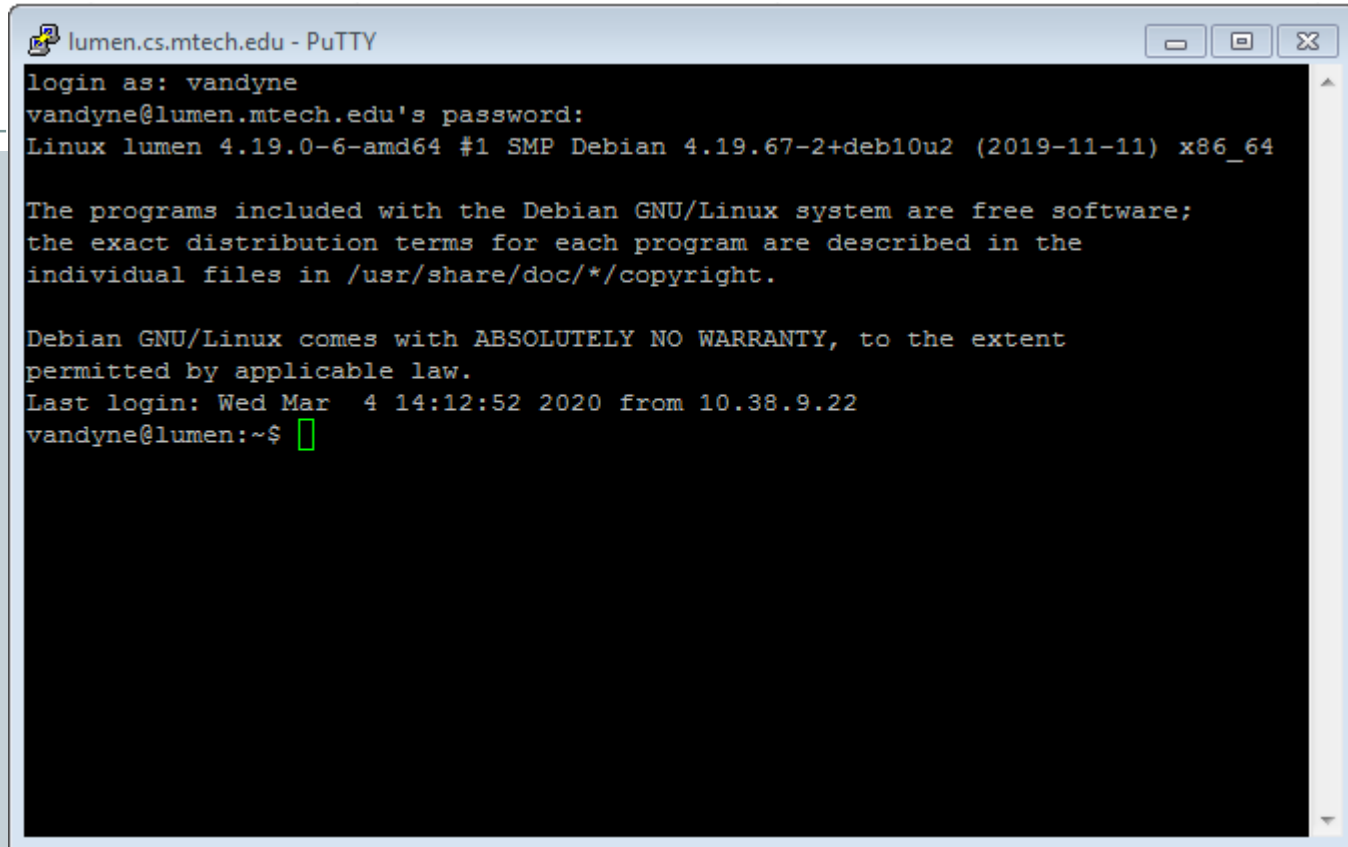


More Linux – Piping and Redirection



The image shows a PuTTY terminal window titled "lumen.cs.mtech.edu - PuTTY". The terminal output is as follows:

```
login as: vandyne
vandyne@lumen.mtech.edu's password:
Linux lumen 4.19.0-6-amd64 #1 SMP Debian 4.19.67-2+deb10u2 (2019-11-11) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

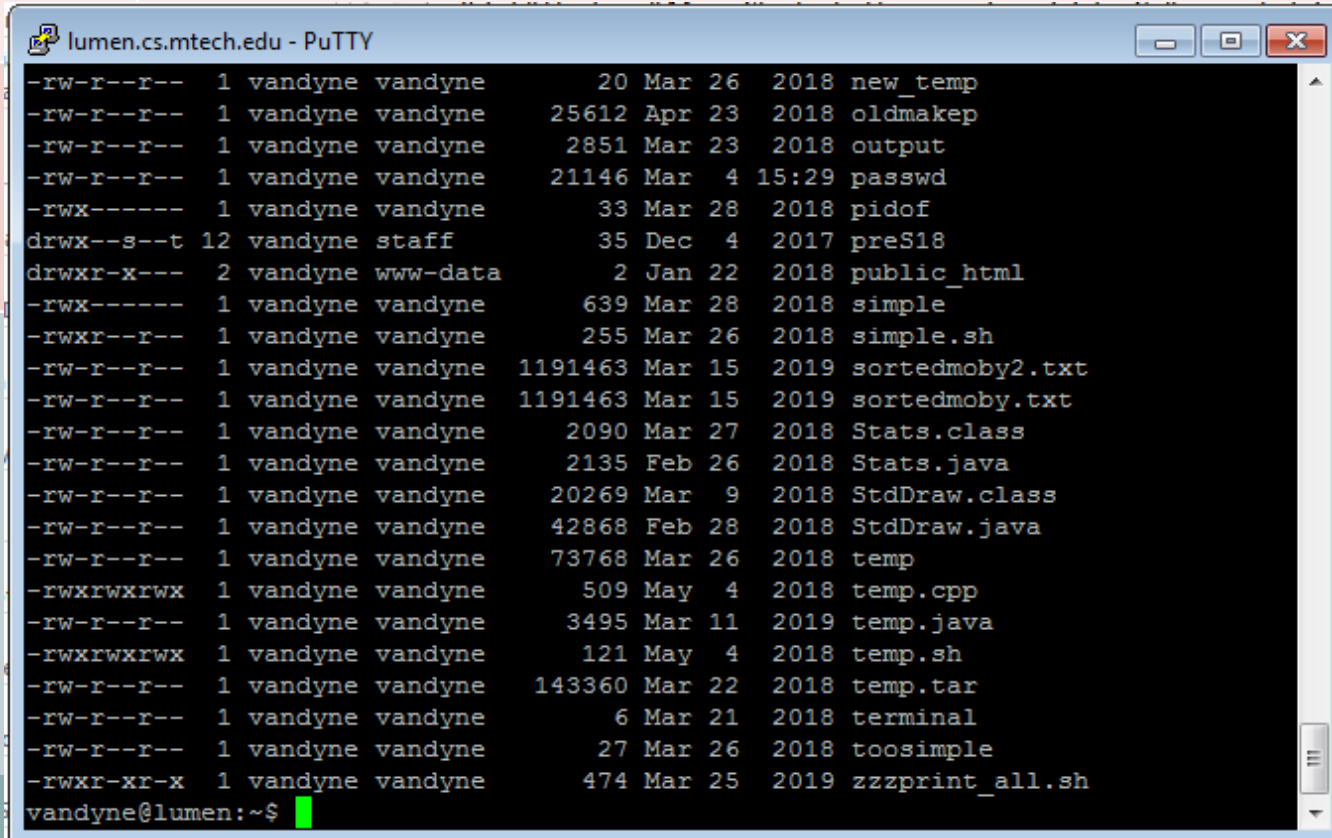
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Wed Mar  4 14:12:52 2020 from 10.38.9.22
vandyne@lumen:~$
```

Outline

- File and Directory Permissions
- File Content
- Finding Files
- Sorting Files
- File Compression
- Processes
- Pipes
- Input/Output Redirection
- Controlling Processes

File and Directory Permissions

Permission	File	Directory
read	User can look at the contents of the file	User can list the files in the directory
write	User can modify the	User can create new files and remove
execute		directory, but have read if they



```
lumen.cs.mtech.edu - PuTTY
-rw-r--r-- 1 vandyne vandyne 20 Mar 26 2018 new_temp
-rw-r--r-- 1 vandyne vandyne 25612 Apr 23 2018 oldmakep
-rw-r--r-- 1 vandyne vandyne 2851 Mar 23 2018 output
-rw-r--r-- 1 vandyne vandyne 21146 Mar 4 15:29 passwd
-rwx----- 1 vandyne vandyne 33 Mar 28 2018 pidof
drwx--s--t 12 vandyne staff 35 Dec 4 2017 preS18
drwxr-x--- 2 vandyne www-data 2 Jan 22 2018 public_html
-rwx----- 1 vandyne vandyne 639 Mar 28 2018 simple
-rwxr--r-- 1 vandyne vandyne 255 Mar 26 2018 simple.sh
-rw-r--r-- 1 vandyne vandyne 1191463 Mar 15 2019 sortedmoby2.txt
-rw-r--r-- 1 vandyne vandyne 1191463 Mar 15 2019 sortedmoby.txt
-rw-r--r-- 1 vandyne vandyne 2090 Mar 27 2018 Stats.class
-rw-r--r-- 1 vandyne vandyne 2135 Feb 26 2018 Stats.java
-rw-r--r-- 1 vandyne vandyne 20269 Mar 9 2018 StdDraw.class
-rw-r--r-- 1 vandyne vandyne 42868 Feb 28 2018 StdDraw.java
-rw-r--r-- 1 vandyne vandyne 73768 Mar 26 2018 temp
-rwxrwxrwx 1 vandyne vandyne 509 May 4 2018 temp.cpp
-rw-r--r-- 1 vandyne vandyne 3495 Mar 11 2019 temp.java
-rwxrwxrwx 1 vandyne vandyne 121 May 4 2018 temp.sh
-rw-r--r-- 1 vandyne vandyne 143360 Mar 22 2018 temp.tar
-rw-r--r-- 1 vandyne vandyne 6 Mar 21 2018 terminal
-rw-r--r-- 1 vandyne vandyne 27 Mar 26 2018 toosimple
-rwxr-xr-x 1 vandyne vandyne 474 Mar 25 2019 zzzprint_all.sh
vandyne@lumen:~$
```

Changing File Permissions

- `chmod options files`
- Two forms:
 - *options* as a sequence of three octal digits
 - ✦ first digit is for owner permissions
 - ✦ second for group permissions
 - ✦ third is for everyone else

```
drwxr-xr-x
-rw-r--r--
-rw-r--r--
-rw-r--r--
-rw-r--r--
drwxr-xr-x
```

- ✦ `chmod 600 private.txt`
`-rw-----`

Permission	Octal	Binary
---	0	000
--X	1	001
-W-	2	010
-WX	3	011
r--	4	100
r-X	5	101
rW-	6	110
rWX	7	111

Changing File Permissions

- `chmod options files`
- Second form:
 - *options* as a sequence of symbols
 - u – user, g – group, o, other, a – all, r – read, w – write, x – execute
 - “+” – add permission, “-” delete permission
 - ✦ `chmod ug=rw,o-rw,a-x private.txt`
`-rw-rw----`

File Contents

- `file filename(s)`
 - Analyzes a file's contents

```
$ file myprog.c letter.txt webpage.html
myprog.c:      C program text
letter.txt:    English text
webpage.html:  HTML document text
```

- `head, tail filename`
 - Displays the first or last few lines in a file
 - You can specify number of lines

```
$ tail -20 messages.txt ←
$ head -5 messages.txt ←
```

File Contents

- `od options filename`
 - Displays binary file contents in different formats

```
$ cat hello.txt ←  
hello world  
$ od -c hello.txt ←  
0000000  h e l l o      w o r l d \n  
0000014  
$ od -x hello.txt ←  
0000000  6865 6c6c 6f20 776f 726c 640a  
0000014
```

Finding Files

- `find directory -name targetfile -print`

```
$ find /home -name "*.txt" -print 2>/dev/null ←
```

- `which command`

```
$ which ls ←  
/bin/l
```

- `locate string`

```
$ locate ".txt" ←
```


Finding Text in Files

- `grep options patterns files`
- Stands for General Regular Expression Print

The caret `^' and the dollar sign `\$' are special characters that match the beginning and end of a line respectively. The dot '.' matches any character. So

```
$ grep ^..[l-z]$ hello.txt ←
```

matches any line in `hello.txt` that contains a three character sequence that ends with a lowercase letter from l to z.

- `egrep options patterns files`
- Stands for Extended grep
 - Can join expressions with or '|', can use parentheses for grouping
 - Can use other regular expression operators:
 - ✦ ?, *, +, {N}, {N,}, {N,M}

Sorting File Contents

- `sort filenames`

```
$ sort input1.txt input2.txt > output.txt ←
```

outputs the sorted concatenation of files `input1.txt` and `input2.txt` to the file `output.txt`.

- `uniq filename`

- Removes duplicate adjacent lines in a file, handy when combined with `sort`:

```
$ sort input.txt | uniq > output.txt ←
```

File Backup – tar – and Compression

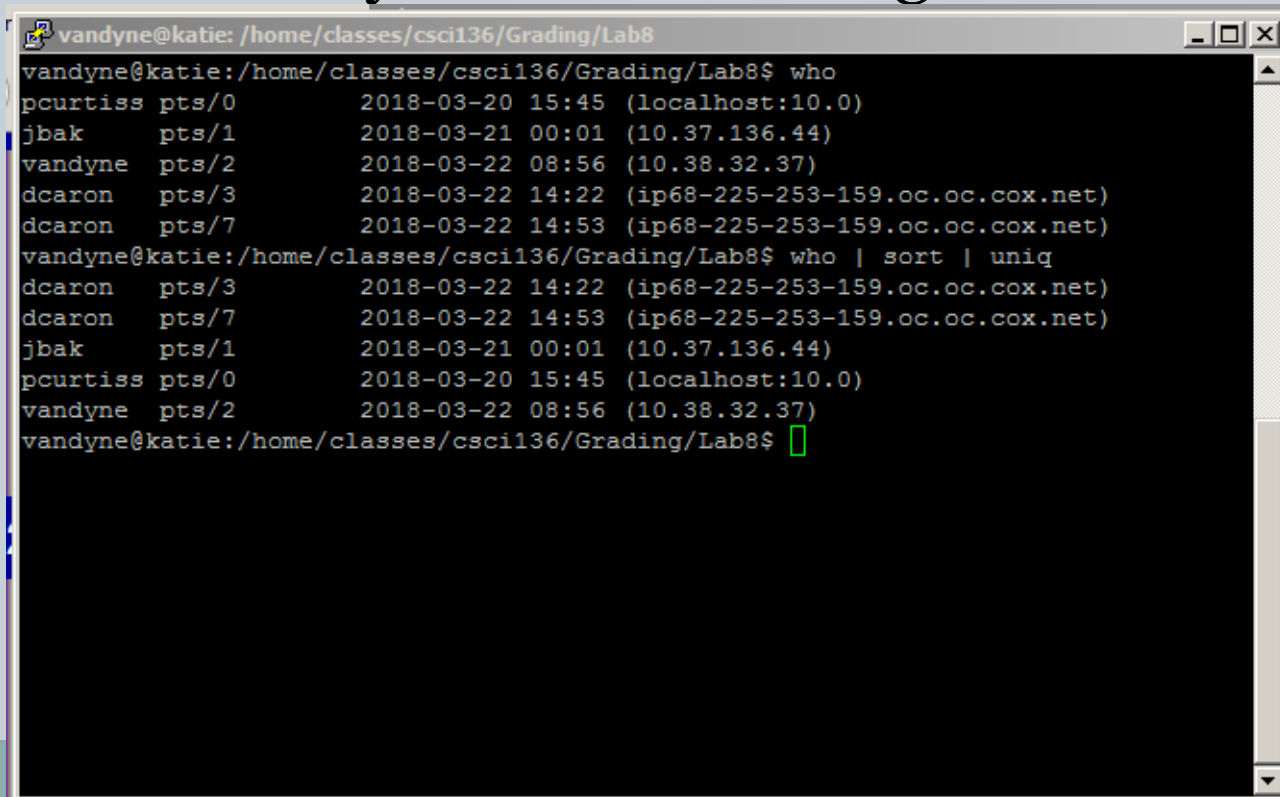
- To create a disk file tar archive:
 - `tar -cvf archivename filenames`
- To list the contents of an archive:
 - `tar -tvf archivename`
- To restore files from a tar archive:
 - `tar -xvf archivename`
- To compress files in a tar archive:
 - `compress filename`
- Or:
 - `gzip filename`

Processes

- A process is a program in execution
 - Each has a unique process identifier (PID)
 - The first process started when Linux boots is init
 - ✦ All processes are children of init
 - Can be any executing program:
 - ✦ Your running Python program
 - ✦ A command you are executing
 - ✦ A daemon started by the Linux kernel
 - ✦ Etc.

Pipes

- Pipe operator ‘|’ creates concurrently executing processes which pass data directly to one another
- Used to combine system utilities together



```
vandyne@katie: /home/classes/csci136/Grading/Lab8
vandyne@katie:/home/classes/csci136/Grading/Lab8$ who
pcurtiss pts/0      2018-03-20 15:45 (localhost:10.0)
jbak     pts/1      2018-03-21 00:01 (10.37.136.44)
vandyne  pts/2      2018-03-22 08:56 (10.38.32.37)
dcaron   pts/3      2018-03-22 14:22 (ip68-225-253-159.oc.oc.cox.net)
dcaron   pts/7      2018-03-22 14:53 (ip68-225-253-159.oc.oc.cox.net)
vandyne@katie:/home/classes/csci136/Grading/Lab8$ who | sort | uniq
dcaron   pts/3      2018-03-22 14:22 (ip68-225-253-159.oc.oc.cox.net)
dcaron   pts/7      2018-03-22 14:53 (ip68-225-253-159.oc.oc.cox.net)
jbak     pts/1      2018-03-21 00:01 (10.37.136.44)
pcurtiss pts/0      2018-03-20 15:45 (localhost:10.0)
vandyne  pts/2      2018-03-22 08:56 (10.38.32.37)
vandyne@katie:/home/classes/csci136/Grading/Lab8$
```

Input/Output Redirection

- **Linux treats everything as a file**
 - Including standard input (keyboard), standard output (screen) and standard error (also the screen)
 - We can redirect input or output from these “files” using the redirection operators `<` and `>`
 - ✦ The “arrow” points to where the input/output goes

Redirecting Standard Output

- To redirect standard output to a file instead of a screen, use the > operator:

```
$ echo hello ←  
hello  
$ echo hello > output ←  
$ cat output ←  
hello
```

- This will create a new blank file each time
 - If you want to append to a file, use >>

```
$ echo bye >> output ←  
$ cat output ←  
hello  
bye
```

Redirecting Standard Error to a File

- Standard input (0), standard output (1) and standard error (2) have those numbers associated with them
- To output any error messages to a file, use 2>

```
$ cat nonexistent 2>errors ←  
$ cat errors ←  
cat: nonexistent: No such file or directory  
$
```

- To output results to one file and errors to another:
 - `find . -print 1>files 2>errors`
- This is **very** handy when you are compiling a program and you get a whole list of error messages

Redirecting Standard Input

- Input will be read from a file rather than from keyboard input

```
$ cat < output ←  
hello  
bye
```

- Remember – the “arrow” points to where the data will go
 - In this case it will come from a file and go into the command “cat”

Controlling Processes

- You can run several processes at the same time
 - Processes can run in the foreground (the thing you currently see) or in the background (you don't see it running, but it is)
- To start a process in the background, use & at the end of the command line:

```
$ find / -print 1>output 2>errors & ←  
[1] 27501  
$
```

- [1] is the job number and 27501 is the process id (PID)
- Note: if your background process prints to the screen, it will show up as you're doing something else

Controlling Processes

- To put the current process in the background:
 - Type Ctrl-Z
- To bring a background process to the foreground:
 - `fg %<job number>`
- To see all of the processes you have running:
 - `ps`

```
$ ps ←  
  PID TTY          TIME CMD  
17717 pts/10        00:00:00 bash  
27501 pts/10        00:00:01 find  
27502 pts/10        00:00:00 ps
```

Controlling Processes

- What if you have a process you want to stop? (Maybe it's in an infinite loop, maybe... it's just a bad process?)
 - You can use:
 - ✦ `kill %<job number>` OR
 - ✦ `kill <PID>`
 - These are polite ways of asking the process to terminate
 - Processes are not always polite in return...
 - ✦ `kill -9 <PID>`
 - This will kill them on contact

Summary

- File and Directory Permissions
- File Content
- Finding Files
- Sorting Files
- File Compression
- Processes
- Pipes
- Input/Output Redirection
- Controlling Processes

