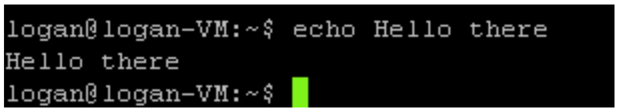
Linux Command Guide

- Logan -

For this guide, I will try to start simple and work my way up to the more complicated topics using layman terms. From echo, to scripting.

echo

The echo command is very simple, yet very powerful as you'll eventually come to learn. Simply type echo, followed by whatever you want to be echoed, and the command will echo it back at you. Example:



```
logan@logan-VM:~$ echo Hello there
Hello there
logan@logan-VM:~$
```

Man

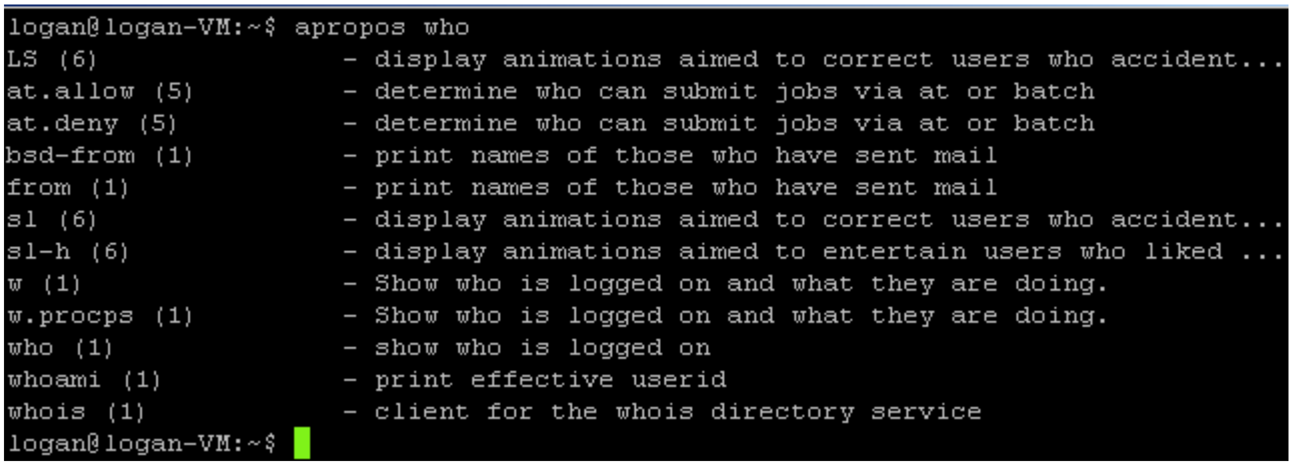
If you don't know any other command, you should at least know man. Man is short for manual and it's a command used to retrieve some help and documentation (manual pages) on various BASH commands for learning. To use it, simply type man followed by the name of the command about which you are interested in learning. For example:

```
man echo
```

will retrieve the manual page for the grep command for your endless reading pleasure.

Apropos

The man pages have short descriptions that you can search using the command apropos. If you have a very basic idea of what the command you're looking for does, but aren't sure what it's called, you can type apropos, followed by a word associated with the function of the command to find it. For example:



```
logan@logan-VM:~$ apropos who
LS (6)               - display animations aimed to correct users who accident...
at.allow (5)         - determine who can submit jobs via at or batch
at.deny (5)          - determine who can submit jobs via at or batch
bsd-from (1)         - print names of those who have sent mail
from (1)             - print names of those who have sent mail
sl (6)               - display animations aimed to correct users who accident...
sl-h (6)             - display animations aimed to entertain users who liked ...
w (1)                - Show who is logged on and what they are doing.
w.procps (1)         - Show who is logged on and what they are doing.
who (1)              - show who is logged on
whoami (1)           - print effective userid
whois (1)            - client for the whois directory service
logan@logan-VM:~$
```

This shows a list of short man page descriptions that contain the keyword “who”.

Info / pinfo

The info command is a lot like the man command, except it will often retrieve more detailed help than that found in the man pages. It is used in the same way as the man command. Info is also more relevant to GNU based utilities, and sometimes, if you can't find a man page for a command, you can find the info for it. Example:

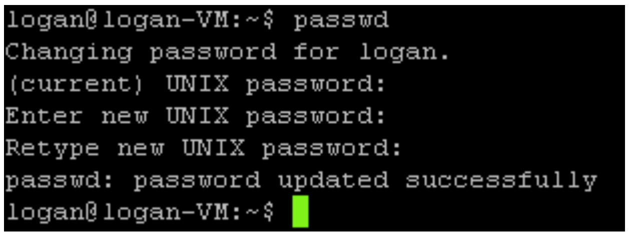
```
info awk
```

Pinfo is the same as info, except its output is color-coded.

Example:

```
pinfo sort
```

Passwd

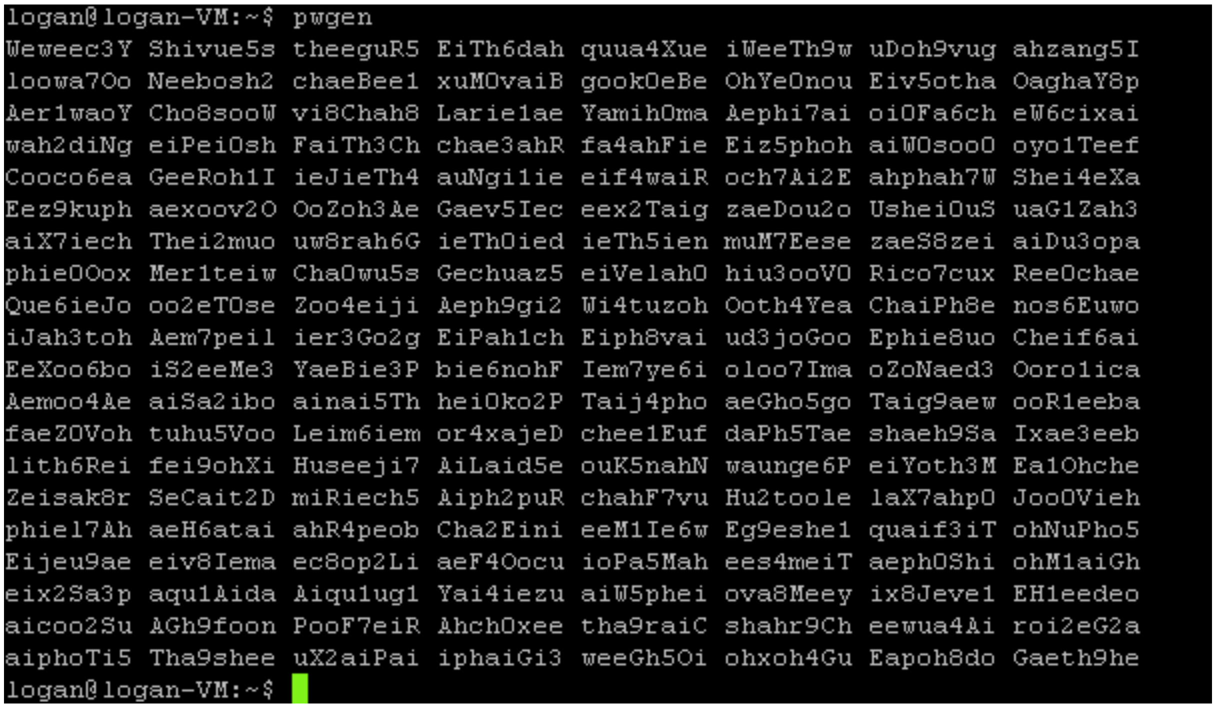


```
logan@logan-VM:~$ passwd
Changing password for logan.
(current) UNIX password:
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
logan@logan-VM:~$
```

The passwd command is for changing passwords for users. Example:

pwgen

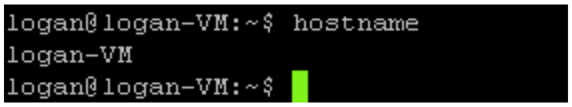
pwgen is a utility for easily generating random passwords. Like this:



```
logan@logan-VM:~$ pwgen
Weweec3Y Shivue5s theeguR5 EiTh6dah quua4Xue iWeeTh9w uDoh9vug ahzang5I
loowa7Oo Neebosh2 chaeBee1 xuM0vaiB gook0eBe OhYe0nou Eiv5otha OaghaY8p
Aer1waoY Cho8sooW vi8Chah8 Larie1ae Yamih0ma Aephi7ai oi0Fa6ch eW6cixai
wah2diNg eiPei0sh FaiTh3Ch chae3ahR fa4ahFie Eiz5phoh aiW0soo0 oyo1Teef
Coocо6ea GeeRoh1I ieJieTh4 auNgi1ie eif4waiR och7Ai2E ahphah7W Shei4eXa
Eez9kuph aexoov2O OoZoh3Ae Gaev5Iec eex2Taig zaeDou2o Ushei0uS uaG1Zah3
aiX7iech Thei2muo uw8rah6G ieTh0ied ieTh5ien muM7Eese zaeS8zei aiDu3opa
phie0Oox Mer1teiw Cha0wu5s Gechuaz5 eiVelah0 hiu3ooV0 Rico7cux Ree0chae
Que6ieJo oo2eT0se Zoo4eiji Aeph9gi2 Wi4tuzoh Ooth4Yea ChaiPh8e nos6Euwo
iJah3toh Aem7peil ier3Go2g EiPah1ch Eiph8vai ud3joGoo Ephie8uo Cheif6ai
EeXoo6bo iS2eeMe3 YaeBie3P bie6nohF Iem7ye6i oloo7Ima oZoNaed3 Ooro1ica
Aemoo4Ae aiSa2ibo ainai5Th hei0ko2P Taij4pho aeGho5go Taig9aew ooR1eeba
faeZ0Voh tuhu5Voo Leim6iem or4xajeD chee1Euf daPh5Tae shaeh9Sa Ixae3eeb
lith6Rei fei9ohXi Huseeji7 AiLaid5e ouK5nahN waunge6P eiYoth3M Ea1Ohche
Zeisak8r SeCait2D miRiech5 Aiph2puR chahF7vu Hu2toole laX7ahp0 Joo0Vieh
phie17Ah aeH6atai ahR4peob Cha2Eini eeM1Ie6w Eg9eshe1 quaif3iT ohNuPho5
Eijeu9ae eiv8Iema ec8op2Li aeF4Oocu ioPa5Mah ees4meiT aeph0Shi ohM1aiGh
eix2Sa3p aqu1Aida Aiqu1ug1 Yai4iezu aiW5phei ova8Meey ix8Jeve1 EH1eedeo
aicoo2Su AGh9foon PooF7eiR Ahch0xee tha9raiC shahr9Ch eewua4Ai roi2eG2a
aiphoTi5 Tha9shee uX2aiPai iphaiGi3 weeGh5Oi ohxoh4Gu Eapoh8do Gaeth9he
logan@logan-VM:~$
```

hostname

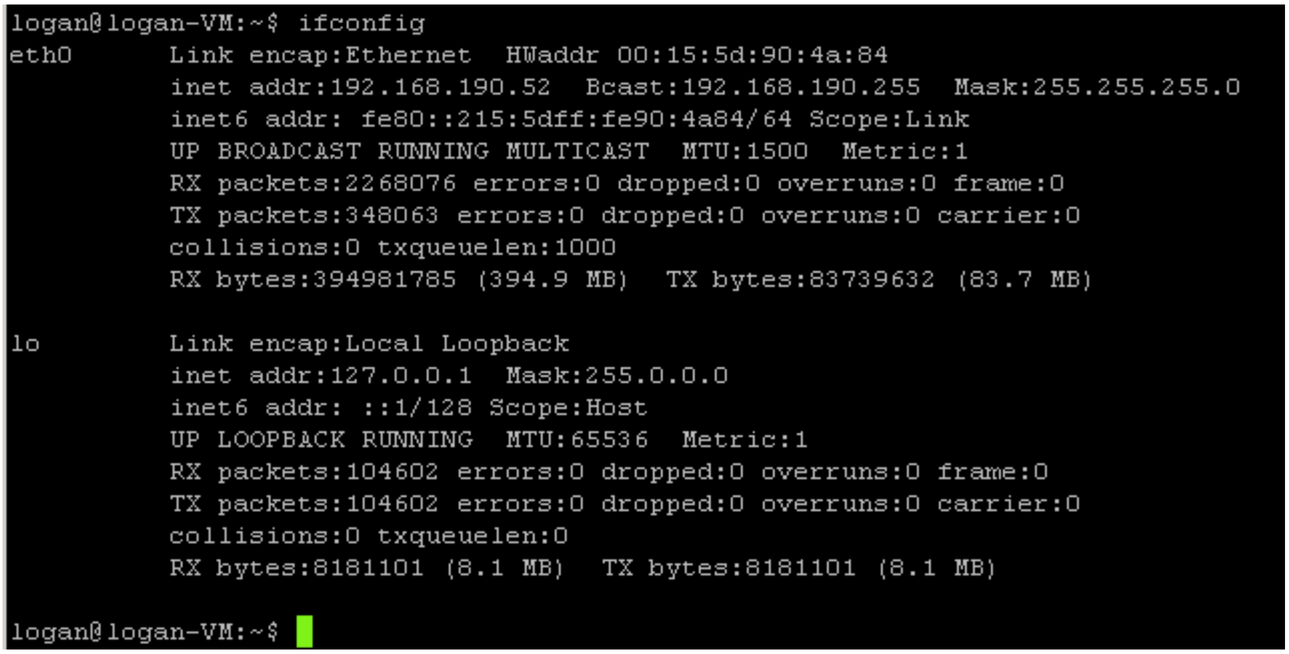
The hostname command can be used to display the currently configured hostname of the local system:



```
logan@logan-VM:~$ hostname
logan-VM
logan@logan-VM:~$
```

ifconfig

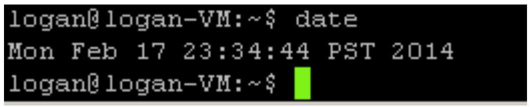
The ifconfig utility displays a bunch of network interface configuration info, and can be used to configure the interfaces given the proper options and command syntax are used, but I've never tried that before. Here's just some quick interface info:



```
logan@logan-VM:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:15:5d:90:4a:84
          inet addr:192.168.190.52  Bcast:192.168.190.255  Mask:255.255.255.0
          inet6 addr: fe80::215:5dff:fe90:4a84/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:2268076 errors:0 dropped:0 overruns:0 frame:0
          TX packets:348063 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:394981785 (394.9 MB)  TX bytes:83739632 (83.7 MB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:104602 errors:0 dropped:0 overruns:0 frame:0
          TX packets:104602 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:8181101 (8.1 MB)  TX bytes:8181101 (8.1 MB)

logan@logan-VM:~$
```

date

The date command displays the current date and time by default.



```
logan@logan-VM:~$ date
Mon Feb 17 23:34:44 PST 2014
logan@logan-VM:~$
```

(I know I stay up really late sometimes.)

more and less

The more command will take the contents of a file, or the output from another command (see pipes), and send it to the console screen in screen-sized allotments so that the user can press the the down arrow key or the space bar to scroll through the entirety of the file contents (or command output) line by line or page by page, respectively. As a good comparison, the man command has its output paged in sort of in this manner, just without using the more or less commands.

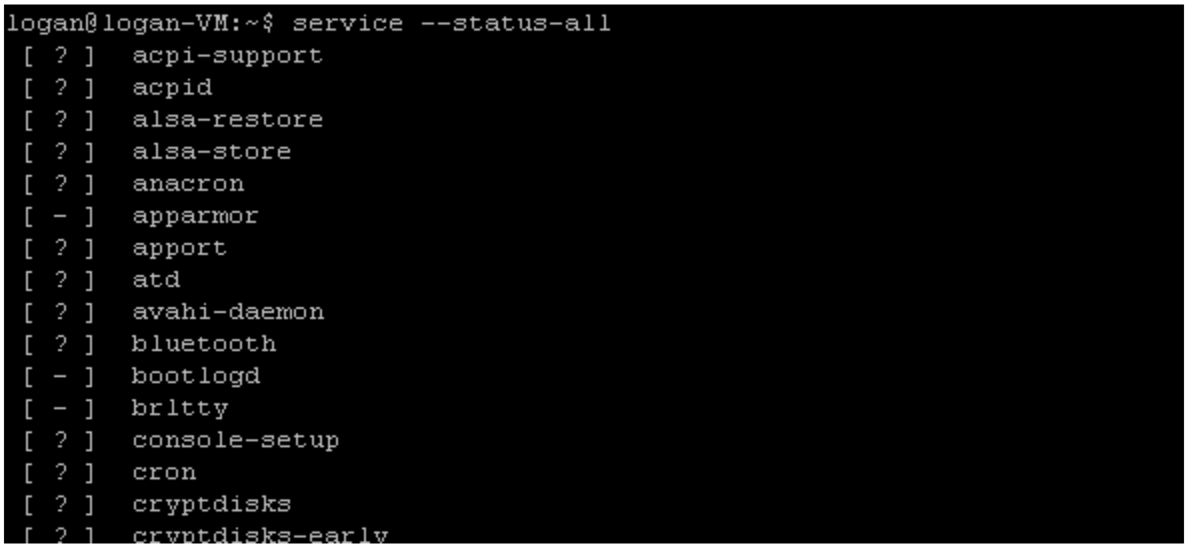
The less command is the same except that it allows the user to scroll up as well as down.

The Up and Down Arrow Keys

The up and down arrow keys can be used to cycle back and forth between previously entered commands. Up for previous, down for next. They are also used with paged output to scroll up and down.

service

The service command is used to manage services. If you want a list of services and their current status, use the –status-all option:

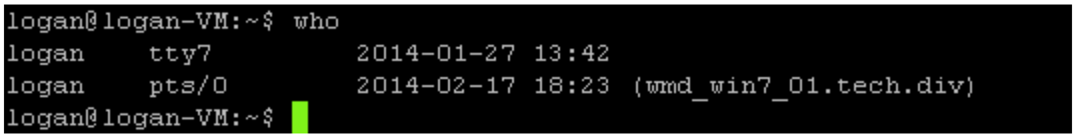


```
logan@logan-VM:~$ service --status-all
 [ ? ]  acpi-support
 [ ? ]  acpid
 [ ? ]  alsa-restore
 [ ? ]  alsa-store
 [ ? ]  anacron
 [ - ]  apparmor
 [ ? ]  apport
 [ ? ]  atd
 [ ? ]  avahi-daemon
 [ ? ]  bluetooth
 [ - ]  bootlogd
 [ - ]  brltty
 [ ? ]  console-setup
 [ ? ]  cron
 [ ? ]  cryptdisks
 [ ? ]  cryptdisks-early
```

The list is long. The column on the left is the status; I think '-' means the service isn't running, I'm not sure what the questions mark means, and I think a '+' indicates the service is running.

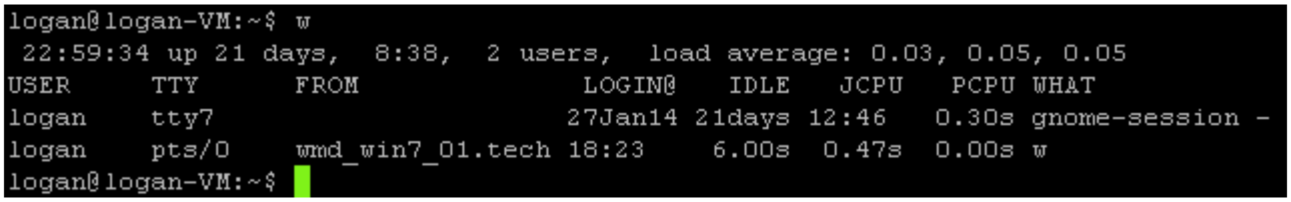
who and w

The who command displays a list of users who are logged on.



```
logan@logan-VM:~$ who
logan    tty7         2014-01-27 13:42
logan    pts/0        2014-02-17 18:23 (wmd_win7_01.tech.div)
logan@logan-VM:~$
```

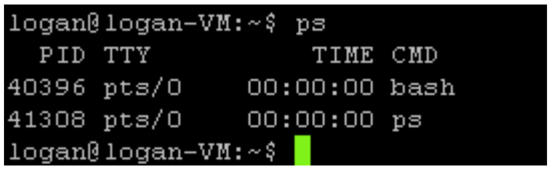
The w command displays who is logged in and what processes they are running.



```
logan@logan-VM:~$ w
 22:59:34 up 21 days,  8:38,  2 users,  load average: 0.03, 0.05, 0.05
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
logan    tty7                      27Jan14 21days 12:46   0.30s gnome-session -
logan    pts/0    wmd_win7_01.tech 18:23    6.00s  0.47s  0.00s w
logan@logan-VM:~$
```

ps

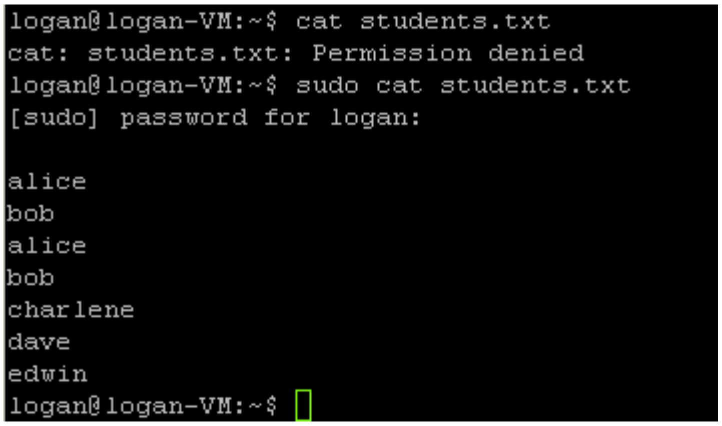
The ps command displays the processes running at the moment the command was entered.



```
logan@logan-VM:~$ ps
  PID TTY          TIME CMD
40396 pts/0    00:00:00 bash
41308 pts/0    00:00:00 ps
logan@logan-VM:~$
```

sudo

The command sudo allows the user, *if he or she is authorized,* to execute a command as root, or as another user if specified. To demonstrate:



```
logan@logan-VM:~$ cat students.txt
cat: students.txt: Permission denied
logan@logan-VM:~$ sudo cat students.txt
[sudo] password for logan:

alice
bob
alice
bob
charlene
dave
edwin
logan@logan-VM:~$
```

Without sudo, permission was denied. With it, free reign was granted—no questions asked.

Use it wisely.

script

The script command, once entered, will start recording every command you enter at the terminal and saving it all to a file named typescript in your current directory. To start it, just type 'script'.

clear

The clear command clears the console buffer so it's empty like this:

```
logan@logan-VM:~$
```

All you need to enter is “clear”.

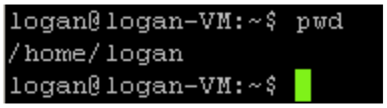
File System

Some of the most important commands are those used for managing the file system and its contents. This is what I'll try to cover in the next sections.

Navigating

pwd

The pwd command stands for present working directory and is used to verify the absolute path of the directory in which the user is working. Here's an example:



```
logan@logan-VM:~$ pwd
/home/logan
logan@logan-VM:~$
```

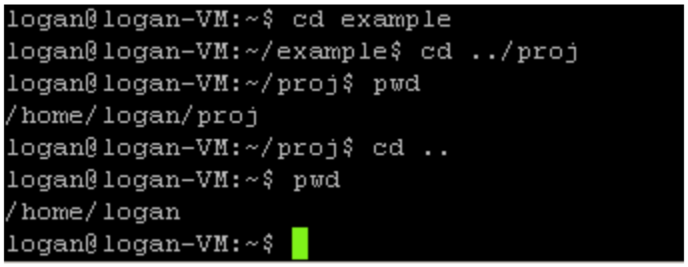
This shows that I'm working in my home directory /home/logan.

Absolute and Relative Paths

An absolute pathname includes the root directory, the target directory or file name, and all the directories directly in between, regardless of the present working directory at the moment the command with the pathname is entered. A relative pathname on the other hand is based primarily on the present working directory, and it traces a path from it, because that's what it's relative to. You can recognize a pathname as being relative by its beginning with a forward-slash (/) specifying the root directory as part of the pathname. If I specify a filename without directories preceding it, I am referring to a file by that name in working directory; thus, the use of simple filenames is an example of relative pathnames.

. and ..

In any given directory, you can use '.' and '..' to refer to the current directory and the parent directory, respectively. We'll learn a little more about '.' later, but here's an example of how '..' can be used:

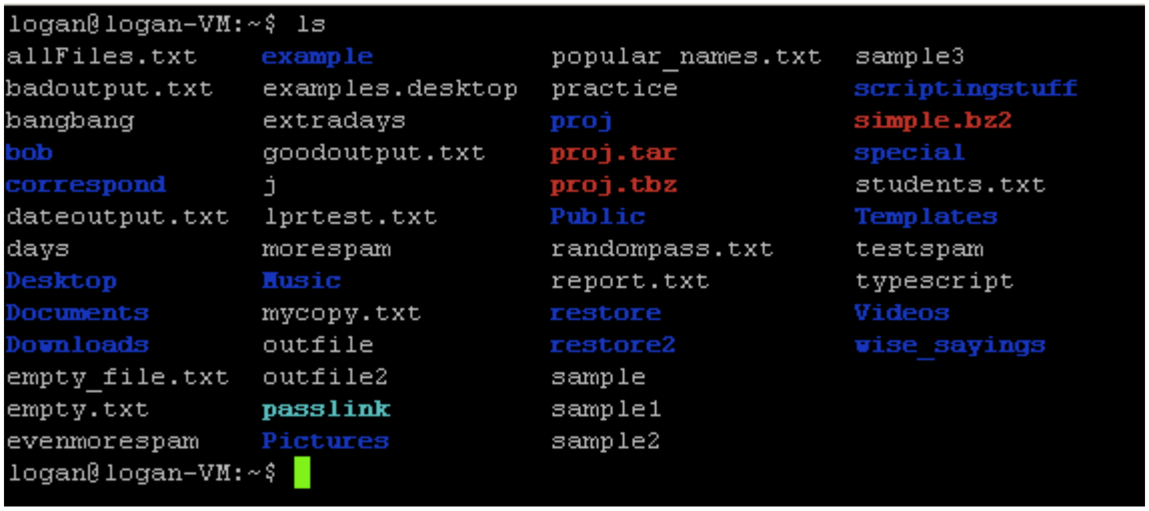


```
logan@logan-VM:~$ cd example
logan@logan-VM:~/example$ cd ../proj
logan@logan-VM:~/proj$ pwd
/home/logan/proj
logan@logan-VM:~/proj$ cd ..
logan@logan-VM:~$ pwd
/home/logan
logan@logan-VM:~$
```

ls

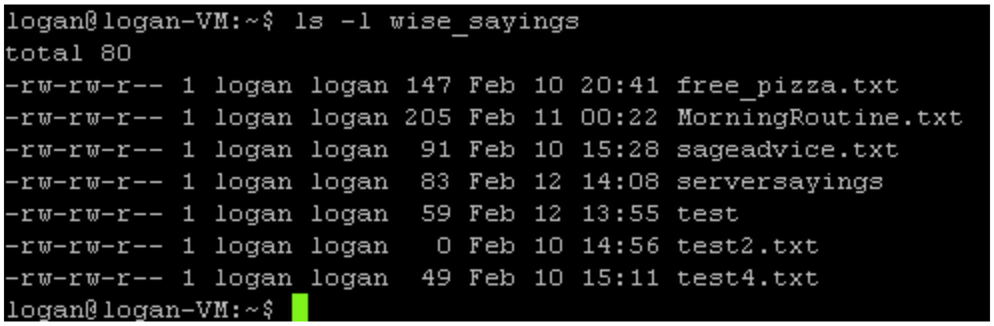
The command ls is used to list the contents of a specified directory (or the current if unspecified).

Here's the contents of my home directory:



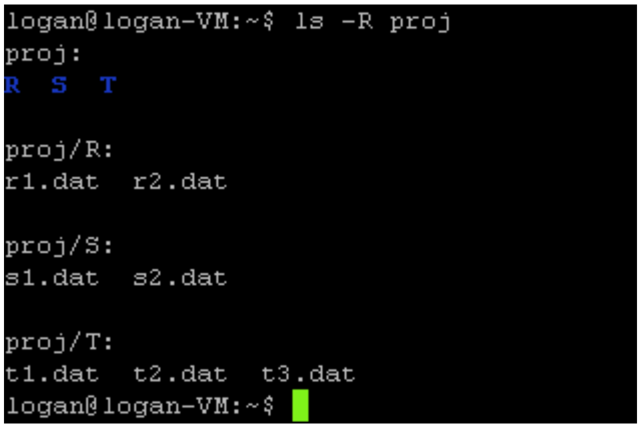
```
logan@logan-VM:~$ ls
allFiles.txt    example           popular_names.txt  sample3
badoutput.txt   examples.desktop  practice           scriptingstuff
bangbang        extradays         proj               simple.bz2
bob             goodoutput.txt    proj.tar           special
correspond      j                 proj.tbz           students.txt
dateoutput.txt  lprtest.txt       Public             Templates
days            morespam          randompass.txt     testspam
Desktop         Music             report.txt         typescript
Documents       mycopy.txt        restore            Videos
Downloads       outfile           restore2           wise_sayings
empty_file.txt  outfile2          sample
empty.txt       passlink          sample1
evenmorespam    Pictures          sample2
logan@logan-VM:~$
```

the -l option, which means long-listing, will give more details in a columned list of the files. Like so:



```
logan@logan-VM:~$ ls -l wise_sayings
total 80
-rw-rw-r-- 1 logan logan 147 Feb 10 20:41 free_pizza.txt
-rw-rw-r-- 1 logan logan 205 Feb 11 00:22 MorningRoutine.txt
-rw-rw-r-- 1 logan logan  91 Feb 10 15:28 sageadvice.txt
-rw-rw-r-- 1 logan logan  83 Feb 12 14:08 serversayings
-rw-rw-r-- 1 logan logan  59 Feb 12 13:55 test
-rw-rw-r-- 1 logan logan   0 Feb 10 14:56 test2.txt
-rw-rw-r-- 1 logan logan  49 Feb 10 15:11 test4.txt
logan@logan-VM:~$
```

the -R option will list the contents of the current directory, along with that of all subdirectories. Like so:

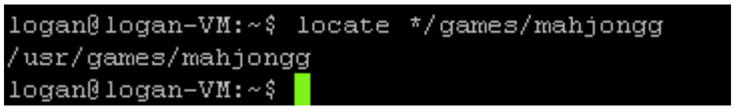


```
logan@logan-VM:~$ ls -R proj
proj:
R  S  T

proj/R:
r1.dat  r2.dat

proj/S:
s1.dat  s2.dat

proj/T:
t1.dat  t2.dat  t3.dat
logan@logan-VM:~$
```

locate (aka mlocate)

The locate command is for finding files (always avoid annoying alliteration) by full path name in the file system. Just type locate, followed by a search pattern that you think might match the name of the file you seek; you can use globbing characters by default. The locate utility uses an index that in some cases might not be the most up to date concerning what files actually exist, but if you're looking for something that is almost always in the same place, you should be able to locate it.

Example:

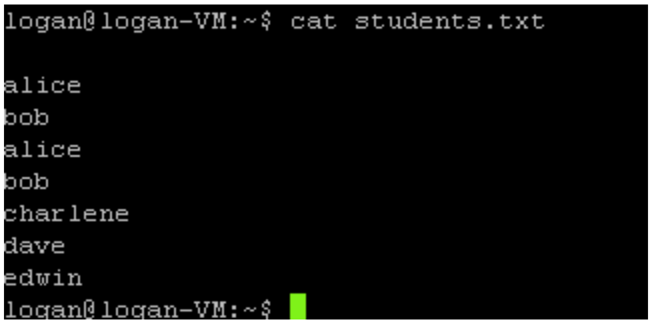


```
logan@logan-VM:~$ locate */games/mahjongg
/usr/games/mahjongg
logan@logan-VM:~$
```

This is how I imagine someone might try to find mahjongg if all they knew was that it was in a folder called games.

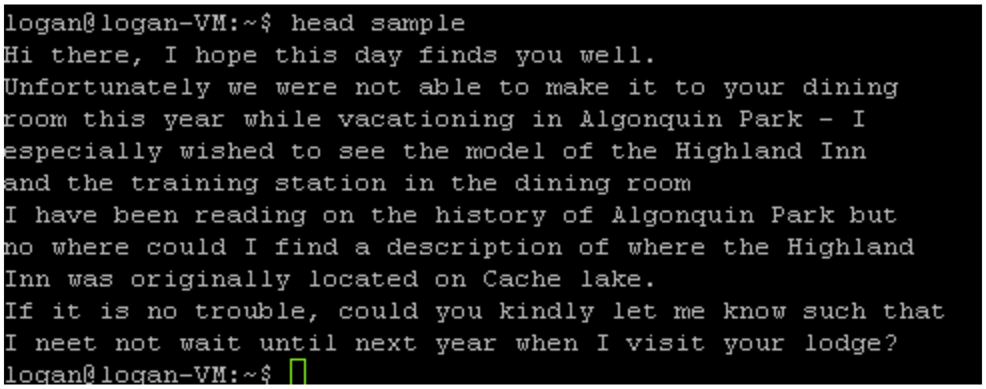
cat, head, and tail

The command cat takes the contents of a file and sends it to stdout. Example:



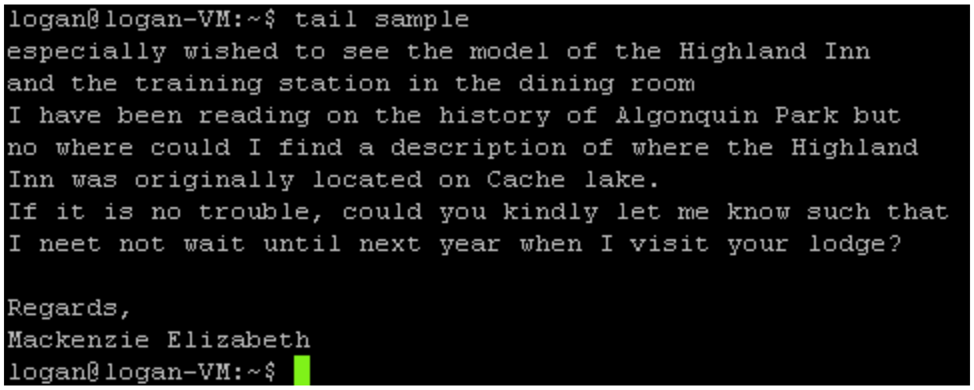
```
logan@logan-VM:~$ cat students.txt

alice
bob
alice
bob
charlene
dave
edwin
logan@logan-VM:~$
```

head does the same thing, except it will only output the first few (ten by default) lines of a file.



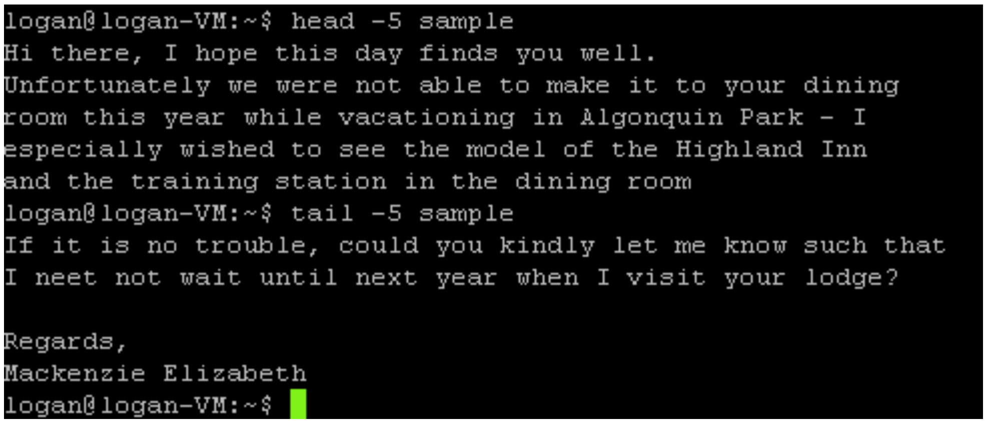
```
logan@logan-VM:~$ head sample
Hi there, I hope this day finds you well.
Unfortunately we were not able to make it to your dining
room this year while vacationing in Algonquin Park - I
especially wished to see the model of the Highland Inn
and the training station in the dining room
I have been reading on the history of Algonquin Park but
no where could I find a description of where the Highland
Inn was originally located on Cache lake.
If it is no trouble, could you kindly let me know such that
I neet not wait until next year when I visit your lodge?
logan@logan-VM:~$
```

tail is like head, except that instead of the first few lines, it outputs the last few lines (still ten by default).



```
logan@logan-VM:~$ tail sample
especially wished to see the model of the Highland Inn
and the training station in the dining room
I have been reading on the history of Algonquin Park but
no where could I find a description of where the Highland
Inn was originally located on Cache lake.
If it is no trouble, could you kindly let me know such that
I neet not wait until next year when I visit your lodge?

Regards,
Mackenzie Elizabeth
logan@logan-VM:~$
```

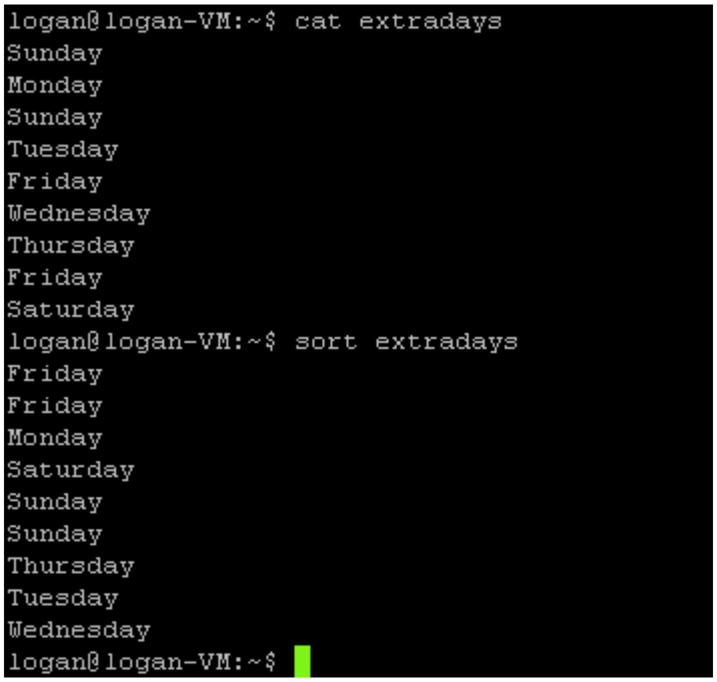
With both head and tail, you can specify the number of lines from the start (head) or end (tail) of the file to send to stdout, like this:



```
logan@logan-VM:~$ head -5 sample
Hi there, I hope this day finds you well.
Unfortunately we were not able to make it to your dining
room this year while vacationing in Algonquin Park - I
especially wished to see the model of the Highland Inn
and the training station in the dining room
logan@logan-VM:~$ tail -5 sample
If it is no trouble, could you kindly let me know such that
I neet not wait until next year when I visit your lodge?

Regards,
Mackenzie Elizabeth
logan@logan-VM:~$
```

sort

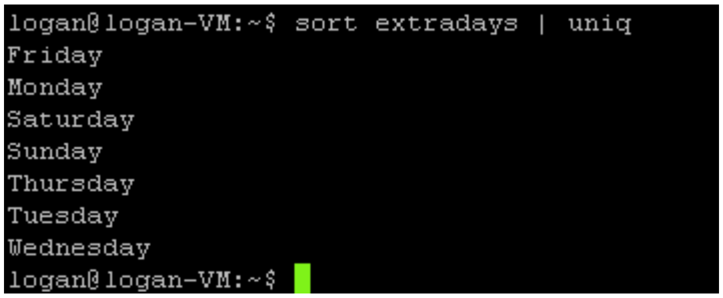
The sort command takes the lines of a file or standard input, and sorts them (alphabetically by default).



```
logan@logan-VM:~$ cat extradays
Sunday
Monday
Sunday
Tuesday
Friday
Wednesday
Thursday
Friday
Saturday
logan@logan-VM:~$ sort extradays
Friday
Friday
Monday
Saturday
Sunday
Sunday
Thursday
Tuesday
Wednesday
logan@logan-VM:~$
```

uniq

The uniq command takes the lines of a file or standard input, and (by default) removes any lines that are identical to the line directly previous.

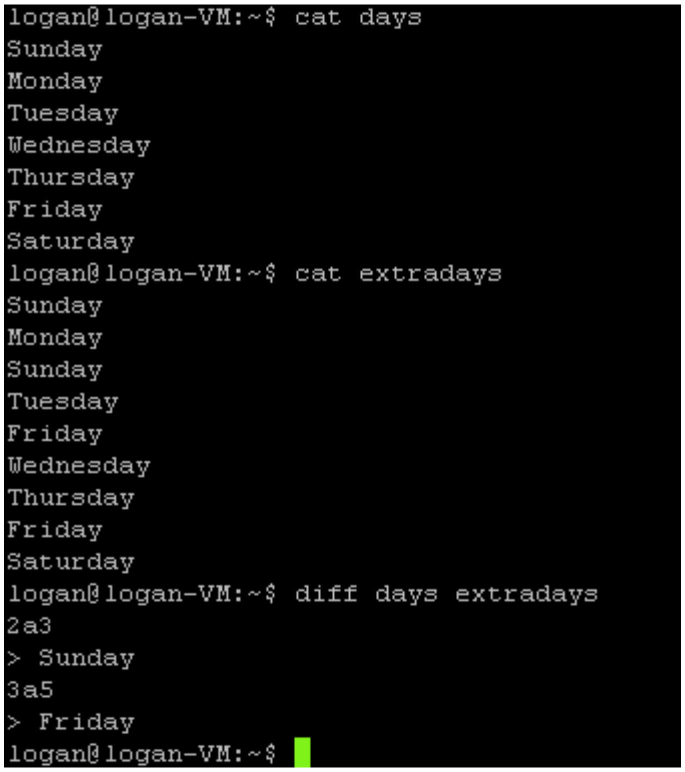


```
logan@logan-VM:~$ sort extradays | uniq
Friday
Monday
Saturday
Sunday
Thursday
Tuesday
Wednesday
logan@logan-VM:~$
```

In this example, I'm piping (see pipes later) the output of the command "sort extradays" into the standard input of the command uniq, and then uniq does its job by removing any lines identical to previous lines. Compare this screen shot to the previous one.

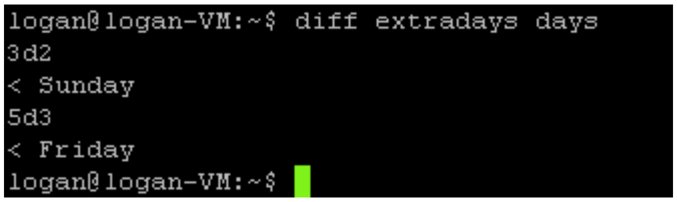
diff

The diff (short for difference) command finds the differences between to files by comparing them both one line at a time.



```
logan@logan-VM:~$ cat days
Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
logan@logan-VM:~$ cat extradays
Sunday
Monday
Sunday
Tuesday
Friday
Wednesday
Thursday
Friday
Saturday
logan@logan-VM:~$ diff days extradays
2a3
> Sunday
3a5
> Friday
logan@logan-VM:~$
```

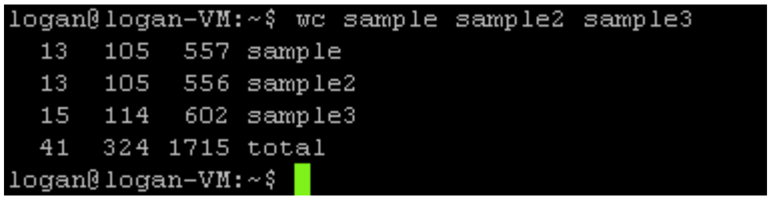
The command shows that there's a Sunday and a Friday in the extradays file that the days file doesn't have. The right angle bracket tells us that the second of the two files specified is the one with the extra differences. If I were to specify the files in reverse order, the angle bracket would be facing the opposite direction:



```
logan@logan-VM:~$ diff extradays days
3d2
< Sunday
5d3
< Friday
logan@logan-VM:~$
```

wc

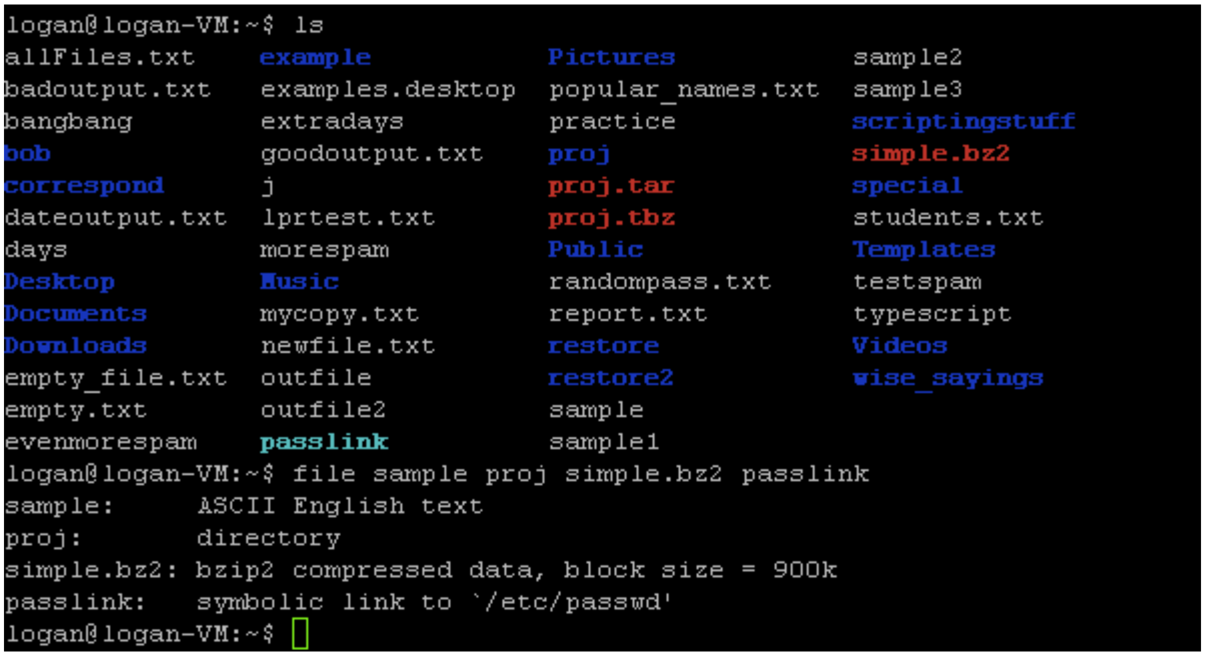
The wc command will output the number of newlines, words, and bytes for the specified file(s):



```
logan@logan-VM:~$ wc sample sample2 sample3
  13  105  557 sample
  13  105  556 sample2
  15  114  602 sample3
  41  324 1715 total
logan@logan-VM:~$
```

file

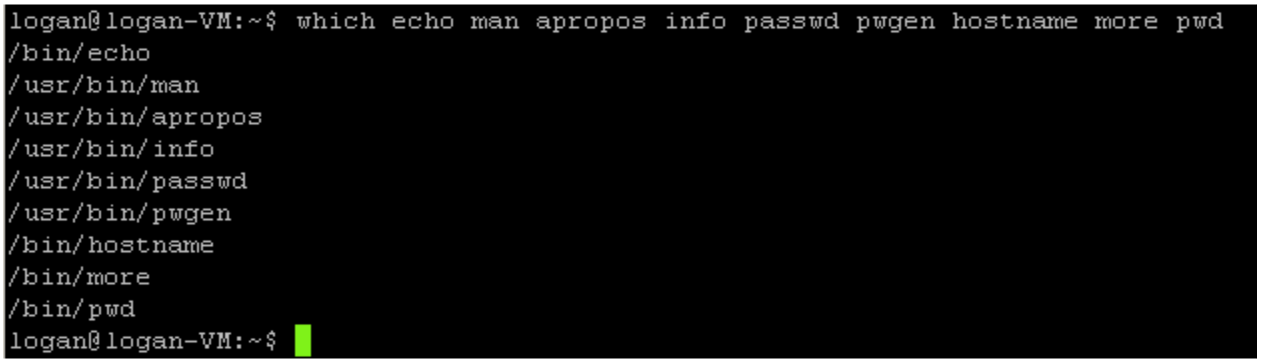
The file command can be used to identify the type of a specific file or files. Here are a few files being tested with this command:



```
logan@logan-VM:~$ ls
allFiles.txt    example           Pictures           sample2
badoutput.txt   examples.desktop  popular_names.txt  sample3
bangbang        extradays         practice           scriptingstuff
bob             goodoutput.txt    proj               simple.bz2
correspond      j                 proj.tar           special
dateoutput.txt  lprtest.txt       proj.tbz           students.txt
days            morespam          Public             Templates
Desktop         Music             randompass.txt     testspam
Documents       mycopy.txt        report.txt         typescript
Downloads       newfile.txt       restore            Videos
empty_file.txt  outfile           restore2           wise_sayings
empty.txt       outfile2          sample
evenmorespam    passlink          sample1
logan@logan-VM:~$ file sample proj simple.bz2 passlink
sample:      ASCII English text
proj:        directory
simple.bz2: bzip2 compressed data, block size = 900k
passlink:    symbolic link to `/etc/passwd'
logan@logan-VM:~$
```

which

As you may have figured out by now, the commands used in bash are all software programs that physically reside somewhere on the hard drive. If you ever wanted to see where, just type "which" followed by the name of the command you want to locate.

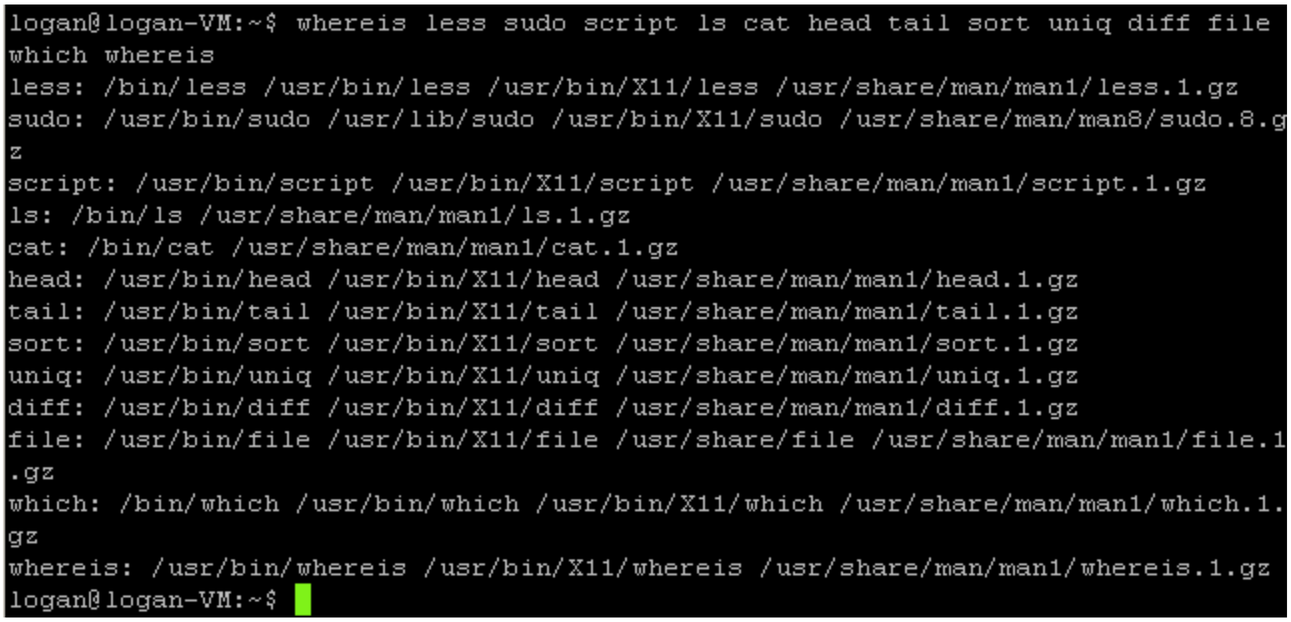


```
logan@logan-VM:~$ which echo man apropos info passwd pwgen hostname more pwd
/bin/echo
/usr/bin/man
/usr/bin/apropos
/usr/bin/info
/usr/bin/passwd
/usr/bin/pwgen
/bin/hostname
/bin/more
/bin/pwd
logan@logan-VM:~$
```

Here the which command lists the respective locations of each utility named.

whereis

There's also a command called whereis to display the location of the binary, source, ***and*** man page files for specified commands. Like so:



```
logan@logan-VM:~$ whereis less sudo script ls cat head tail sort uniq diff file
which whereis
less: /bin/less /usr/bin/less /usr/bin/X11/less /usr/share/man/man1/less.1.gz
sudo: /usr/bin/sudo /usr/lib/sudo /usr/bin/X11/sudo /usr/share/man/man8/sudo.8.g
z
script: /usr/bin/script /usr/bin/X11/script /usr/share/man/man1/script.1.gz
ls: /bin/ls /usr/share/man/man1/ls.1.gz
cat: /bin/cat /usr/share/man/man1/cat.1.gz
head: /usr/bin/head /usr/bin/X11/head /usr/share/man/man1/head.1.gz
tail: /usr/bin/tail /usr/bin/X11/tail /usr/share/man/man1/tail.1.gz
sort: /usr/bin/sort /usr/bin/X11/sort /usr/share/man/man1/sort.1.gz
uniq: /usr/bin/uniq /usr/bin/X11/uniq /usr/share/man/man1/uniq.1.gz
diff: /usr/bin/diff /usr/bin/X11/diff /usr/share/man/man1/diff.1.gz
file: /usr/bin/file /usr/bin/X11/file /usr/share/file /usr/share/man/man1/file.1
.gz
which: /bin/which /usr/bin/which /usr/bin/X11/which /usr/share/man/man1/which.1.
gz
whereis: /usr/bin/whereis /usr/bin/X11/whereis /usr/share/man/man1/whereis.1.gz
logan@logan-VM:~$
```

Managing Files and Directories

touch

touch is a command generally used to create empty files.

Example:

```
touch emptyfile.txt
```

cp

cp (meaning copy) is a command used take a file (or files), and replicate its (or their) contents to a new file (with a different name or in a different location).

Example:

```
cp oldfile.txt newfile.txt
```

Redirection

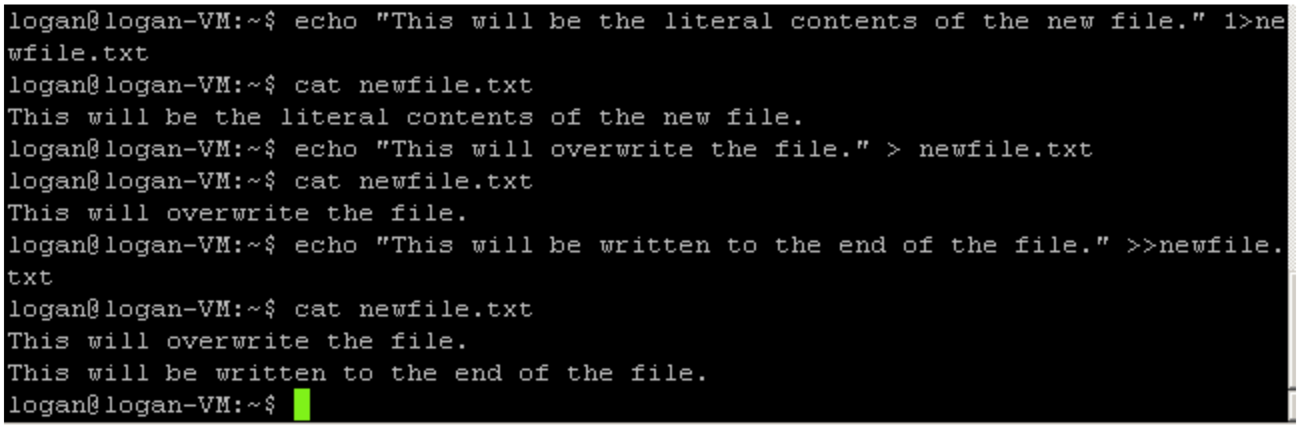
Redirection is where you take the stdin, stdout, and/or stderr of a command and redirect it, using a right angle bracket (>) to a file with a relative or absolute pathname. There are three files descriptors: 0 for stdin, 1 for stdout, and 2 for stderr.

For example:

```
echo "This will be the literal contents of the new file." 1>
newfile.txt
```

Another thing you can do with redirection is append to a file, which means that, instead of overwriting the file to which you redirect, you simply write to the end of the file. To do this, simply use TWO angle brackets (>>) and the command interpreter will know that you want it to append.

Here's a simple demonstration of redirection:



```
logan@logan-VM:~$ echo "This will be the literal contents of the new file." 1>ne
wfile.txt
logan@logan-VM:~$ cat newfile.txt
This will be the literal contents of the new file.
logan@logan-VM:~$ echo "This will overwrite the file." > newfile.txt
logan@logan-VM:~$ cat newfile.txt
This will overwrite the file.
logan@logan-VM:~$ echo "This will be written to the end of the file." >>newfile.
txt
logan@logan-VM:~$ cat newfile.txt
This will overwrite the file.
This will be written to the end of the file.
logan@logan-VM:~$
```

Text Editors and Word Processor (and various other programs)

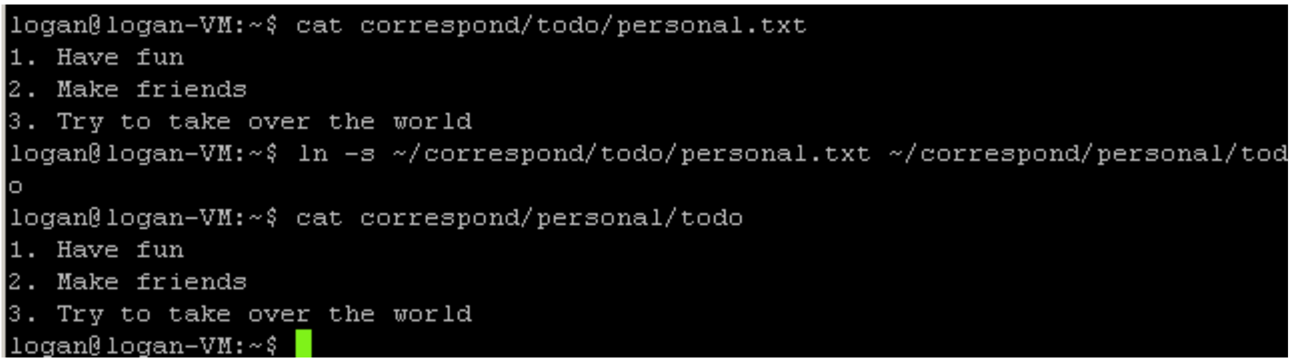
You can usually use a word processor or text editor such as vi to create files. The files are created when you "write" or save them. I won't go into to much detail on using vi in this document.

At any rate, vi is generally what one would use to make detailed edits to files that have already been created.

ln

The ln command is used for creating links of various kinds of which I will focus only on symbolic.

To create a symbolic link to a file, use the -s option with ln.

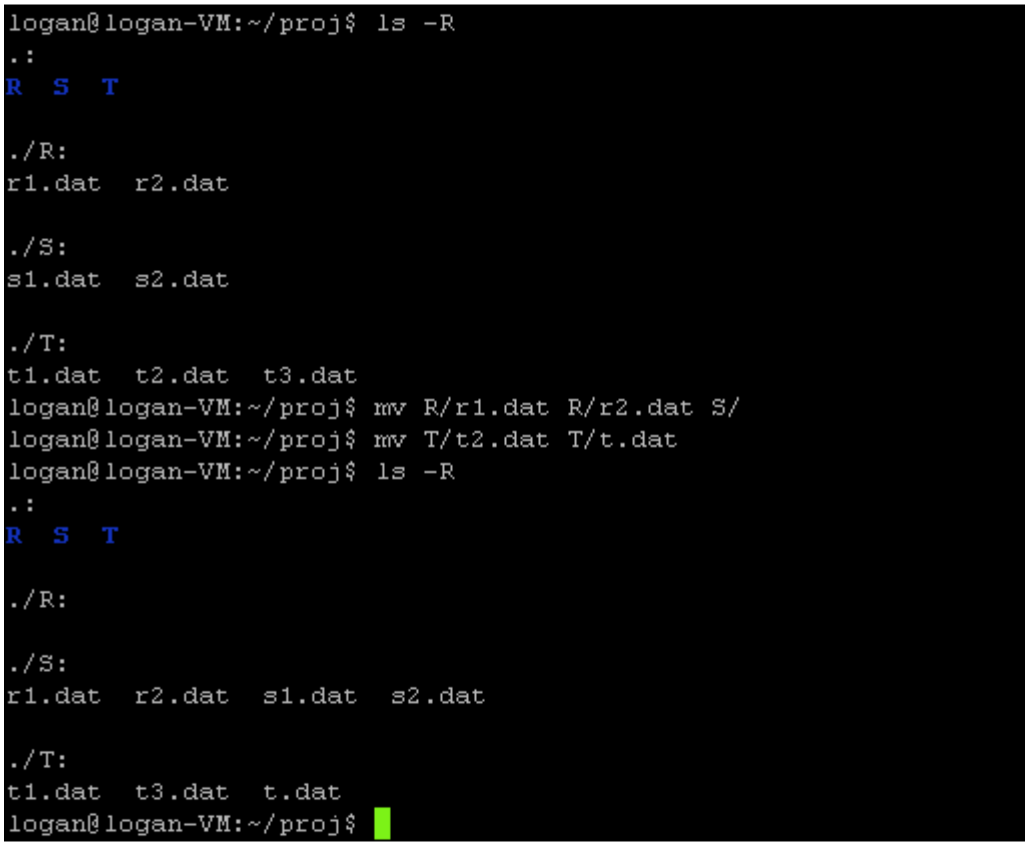


```
logan@logan-VM:~$ cat correspond/todo/personal.txt
1. Have fun
2. Make friends
3. Try to take over the world
logan@logan-VM:~$ ln -s ~/correspond/todo/personal.txt ~/correspond/personal/tod
o
logan@logan-VM:~$ cat correspond/personal/todo
1. Have fun
2. Make friends
3. Try to take over the world
logan@logan-VM:~$
```

In the above example, I output the contents of a file using cat, then I create a symbolic link to that file named “todo” and use cat on the link, thus demonstrating how links may be used.

mv

The mv command is used to take the file (or files) specified, and either rename the file (only with singular), or move the file(s) to a specified path. Example:



```
logan@logan-VM:~/proj$ ls -R
.:
R  S  T

./R:
r1.dat  r2.dat

./S:
s1.dat  s2.dat

./T:
t1.dat  t2.dat  t3.dat
logan@logan-VM:~/proj$ mv R/r1.dat R/r2.dat S/
logan@logan-VM:~/proj$ mv T/t2.dat T/t.dat
logan@logan-VM:~/proj$ ls -R
.:
R  S  T

./R:

./S:
r1.dat  r2.dat  s1.dat  s2.dat

./T:
t1.dat  t3.dat  t.dat
logan@logan-VM:~/proj$ 
```

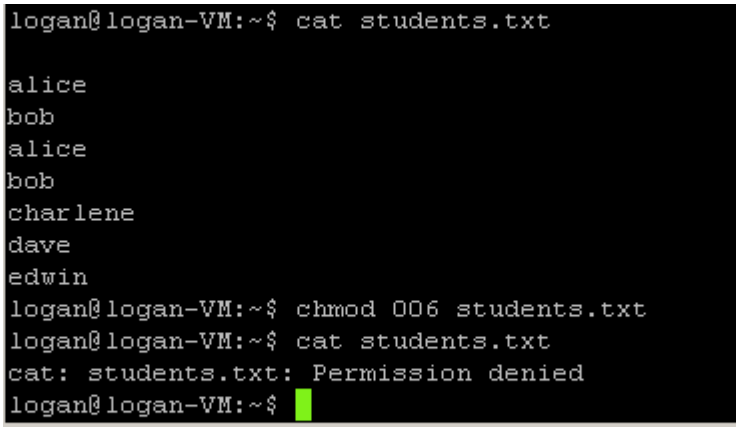
In this example I first moved r1.dat and r2.dat and moved them into the S directory. Then I renamed t2.dat, in the T directory, to just t.dat.

Permissions and chmod

In Linux, there are different users, as well as groups of users, who have permissions to do certain things with certain files. For every file, one user is considered the **owner** of that file—usually the user that created it. The files also have a **group** that they are associated with—usually the group that the owner was in when he created the file. Apart from those to classifications, there's everybody else, or **other.**

For each of these three classifications, every file has set permissions to allow them individually to **read**, **write**, and/or **execute** the files. These permissions can be seen from the first column of the output of the command ls -l seen previously.

The command chmod is used to change basic permissions for files.



```
logan@logan-VM:~$ cat students.txt

alice
bob
alice
bob
charlene
dave
edwin
logan@logan-VM:~$ chmod 006 students.txt
logan@logan-VM:~$ cat students.txt
cat: students.txt: Permission denied
logan@logan-VM:~$
```

As you can see, I entered three digits. The first digit is the permission assigned to the owner, the second digit to the group, and the third to everyone else. The digits are octal which means they're never greater than 7. To figure out what permission one digit gives, simply add up the values; 4 for read, 2 for write, and 1 for execute. For example: 5 = 4 + 1, so 5 means read and execute, but no write. 6 = 4 + 2, so 6 means read and write, but no execute. 0 = no permissions.

todos and fromdos

Linux systems have different formatting for text files than DOS systems. To convert between these two formats for use on their respective systems, the commands todos and fromdos will convert from whatever format Linux uses to whatever format DOS uses and vice versa respectively.

Example:

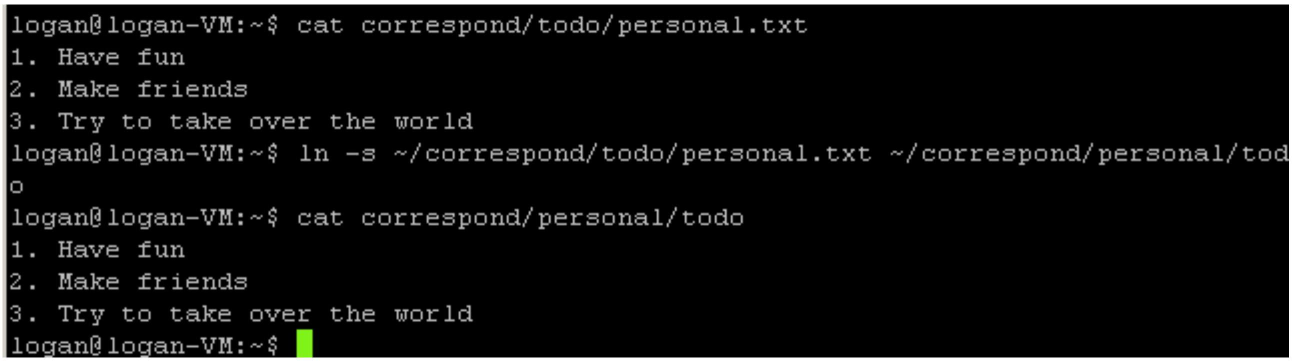
```
todos textfile.txt

fromdos textfile.txt
```

mkdir

mkdir is a command used to create directories.

example:

```
mkdir /home/testuser01/myNewDirectory
/home/testuser01/myOtherNewDirectory
```



```
logan@logan-VM:~$ cat correspond/todo/personal.txt
1. Have fun
2. Make friends
3. Try to take over the world
logan@logan-VM:~$ ln -s ~/correspond/todo/personal.txt ~/correspond/personal/tod
o
logan@logan-VM:~$ cat correspond/personal/todo
1. Have fun
2. Make friends
3. Try to take over the world
logan@logan-VM:~$ 
```

rm

rm is used to remove files and directories.

Example:
```
rm students.txt
```

With the -r option, rm can be used to recursively remove a directory along with all of its contents.

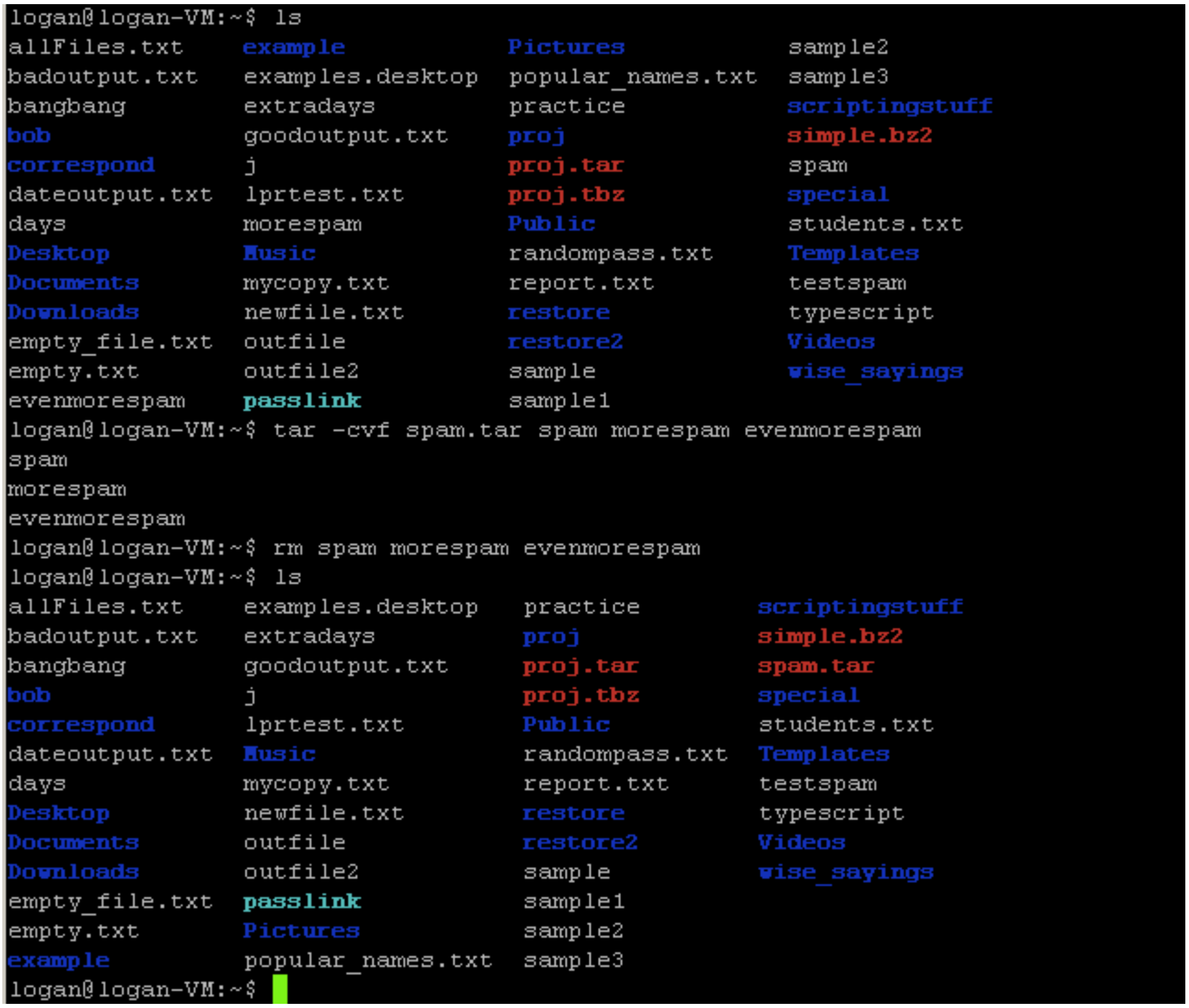
Use it wisely.

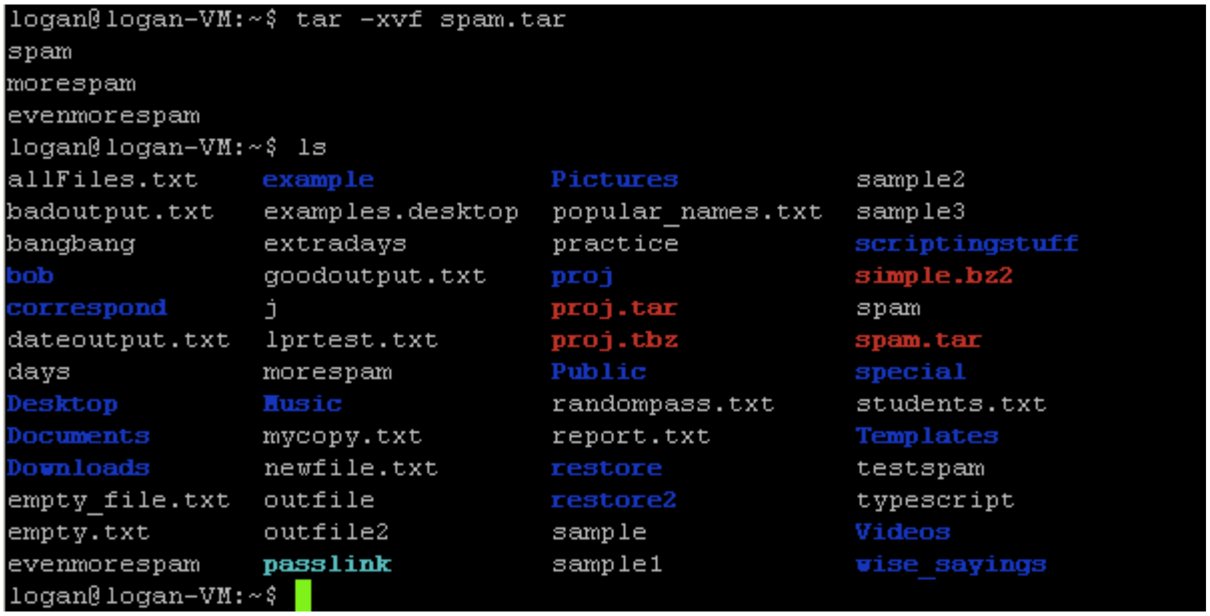
Archiving and Compressing

In this context, archival pretty much refers to the gathering of multiple files all wrapped up into one file called an archive (and I don't mean a directory). Compression is taking a file (which could be an archive) and running it through some super cool algorithm to make it take up less disk space.

tar

The tar command is for archiving files. For creating an archive, it's typical to use the options -cvf. It takes the files specified and puts them in a file whose name is specified by the first argument. It's really easier if I show you:


```
logan@logan-VM:~$ ls
allFiles.txt    example          Pictures          sample2
badoutput.txt   examples.desktop popular_names.txt sample3
bangbang        extradays        practice          scriptingstuff
bob             goodoutput.txt   proj              simple.bz2
correspond      j                proj.tar          spam
dateoutput.txt  lprtest.txt      proj.tbz          special
days            morespam         Public            students.txt
Desktop         Music            randompass.txt    Templates
Documents       mycopy.txt       report.txt        testspam
Downloads       newfile.txt      restore           typescript
empty_file.txt  outfile          restore2          Videos
empty.txt       outfile2         sample            wise_sayings
evenmorespam    passlink         sample1
logan@logan-VM:~$ tar -cvf spam.tar spam morespam evenmorespam
spam
morespam
evenmorespam
logan@logan-VM:~$ rm spam morespam evenmorespam
logan@logan-VM:~$ ls
allFiles.txt    examples.desktop  practice        scriptingstuff
badoutput.txt   extradays         proj            simple.bz2
bangbang        goodoutput.txt    proj.tar        spam.tar
bob             j                 proj.tbz        special
correspond      lprtest.txt       Public          students.txt
dateoutput.txt  Music             randompass.txt  Templates
days            mycopy.txt        report.txt      testspam
Desktop         newfile.txt       restore         typescript
Documents       outfile           restore2        Videos
Downloads       outfile2          sample          wise_sayings
empty_file.txt  passlink          sample1
empty.txt       Pictures          sample2
example         popular_names.txt sample3
logan@logan-VM:~$ 
```
Here I use ls to show you what's in the current directory, then I use tar to gather copies of the files spam, morespam, and evenmorespam into an archive called spam.tar. Then I remove the files with rm, but the archive is still there (in red). Now I can use tar -xvf to extract the files from the archive:

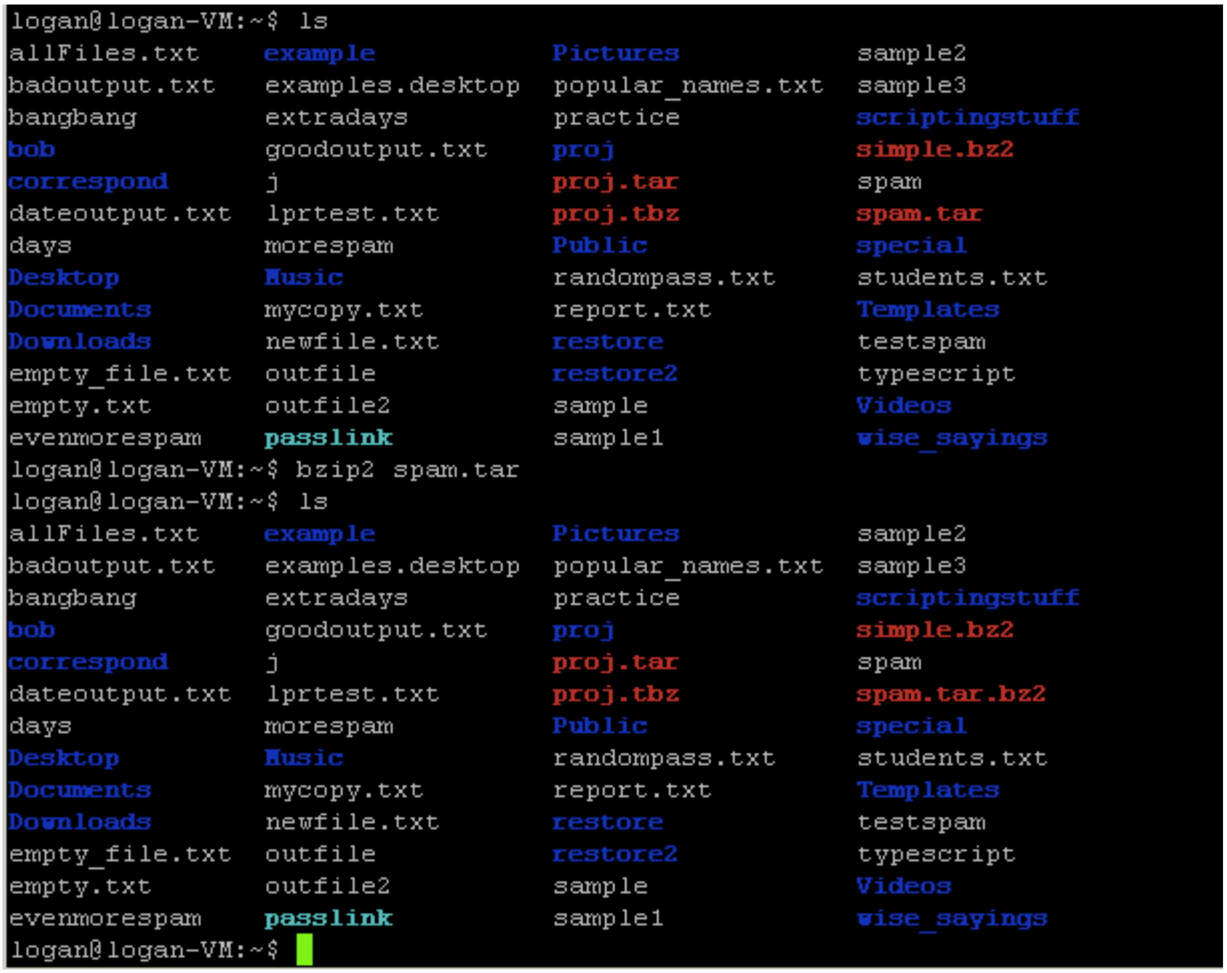


```
logan@logan-VM:~$ tar -xvf spam.tar
spam
morespam
evenmorespam
logan@logan-VM:~$ ls
allFiles.txt     example           Pictures           sample2
badoutput.txt    examples.desktop  popular_names.txt  sample3
bangbang         extradays         practice           scriptingstuff
bob              goodoutput.txt    proj               simple.bz2
correspond       j                 proj.tar           spam
dateoutput.txt   lprtest.txt       proj.tbz           spam.tar
days             morespam          Public             special
Desktop          Music             randompass.txt     students.txt
Documents        mycopy.txt        report.txt         Templates
Downloads        newfile.txt       restore            testspam
empty_file.txt   outfile           restore2           typescript
empty.txt        outfile2          sample             Videos
evenmorespam     passlink          sample1            wise_sayings
logan@logan-VM:~$
```

And there they are again.

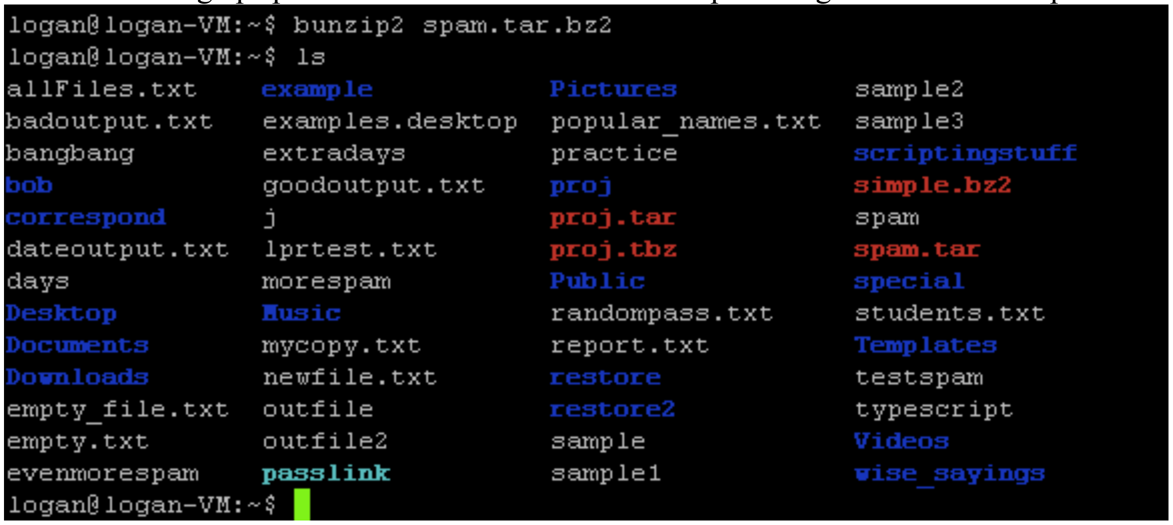
bzip2 and gzip

These two commands are both used for compression. I believe the difference is the algorithms they employ, but the usage is pretty much the same for both, and it's fairly straightforward. Just type the command name, followed by the name of the file you wish to compress.



```
logan@logan-VM:~$ ls
allFiles.txt    example           Pictures           sample2
badoutput.txt   examples.desktop  popular_names.txt  sample3
bangbang        extradays         practice           scriptingstuff
bob             goodoutput.txt    proj               simple.bz2
correspond      j                 proj.tar           spam
dateoutput.txt  lprtest.txt       proj.tbz           spam.tar
days            morespam          Public             special
Desktop         Music             randompass.txt     students.txt
Documents       mycopy.txt        report.txt         Templates
Downloads       newfile.txt       restore            testspam
empty_file.txt  outfile           restore2           typescript
empty.txt       outfile2          sample             Videos
evenmorespam    passlink          sample1            wise_sayings
logan@logan-VM:~$ bzip2 spam.tar
logan@logan-VM:~$ ls
allFiles.txt    example           Pictures           sample2
badoutput.txt   examples.desktop  popular_names.txt  sample3
bangbang        extradays         practice           scriptingstuff
bob             goodoutput.txt    proj               simple.bz2
correspond      j                 proj.tar           spam
dateoutput.txt  lprtest.txt       proj.tbz           spam.tar.bz2
days            morespam          Public             special
Desktop         Music             randompass.txt     students.txt
Documents       mycopy.txt        report.txt         Templates
Downloads       newfile.txt       restore            testspam
empty_file.txt  outfile           restore2           typescript
empty.txt       outfile2          sample             Videos
evenmorespam    passlink          sample1            wise_sayings
logan@logan-VM:~$
```

Here you can see the file spam.tar became spam.tar.bz2 which is now compressed. If I was using gzip, I would enter "gzip spam.tar" and the result would be spam.tar.gz. Now to decompress:

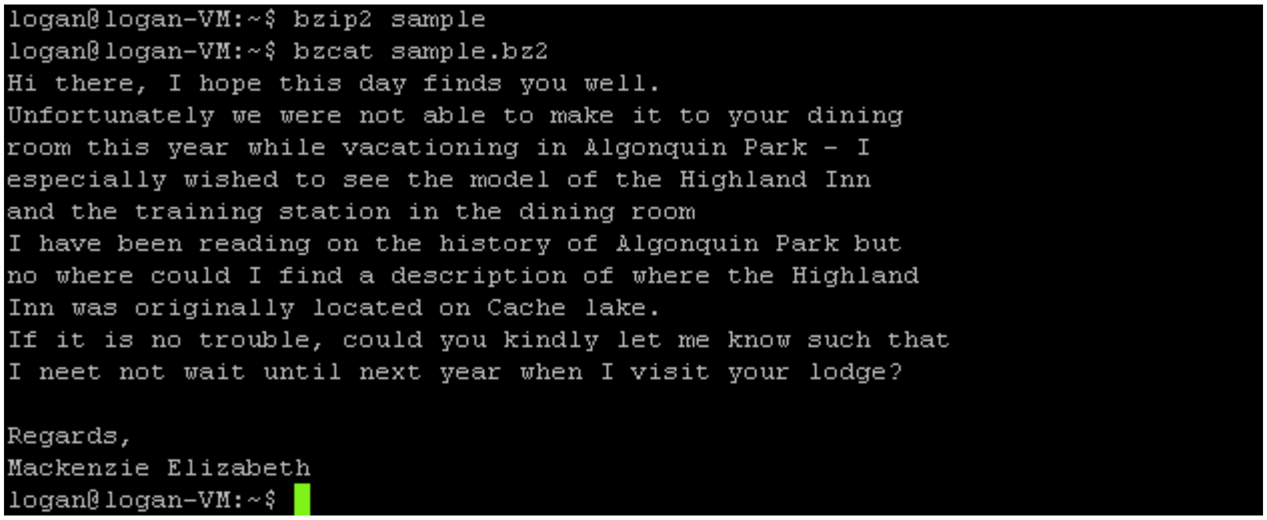


```
logan@logan-VM:~$ bunzip2 spam.tar.bz2
logan@logan-VM:~$ ls
allFiles.txt    example           Pictures           sample2
badoutput.txt   examples.desktop  popular_names.txt  sample3
bangbang        extradays         practice           scriptingstuff
bob             goodoutput.txt    proj               simple.bz2
correspond      j                 proj.tar           spam
dateoutput.txt  lprtest.txt       proj.tbz           spam.tar
days            morespam          Public             special
Desktop         Music             randompass.txt     students.txt
Documents       mycopy.txt        report.txt         Templates
Downloads       newfile.txt       restore            testspam
empty_file.txt  outfile           restore2           typescript
empty.txt       outfile2          sample             Videos
evenmorespam    passlink          sample1            wise_sayings
logan@logan-VM:~$
```

If I had used gzip compression and wanted to decompress the file spam.tar.gz, I would enter "gunzip spam.tar.gz".

bzcat

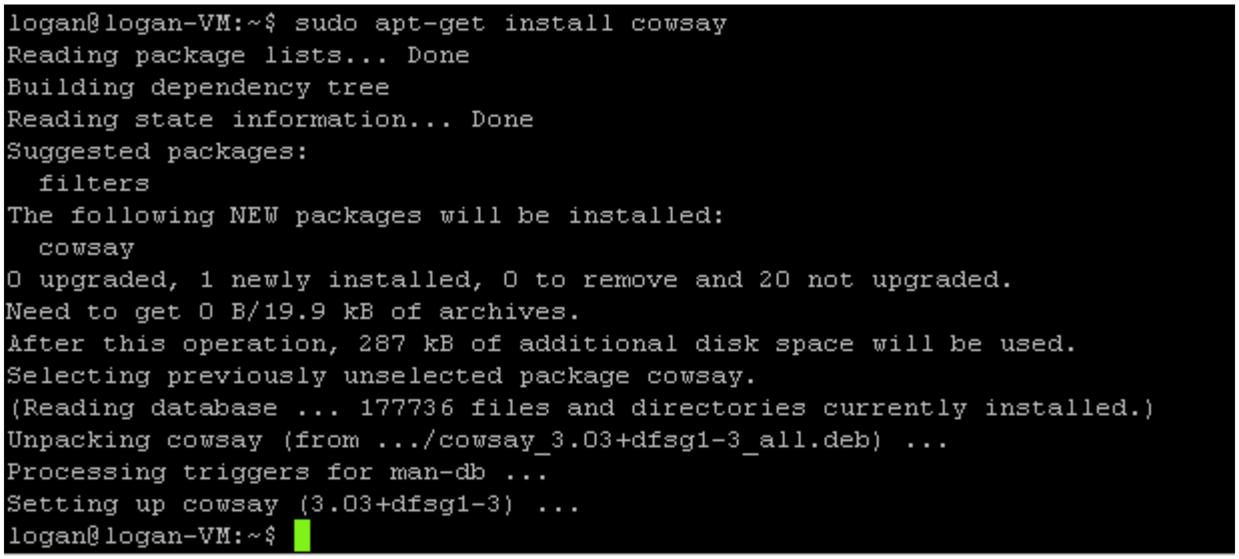
This command is like cat, but it's meant to output the contents of files that have been compressed using bzip2 compression. I'm going to compress the file “sample” and use bzcat on the compressed file to demonstrate further:



```
logan@logan-VM:~$ bzip2 sample
logan@logan-VM:~$ bzcat sample.bz2
Hi there, I hope this day finds you well.
Unfortunately we were not able to make it to your dining
room this year while vacationing in Algonquin Park - I
especially wished to see the model of the Highland Inn
and the training station in the dining room
I have been reading on the history of Algonquin Park but
no where could I find a description of where the Highland
Inn was originally located on Cache lake.
If it is no trouble, could you kindly let me know such that
I neet not wait until next year when I visit your lodge?

Regards,
Mackenzie Elizabeth
logan@logan-VM:~$
```

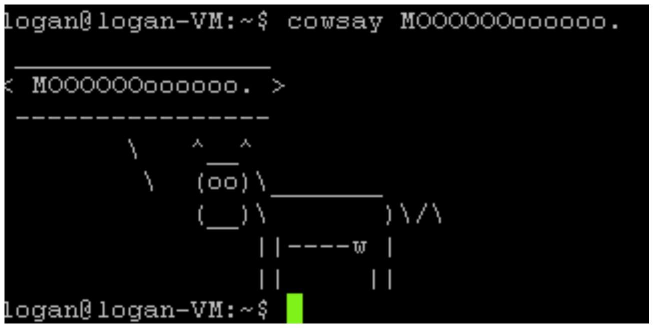
apt-get

The apt-get utility is used for installing new packages like games and stuff. You need to have the right privileges to install anything, so it's typical to use sudo apt-get. There are other features, but you have to specify “install” to actually install the packages. Here's an example of using apt-get to install a silly little program called cowsay:



```
logan@logan-VM:~$ sudo apt-get install cowsay
Reading package lists... Done
Building dependency tree
Reading state information... Done
Suggested packages:
  filters
The following NEW packages will be installed:
  cowsay
0 upgraded, 1 newly installed, 0 to remove and 20 not upgraded.
Need to get 0 B/19.9 kB of archives.
After this operation, 287 kB of additional disk space will be used.
Selecting previously unselected package cowsay.
(Reading database ... 177736 files and directories currently installed.)
Unpacking cowsay (from .../cowsay_3.03+dfsg1-3_all.deb) ...
Processing triggers for man-db ...
Setting up cowsay (3.03+dfsg1-3) ...
logan@logan-VM:~$
```

Let's test it:



```
logan@logan-VM:~$ cowsay MOOOOOOoooooo.
 ________________
< MOOOOOOoooooo. >
 ----------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
logan@logan-VM:~$
```

Printing Stuff to a Printer

Printers are cool. Except they use paper, which might mean killing some trees.

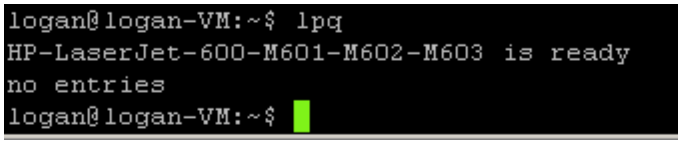
lpr is a command used to send a file as a print job to the default printer configured.

Example:

```
lpr students.txt
```

Whatever's in students.txt, it will be typed on a piece of paper that will be dispensed by your printer. Congratulations.

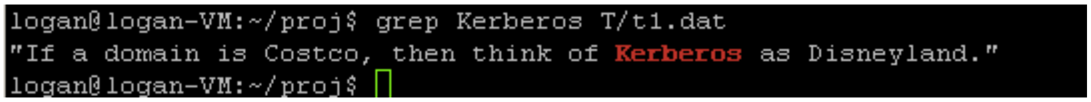
lpq will display the status of the queue for the local printer. Example:



```
logan@logan-VM:~$ lpq
HP-LaserJet-600-M601-M602-M603 is ready
no entries
logan@logan-VM:~$
```

grep

The grep utility is used for searching the contents of a file or the output from another command (see pipes) for a specified text pattern (it used Basic Regular Expressions by default). Without any options, grep's output will be the whole line(s) on which the matching text was found, with the matching part(s) highlighted.



```
logan@logan-VM:~/proj$ grep Kerberos T/t1.dat
"If a domain is Costco, then think of Kerberos as Disneyland."
logan@logan-VM:~/proj$
```

Pipes

Piping is where you take of the stdout of one command and use it as the stdin of a second one. It is done using the pipe (|) symbol in the form `firstcommand | secondcommand.`

Example:

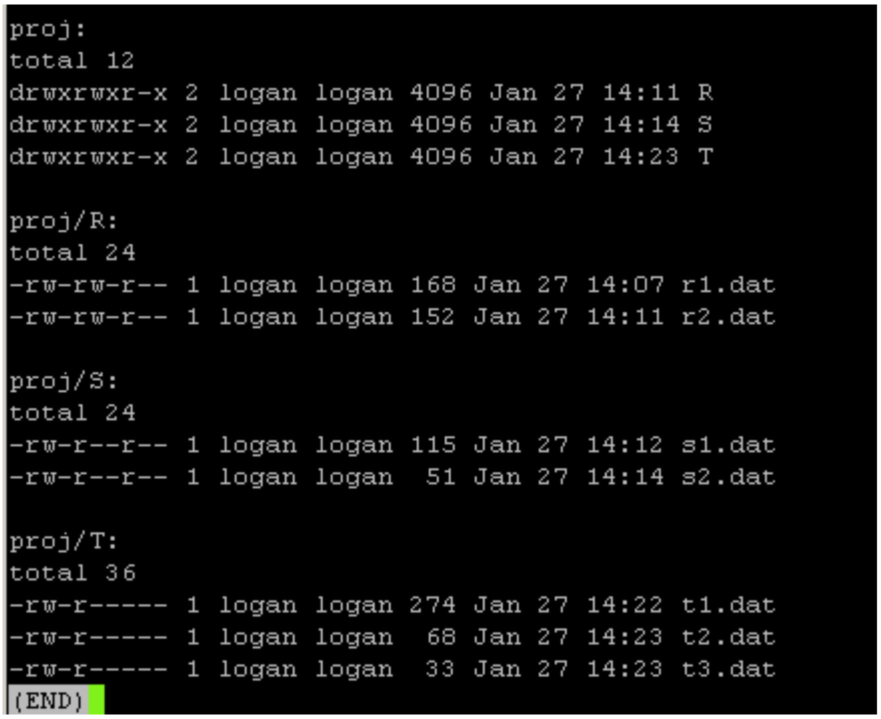
```
cat EULA.TXT | grep responsibility
```

In this example we're taking the stdout from the command "`cat EULA.TXT`", which is essentially the contents of the file EULA.TXT, and piping, or using it **as** the *stdin* of the command "`grep responsibility`", in order to search it for any instances of that word.

Another example:

```
ls -lR proj | less
```

This example takes the stdout from the command "`ls -lR proj | less`", which recursively lists, in long listing format, the contents of the proj directory and all subdirectories, and uses it as the stdin of the `less` command which simply divides the output into scrollable, screen-sized pages. Here's what



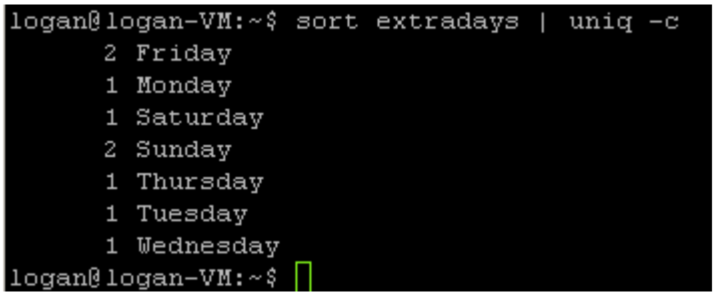
```
proj:
total 12
drwxrwxr-x 2 logan logan 4096 Jan 27 14:11 R
drwxrwxr-x 2 logan logan 4096 Jan 27 14:14 S
drwxrwxr-x 2 logan logan 4096 Jan 27 14:23 T

proj/R:
total 24
-rw-rw-r-- 1 logan logan 168 Jan 27 14:07 r1.dat
-rw-rw-r-- 1 logan logan 152 Jan 27 14:11 r2.dat

proj/S:
total 24
-rw-r--r-- 1 logan logan 115 Jan 27 14:12 s1.dat
-rw-r--r-- 1 logan logan  51 Jan 27 14:14 s2.dat

proj/T:
total 36
-rw-r----- 1 logan logan 274 Jan 27 14:22 t1.dat
-rw-r----- 1 logan logan  68 Jan 27 14:23 t2.dat
-rw-r----- 1 logan logan  33 Jan 27 14:23 t3.dat
(END)
```

the output looks like:

One more example:

```
sort extradays | uniq -c
```

This takes the output from "`sort extradays`", and uses it as the input for "`uniq -c`". Effectively, the lines of contents of the file extradays are sorted in alphabetical order by the sort command, and then any occurences of duplicate lines that are directly above another will be removed by the uniq command and counted (because of the -c option).



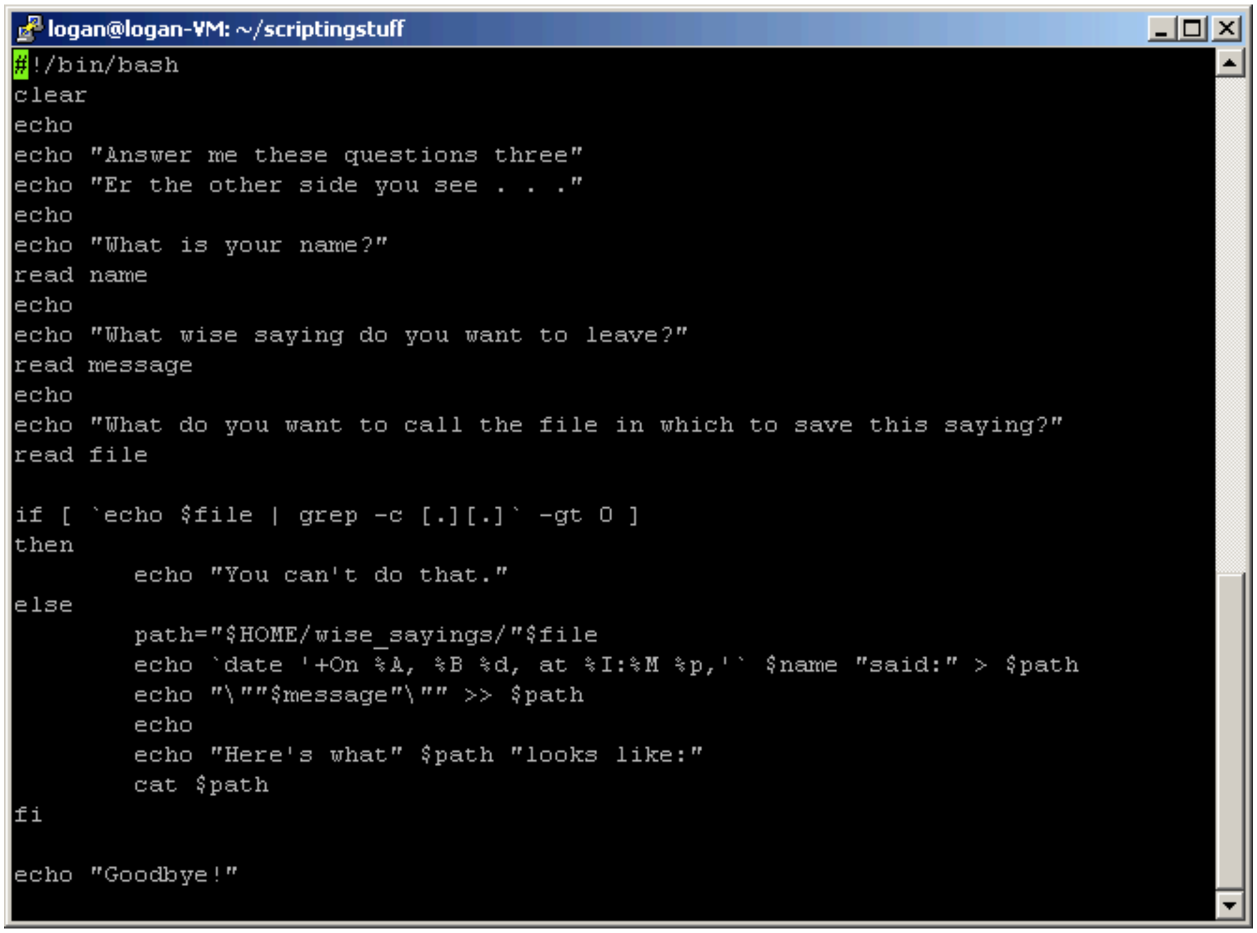
```
logan@logan-VM:~$ sort extradays | uniq -c
      2 Friday
      1 Monday
      1 Saturday
      2 Sunday
      1 Thursday
      1 Tuesday
      1 Wednesday
logan@logan-VM:~$
```

Scripting

Scripting is not as hard as it may sound at first. In simple terms, scripting is taking a series of commands and recording them in a file to be executed together. And that's all scripts are: files. Normally I make them using vi, but whatever text editor you're comfy with will suffice just fine.

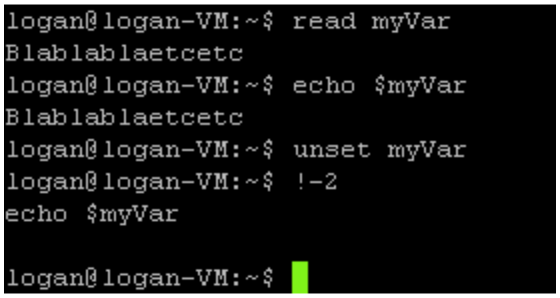
There are different kinds of scripts, but I'm only talking about BASH scripts for now. The first line in all bash scripts is “#!/bin/bash”, and it tells Linux where to look for the program that will be used to execute the lines that come after; in this case, the program is BASH itself.

After that, each line is a command like any of the commands you learned previously. The thing to note is that you will be using this script *file* to execute all of those commands together in the specific sequence in which they are written from one line to the next. There are also ways (if-statements) to make it skip lines depending on what happens before it gets to those lines, as well as ways (loops) to make it repeat certain lines until something changes that tells it to move on. Here is a somewhat complicated example of a script:



```
#!/bin/bash
clear
echo
echo "Answer me these questions three"
echo "Er the other side you see . . ."
echo
echo "What is your name?"
read name
echo
echo "What wise saying do you want to leave?"
read message
echo
echo "What do you want to call the file in which to save this saying?"
read file

if [ `echo $file | grep -c [.][.]` -gt 0 ]
then
        echo "You can't do that."
else
        path="$HOME/wise_sayings/"$file
        echo `date '+On %A, %B %d, at %I:%M %p,'` $name "said:" > $path
        echo "\""$message"\"" >> $path
        echo
        echo "Here's what" $path "looks like:"
        cat $path
fi

echo "Goodbye!"
```

The read command (first seen on the eighth line of the script), once executed, waits for the user to type something and press the enter key. It will take whatever they entered, and store it in the variable whose name was specified (in this example that would be “name”). Then, in later parts of the script, that variable can be used again to do other things, including answer simple questions, and perform math operations. If you want to clear a variable (make it empty), you can use the command **unset** followed by the variable name:



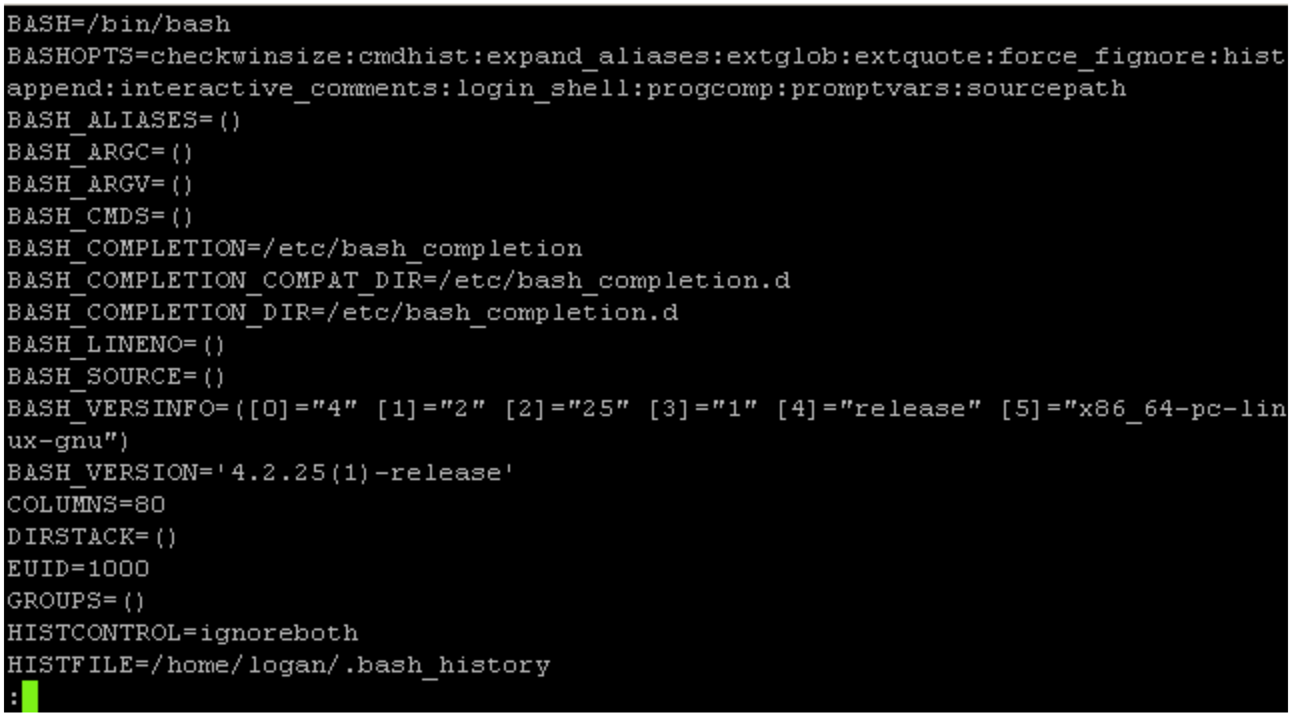
```
logan@logan-VM:~$ read myVar
Blablablaetcetc
logan@logan-VM:~$ echo $myVar
Blablablaetcetc
logan@logan-VM:~$ unset myVar
logan@logan-VM:~$ !-2
echo $myVar

logan@logan-VM:~$
```

So! Once you've save the script file, the first thing you'll probably want to do is test it. How do you do that? If you want to **execute** a file, well, first you need permission. So we'll use chmod to do the trick.

Once you have execute permission, you can execute it, but the catch is that you can't use a relative path. What you *can* do is, if the script is in your pwd, then enter "./myscript.sh" and the '.' will be recognized as the absolute path to the file. That's what the '.' means: the current directory.

set

The set command displays a practically-impossible-to-read list of environment variables.

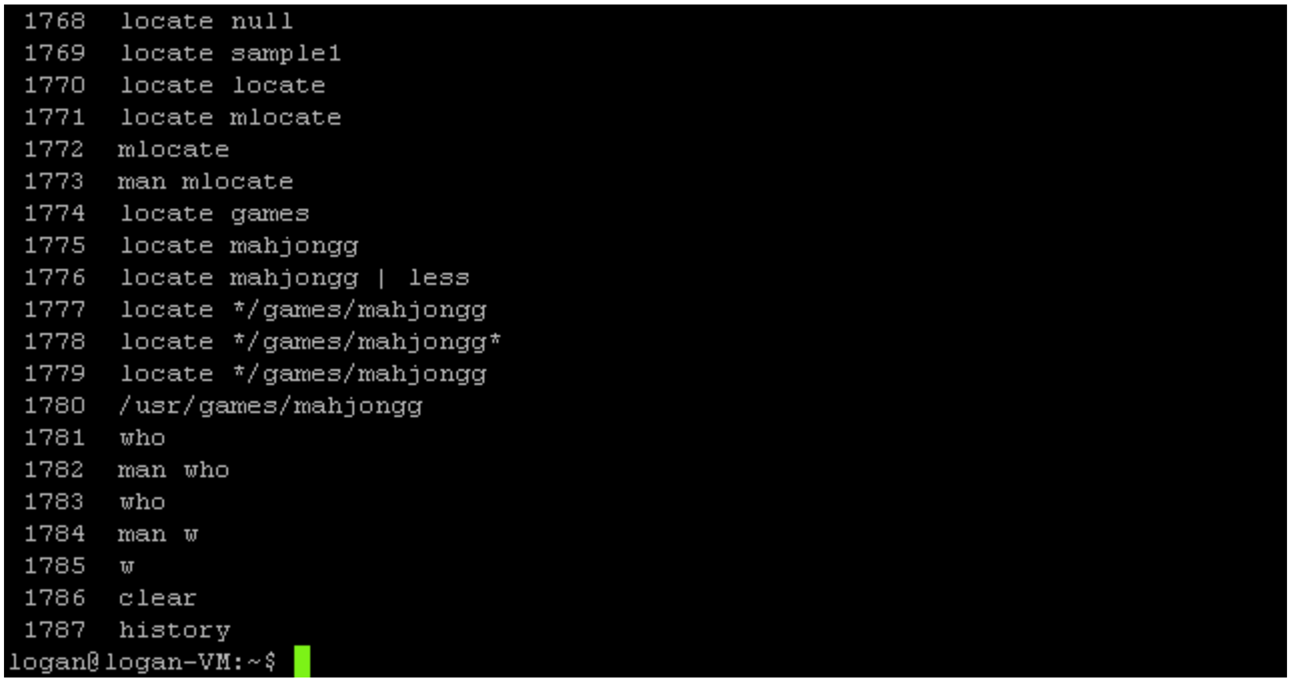


```
BASH=/bin/bash
BASHOPTS=checkwinsize:cmdhist:expand_aliases:extglob:extquote:force_fignore:hist
append:interactive_comments:login_shell:progcomp:promptvars:sourcepath
BASH_ALIASES=()
BASH_ARGC=()
BASH_ARGV=()
BASH_CMDS=()
BASH_COMPLETION=/etc/bash_completion
BASH_COMPLETION_COMPAT_DIR=/etc/bash_completion.d
BASH_COMPLETION_DIR=/etc/bash_completion.d
BASH_LINENO=()
BASH_SOURCE=()
BASH_VERSINFO=([0]="4" [1]="2" [2]="25" [3]="1" [4]="release" [5]="x86_64-pc-lin
ux-gnu")
BASH_VERSION='4.2.25(1)-release'
COLUMNS=80
DIRSTACK=()
EUID=1000
GROUPS=()
HISTCONTROL=ignoreboth
HISTFILE=/home/logan/.bash_history
:
```

This is the output of "set | less". If you were looking for the value of a particular variable, you can always pipe it to grep.

history

The history command is great for viewing commands you previously entered.

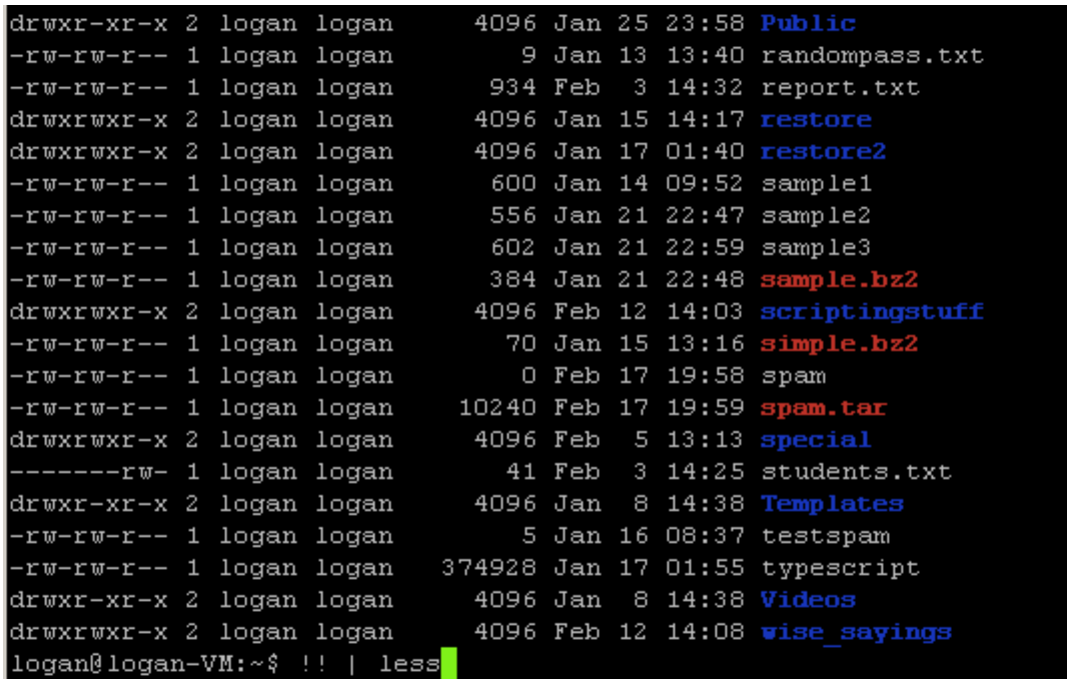


```
 1768  locate null
 1769  locate sample1
 1770  locate locate
 1771  locate mlocate
 1772  mlocate
 1773  man mlocate
 1774  locate games
 1775  locate mahjongg
 1776  locate mahjongg | less
 1777  locate */games/mahjongg
 1778  locate */games/mahjongg*
 1779  locate */games/mahjongg
 1780  /usr/games/mahjongg
 1781  who
 1782  man who
 1783  who
 1784  man w
 1785  w
 1786  clear
 1787  history
logan@logan-VM:~$ 
```

As you can see, it pretty much keeps track of everything. If you wanted to go way back, you can just pipe the command into less. 1787 isn't so bad. You can see I use man practically all the time because I don't have the best memory.

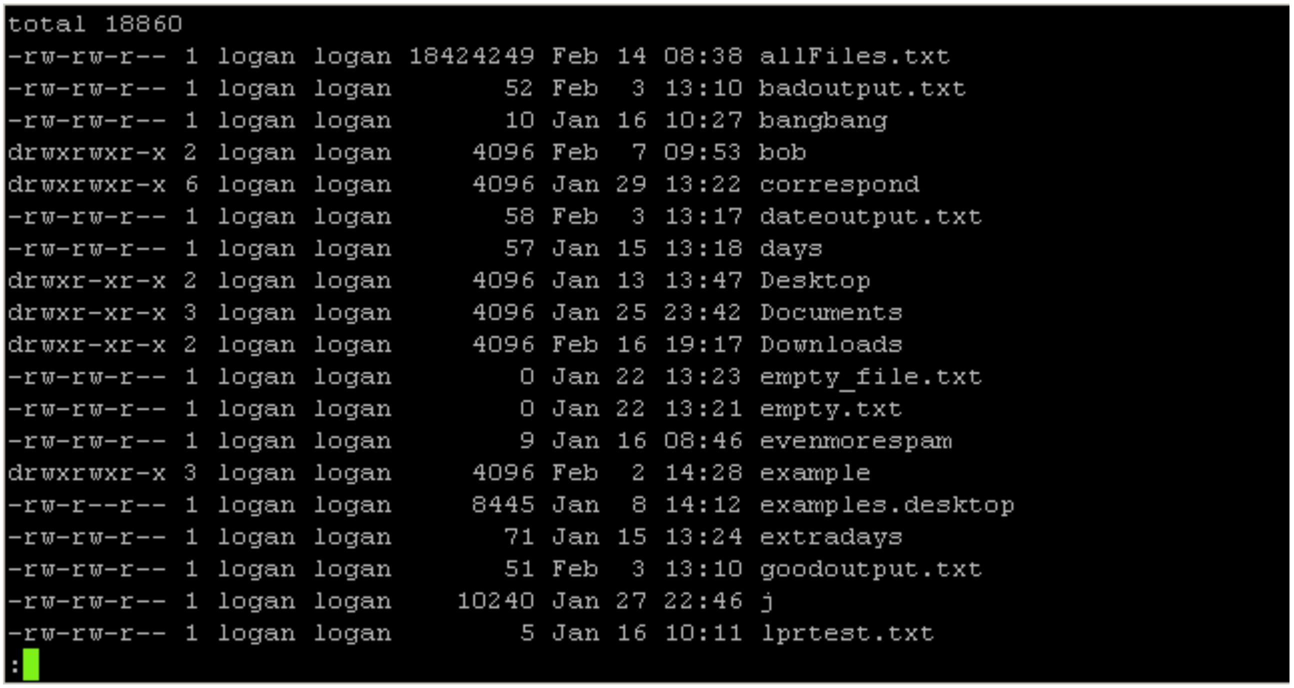
Here's something cool to know: you know how you can use ~ to mean your home, '.' for the current, and '..' for the parent directory so you can use those special characters instead of having to use absolute paths all the time? If you look in the man page for history, it tells you about a couple shortcut characters that can be used to quickly re-type previously entered commands. If you type part of a command and accidentally hit enter before typing the whole thing, you can use "!!" to mean exactly what you just entered, and then just finish typing (i.e. "!!therest of thecommand"). Or if you just realized you wanted to pipe the output to less, the same still applies.

For example:



```
drwxr-xr-x 2 logan logan     4096 Jan 25 23:58 Public
-rw-rw-r-- 1 logan logan        9 Jan 13 13:40 randompass.txt
-rw-rw-r-- 1 logan logan      934 Feb  3 14:32 report.txt
drwxrwxr-x 2 logan logan     4096 Jan 15 14:17 restore
drwxrwxr-x 2 logan logan     4096 Jan 17 01:40 restore2
-rw-rw-r-- 1 logan logan      600 Jan 14 09:52 sample1
-rw-rw-r-- 1 logan logan      556 Jan 21 22:47 sample2
-rw-rw-r-- 1 logan logan      602 Jan 21 22:59 sample3
-rw-rw-r-- 1 logan logan      384 Jan 21 22:48 sample.bz2
drwxrwxr-x 2 logan logan     4096 Feb 12 14:03 scriptingstuff
-rw-rw-r-- 1 logan logan       70 Jan 15 13:16 simple.bz2
-rw-rw-r-- 1 logan logan        0 Feb 17 19:58 spam
-rw-rw-r-- 1 logan logan    10240 Feb 17 19:59 spam.tar
drwxrwxr-x 2 logan logan     4096 Feb  5 13:13 special
-------rw- 1 logan logan       41 Feb  3 14:25 students.txt
drwxr-xr-x 2 logan logan     4096 Jan  8 14:38 Templates
-rw-rw-r-- 1 logan logan        5 Jan 16 08:37 testspam
-rw-rw-r-- 1 logan logan   374928 Jan 17 01:55 typescript
drwxr-xr-x 2 logan logan     4096 Jan  8 14:38 Videos
drwxrwxr-x 2 logan logan     4096 Feb 12 14:08 wise_sayings
logan@logan-VM:~$ !! | less
```

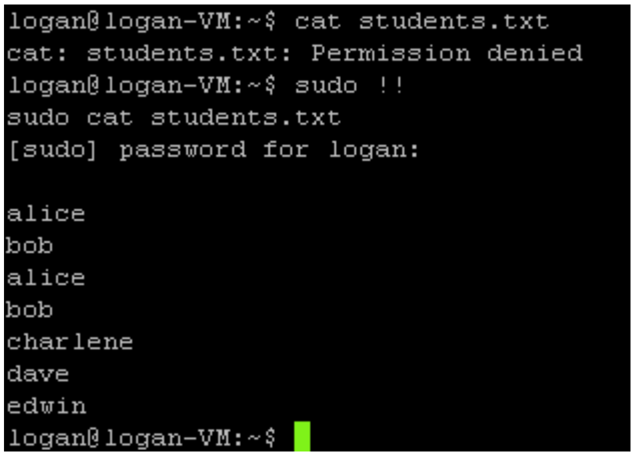
Here I just entered “ls -l” and am about to enter “!! | less”:



```
total 18860
-rw-rw-r-- 1 logan logan 18424249 Feb 14 08:38 allFiles.txt
-rw-rw-r-- 1 logan logan       52 Feb  3 13:10 badoutput.txt
-rw-rw-r-- 1 logan logan       10 Jan 16 10:27 bangbang
drwxrwxr-x 2 logan logan     4096 Feb  7 09:53 bob
drwxrwxr-x 6 logan logan     4096 Jan 29 13:22 correspond
-rw-rw-r-- 1 logan logan       58 Feb  3 13:17 dateoutput.txt
-rw-rw-r-- 1 logan logan       57 Jan 15 13:18 days
drwxr-xr-x 2 logan logan     4096 Jan 13 13:47 Desktop
drwxr-xr-x 3 logan logan     4096 Jan 25 23:42 Documents
drwxr-xr-x 2 logan logan     4096 Feb 16 19:17 Downloads
-rw-rw-r-- 1 logan logan        0 Jan 22 13:23 empty_file.txt
-rw-rw-r-- 1 logan logan        0 Jan 22 13:21 empty.txt
-rw-rw-r-- 1 logan logan        9 Jan 16 08:46 evenmorespam
drwxrwxr-x 3 logan logan     4096 Feb  2 14:28 example
-rw-r--r-- 1 logan logan     8445 Jan  8 14:12 examples.desktop
-rw-rw-r-- 1 logan logan       71 Jan 15 13:24 extradays
-rw-rw-r-- 1 logan logan       51 Feb  3 13:10 goodoutput.txt
-rw-rw-r-- 1 logan logan    10240 Jan 27 22:46 j
-rw-rw-r-- 1 logan logan        5 Jan 16 10:11 lprtest.txt
:
```

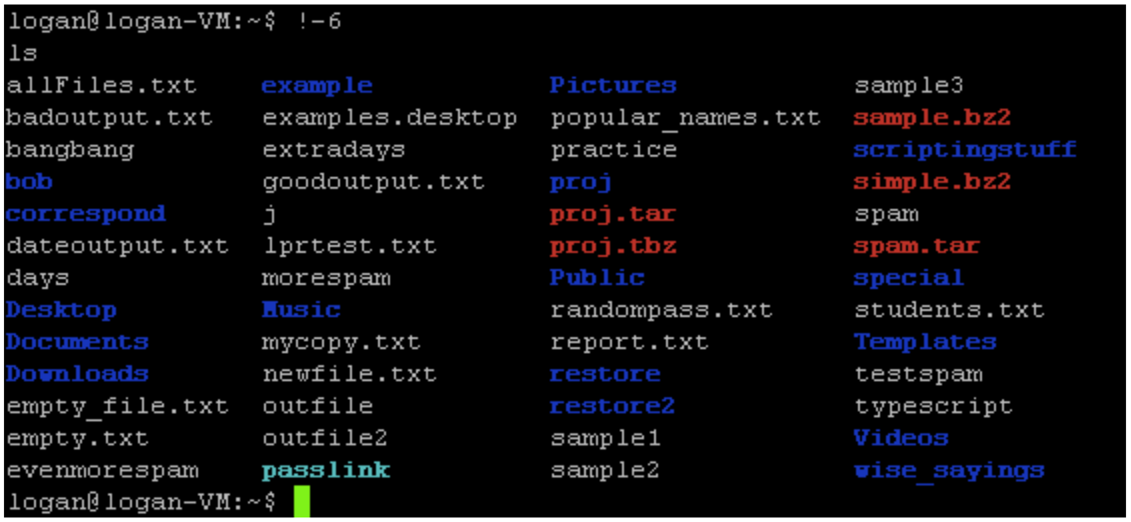
It worked!

It also works for adding to the beginning of a command, like if you forgot to use sudo:



```
logan@logan-VM:~$ cat students.txt
cat: students.txt: Permission denied
logan@logan-VM:~$ sudo !!
sudo cat students.txt
[sudo] password for logan:

alice
bob
alice
bob
charlene
dave
edwin
logan@logan-VM:~$
```

Now say you wanted to enter the same command you entered n commands ago. Easy! Just type !-n, replacing 'n' with the number of commands ago the one you want was entered:



```
logan@logan-VM:~$ !-6
ls
allFiles.txt     example           Pictures           sample3
badoutput.txt    examples.desktop  popular_names.txt  sample.bz2
bangbang         extradays         practice           scriptingstuff
bob              goodoutput.txt    proj               simple.bz2
correspond       j                 proj.tar           spam
dateoutput.txt   lprtest.txt       proj.tbz           spam.tar
days             morespam          Public             special
Desktop          Music             randompass.txt     students.txt
Documents        mycopy.txt        report.txt         Templates
Downloads        newfile.txt       restore            testspam
empty_file.txt   outfile           restore2           typescript
empty.txt        outfile2          sample1            Videos
evenmorespam     passlink          sample2            wise_sayings
logan@logan-VM:~$
```

So I entered “ls” 6 commands ago. Neat huh?

In Conclusion

My final word of advice will be that when it comes to learning the in-depth details of commands and their various options, ***in the long run*** it is more practical to read the man pages and apply them to memory than to permanently depend on a cheat sheet or guide such as this; all though having a reference for a quick refresh is always useful if you simply haven't used a certain command in a long time. But, at that point, it would be recommended that you start to make your own notes so that they can be personalized to your style and preference.