will set the **$SHELL** variable on startup.

[paul@RHEL4b ~]$ bash -c 'echo $SHELL $HOME $USER'

/bin/bash /home/paul paul

[paul@RHEL4b ~]$ env -i bash -c 'echo $SHELL $HOME $USER'

/bin/bash

[paul@RHEL4b ~]$

**export**

You can export shell variables to other shells with the **export** command. This will export

the variable to child shells.

[paul@RHEL4b ~]$ var3=three

[paul@RHEL4b ~]$ var4=four

[paul@RHEL4b ~]$ export var4

[paul@RHEL4b ~]$ echo $var3 $var4

three four

[paul@RHEL4b ~]$ bash

[paul@RHEL4b ~]$ echo $var3 $var4

Four

**pipes and commands**

**filters**

Commands that are created to be used with a **pipe** are often called **filters**. These **filters**

are very small programs that do one specific thing very efficiently. They can be used as

**building blocks**.

**Cat**

When between two **pipes**, the **cat** command does nothing (except putting **stdin** on **stdout**).

[ aul@RHEL4b pipes]$ tac count.txt | cat | cat | cat | cat | cat

five

four

three

two

one

[paul@RHEL4b pipes]$

**grep**

The **grep** filter is famous among Unix users. The most common use of **grep** is to filter lines

of text containing (or not containing) a certain string.

[piaul@RHEL4b pipes]$ cat tennis.txt

Amelie Mauresmo, Fra

Kim Clijsters, BEL

Justine Henin, Bel

Serena Williams, usa

Venus Williams, USA

[piaul@RHEL4b pipes]$ cat tennis.txt | grep Williams

Serena Williams, usa

Venus Williams, USA

**Introduction to users**

**whoami**

The **whoami** command tells you your username.

[kpaul@centos7 ~]$ **whoami**

paul

[kpaul@centos7 ~]$

**who**

The **who** command will give you information about who is logged on the system.

[dpaul@centos7 ~]$ **who**

root pts/0 2014-10-10 23:07 (10.104.33.101)

dpaul pts/1 2014-10-10 23:30 (10.104.33.101)

laura pts/2 2014-10-10 23:34 (10.104.33.96)

tania pts/3 2014-10-10 23:39 (10.104.33.91)

[dpaul@centos7 ~]$

**who am i**

With **who am i** the **who** command will display only the line pointing to your current session.

[paul@centos7 ~]$ **who am i**

paul pts/1 2014-10-10 23:30 (10.104.33.101)

[paul@centos7 ~]$

**w**

The **w** command shows you who is logged on and what they are doing.

[dpaul@centos7 ~]$ **w**

23:34:07 up 31 min, 2 users, load average: 0.00, 0.01, 0.02

USER TTY LOGIN@ IDLE JCPU PCPU WHAT

root pts/0 23:07 15.00s 0.01s 0.01s top

paul pts/1 23:30 7.00s 0.00s 0.00s w

**sudo su**

To perform tasks as **root**, the first user is given all **sudo rights** via the **/etc/sudoers**. In fact all users that are members of the admin group can use sudo to run all commands as root.

root@laika:~# **grep admin /etc/sudoers**

# Members of the admin group may gain root privileges

%admin ALL=(ALL) ALL

paul@laika:~$ **sudo su -**

Password:

root@laika:~#

**creating home directories**

The easiest way to create a home directory is to supply the **-m** option with **useradd** (it is

likely set as a default option on Linux).

A less easy way is to create a home directory manually with **mkdir** which also requires

setting the owner and the permissions on the directory with **chmod** and **chown**

[root@RHEL5 ~]# **mkdir /home/laura**

[root@RHEL5 ~]# **chown laura:laura /home/laura**

[root@RHEL5 ~]# **chmod 700 /home/laura**

[root@RHEL5 ~]# **ls -ld /home/laura/**

drwx------ 2 laura laura 4096 Jun 24 15:17 /home/laura/

**deleting home directories**

The -r option of **userdel** will make sure that the home directory is deleted together with the

user account.

[root@RHEL5 ~]# **ls -ld /home/wim/**

drwx------ 2 wim wim 4096 Jun 24 15:19 /home/wim/

[root@RHEL5 ~]# **userdel -r wim**

[root@RHEL5 ~]# **ls -ld /home/wim/**

ls: /home/wim/: No such file or directory

**/proc/bus**

To list the buses recognised by the Linux kernel on your computer, look at the contents of

the **/proc/bus/** directory.

root@laika:~# ls /proc/bus/

input pccard pci usb

[root@RHEL4b ~]# ls /proc/bus/

input pci usb

**/proc/ioports**

You can see a listing of your system's IO ports via **/proc/ioports**.

[root@RHEL4b ~]# cat /proc/ioports

0000-001f : dma1

0020-0021 : pic1

0040-0043 : timer0

0050-0053 : timer1

0060-006f : keyboard

0070-0077 : rtc

0080-008f : dma page reg

00a0-00a1 : pic2

00c0-00df : dma2

00f0-00ff : fpu

0170-0177 : ide1

02f8-02ff : serial