The Linux Command Line Notes

Jordan Mandel

September 7, 2022

Ch. 6. Redirection

`ls -l /bin/usr 2> ls-error.txt` `ls -l /bing/usr > ls-output.txrt 2>&1` `ls -l /bin/usr &> ls-output.txt` `ls -l /bin/usr &>> ls-output.txt` `ls /usr/bin | tee ls.txt | grep zip`

Ch. 7. Seeing the world as the shell sees it

Expansion

we had pathname: `ls .[!.]* echo .[!.]*`

tilde, `ls ~`

arithmetic `echo $(($((5**2)) * 3))` Can group with parentheses eliminating need for inner expression `echo $(((5**2) * 3))`

brace . . , comma separated lists, cartesian products when multiple or nested brace expansions are used), parameter, and command expansion.

`echo Front-{A,B,C}-Back` `echo {001..15}` `echo a{A{1,2},B{3,4}}b` `mkdir {2007..2009}-{01..12}`

parameter `echo $USER`

can get variables with `printenv | less`

command ehco $USER `ls -l $(which cp)` can also use backtics for command subst5itution

Quoting

Double Quotes

- word splitting (suppression of extra spaces/new lines), pathname expansion (ie with wildcards), tilde expansion, and brace expansion are suppressed;
  - none of these use the dollar sign
  - Can escape the dollar sign in double quotes with a backslash
  - can also use backtick
- but we can do command substitution (which itself can have expansion), arithmetic expansion, and parameter epansion
  - all of these use the dollar sign followed by parentheses
  - Can escape the dollar sign in double quotes with a backslash

note the interesting example of `echo` not outputting some intended line breaks/etc due to word splitting. get around this with double quotes. example of this is `echo $(cal)` vs `echo "$(cal)"`

Single Quotes Single quotes suppress all expansions.

backslash Can use it to escape characters, including special characters in file names.

for example:

`sleep 10; echo -e "Time's up\a"`

We could also do this:

`sleep 10; echo "Time's up" $'\a'`

Ch. 15. Storage Media

there is the `mount` command to see the mounted systems

look in `\etc\fstab`

`journalctl -f`, it can also be `sudo tail -f /var/log/messages` on some systems.

`ls /dev`

codes are: `fd*` for floppy disks. `hd*` for motherboard storage; alternating m*ster and sl*ve. Number for partition.

`lp*` for printer

`sd*` for storage devices like usb drives.

`sr*` for optical drives. can also see symlinks to device files

there is also `lsblk`

`Bus 001 Device 004: ID 1e4e:0102 Cubeternet GL-UPC822 UVC WebCam`

`fuser /dev/video0` corresponds to `/dev/bus/usb/001/004`

there is `mount -t iso9660 /dev/sdc /mnt/cdrom` to actually mount something

ther is also `lsof` and `fuser`

Creating File Systems

```
sudo umount /dev/sdb1
sudo fdisk /dev/sdb
```

then there are several options

then use `mkfs` to make a new file system. for example `sudo mkfs -t ext4 /dev/sdb1`

`fsck` can check/repair a file system

`dd` can do a direct copy of data between media.

can be used for `iso` files.

there is also `genisoimage` which can be used to generate iso image from files gathered into a directory.

can mount iso images that are still onyour hard drive:

`mount -t iso9660 -o loop image.iso /mnt/iso_image`

there is also `wodim` to blank, and to write too.

there is the `dd` command `genisoimage` for creating something from a collection of files

Ch 16. Networking

`ping nsa.gov`

these trace packets across networks

```
traceroute
traceroute -TI
```

```
tracepath
```

internal network configuration to look at network configuration:

```
ip addr
```

to look at network configurations

```
# like ifconfig
netstat -ie
# show kernel internal network settings
netstat -r
```

transporting files across a network

ftp login to server (not secure). `lcd` changes directory on local machine. `get [file]` will start the file transfer. can type `help` to get more info.

lftp better than ftp; use this; but should be using sftp in general anyway.

wget

secure communication

ssh authenticates remote server, encrypts all data sent/received runs on port (have to learn about ports) 22.

more flexible than ftp or lftp

```
ssh user@remote_host
```

if there is a prloblem it will tell us what line the `~/.ssh/known_hosts` file has the outdated key

can run a single command

```
ssh remote-sys 'ls *' > dirlist.txt
```

to get the x system from remote server

```
ssh -X remote-sys #either X or Y depending on the system
ssh -Y remote-sys
```

scp and sftp

```
scp remote-sys:document.txt .
```

or if the remote hostname is different than the local hostname:

```
scp bob@remote-sys:document.txt .
```

```
sftp remote-sys
ls
lcd Desktop
get file.txt
bye
```

there is puTTY for windows but it might have something built-in now.

more reading: `Linus Network Administrator's Guide`

wikipedia articles for ip addresses, host names, URIs.

Ch. 17. Searching For Files

easy way with `locate`

locate finds things the easy way:

```
locate bin/zip
```

```
locate zip | grep bin
```

but I might have to do `sudo updatedb`. Might have to set a cron job.

hard way with `find`

```
find ~ -type d | wc -l
```

Tests

File	Type Description
b	Block special device file
c	Character special device file
d	Directory
f	Regular file
l	Symbolic link

```
find ~ -type f -name "*.JPG" -size +1M | wc -l
```

The plus sign means 'more than'. Minus sign means less than Available sizes are:

```
b 512-byte blocks. This is the default if no unit is specified.
c Bytes.
w 2-byte words.
k Kilobytes (units of 1024 bytes).
M Megabytes (units of 1048576 bytes).
G Gigabytes (units of 1073741824 bytes).
```

here are some more options (can use cmin in man page to find the rest)

```
-cmin n
Match files or directories whose content or attributes were
last modified exactly n minutes ago. To specify less than n
minutes ago, use -n, and to specify more than n minutes
ago, use +n.
```

Numeric arguments above can take `+` and `-`.

operators can use `-and` `-or` `-not` and escaped parentheses.

for example:

```
find ~ \( -type f -not -perm 0600 \) -or \( -type d -not -perm 0700 \)
```

does logical short-circuiting

predifiened actions can perform actions on the found files: `-delete -ls -print -quit` (more in man pages): print is used if nothing is specified

so `find ~` is the same as `find ~ -print`

we can also go

```
find ~ -type f -name '*.bak' -delete
```

the `and` is implied

the logical operators can be used to control the actions;

```
find ~ -type f -and -name '*.bak' -and -print
```

if we put the print-first it would be different; it would print before doing the tests.

user-defined actions instead of `-delete` we can go: `-exec rm '{}' ';'` Have to quote. Semicolon is necessary delimeter. Braces represent the filepath of the file found. Can use `-ok` rather than `-exec` to get confirmation for each action.

for example: `find ~ -type f -name 'foo*' -ok ls -l '{}' ';'`

impriving efficiency `-exec` uses a new instance of each command for each file found. we can use `xargs` or a certain feature of `find` itself to get thorugh this.

the `find` way replace `';'` with `+` and it will execute on each file.

the `xargs` way

```
find ~ -type f -name `foo*` -print | xargs ls -l
```

if too many arguments are given it will start over with next argument after reaching end.

To deal with funny file names we can use the `-print0` which separates with null characters and the `--null` argument to xargs.

```
find ~ -iname '*.jpg' -print0 | xargs --null ls -l
```

return to playground

```
mkdir -p playground/dir{001..1000}
touch playground/dir-{001..100}/file-{A-Z}
find playground -type f -name 'file-A'
[me@linuxbox ~]$ find playground -type f -name 'file-B' -exec touch
find playground -type f -newer playground/timestamp

[me@linuxbox ~]$ find playground \( -type f -not -perm 0600 \) -or \( -type d -not -perm 0700 \)

[me@linuxbox ~]$ find playground \( -type f -not -perm 0600 -exec
chmod 0600 '{}' ';' \) -or \( -type d -not -perm 0700 -exec chmod 0700 '{}' ';' \)
```

there are more options: `-depth` does depth first actions. `-maxdepth`, `-mindepth`, `-mount` (dont traverse directories on other file systems), `-noleaf` better to to use when using DOS -like file systems. #### Other options

`-depth`: process files before directory itself

`-maxdepth`, `-mindepth`: how deep to go before performing tests and actions `-mount` dont go doun mounted file systems `-noleaf` don't perform optimizations based on the assumption that it is a unix file system

18. - Archiving and Backup

Compressing Files

`gzip`, `gunzip` (and associated options) `zcat`, `zless`

`bzip2`

`tar`

`rsync`

can just go

compressing

```
Option
Long Option
Description
-c
--stdout
--to-stdout
Write output to standard output and keep the
original files.

-d
--decompress
--uncompress
Decompress. This causes gzip to act like
gunzip.

-f
--force
Force compression even if a compressed
version of the original file already exists.

-h
--help
Display usage information.

-l
--list
List compression statistics for each file
compressed.

-r
--recursive
If one or more arguments on the command line
is a directory, recursively compress files
contained within them.

-t
--test
Test the integrity of a compressed file.

-v
--verbose
```

`Display verbose messages while compressing.`
`-number`

Set amount of compression. number is an integer in the range of 1 (fastest, least compression) to 9 (slowest, most compression). The values 1 and 9 may also be `gunzip foo.txt` leaving of the `.gz` because it is assumed.

`gzip` can use stdio

`zcat` uses cat on the zipped file. there is also `zless`

there is also `bzip` which is better. `bunzip` `bzcat`

`bz2recover` for damaged bz2 files.

archiving

remembering them for long term storage.

`tar c path`

these are modes, there are also options

c is for create x is for extract r is for append t is for list

[make playground]

```
tar cf playground.tar playground

tar tf playgorund.tar
tar tvf playground.tar # verbose
```

make directory, change into it

`tar xf ../playground.tar`

ownership is transferred to decompressor

funny pathnames

```
mkdir foo
cd ~
tar cf playground2.tar ~/playground
cd foo
tar xf ../playground2.tar
ls
```

output is `home` because of how the pathnames work.

can go

`tar xf archive.tar pathname`

or

`tar xf ../playground2.tar --wildards 'home/me/playground/dir-*/file-A'`

only gets the specified files. the latter wildcards only work for gnu TAR.

can use it with `find`

`[me@linuxbox ~]$ find playground -name 'file-A' -exec tar rf playground.tar '{}' '+'`

recall that `+` makes it run only once.

can also make incremental backups; newer than last tar in append mode; look up more later?; good online documentation for this

can use stdin/stdout

```
[me@linuxbox ~]$ find playground -name 'file-A' | tar cf - --files-
from=- | gzip > playground.tgz
```

- means stdin/out

can shorten it to this:

```
[me@linuxbox ~]$ find playground -name 'file-A' | tar czf
playground.tgz -T -
```

If we had wanted to create a bzip2-compressed archive instead, we could have done this:

```
[me@linuxbox ~]$ find playground -name 'file-A' | tar cjf
playground.tbz -T -
```

z for gzip.

tar over ssh

```
[me@linuxbox ~]$ mkdir remote-stuff
[me@linuxbox ~]$ cd remote-stuff
[me@linuxbox remote-stuff]$ ssh remote-sys 'tar cf - Documents' | tar
xf -
me@remote-sys's password:
[me@linuxbox remote-stuff]$ ls
Documents
```

f sends tar to stdout, sx is for expand mode

zip

```
zip -r playground.zip playground # have to do -r to get contents
```

the zip program will update archives rather than replacing them

the `-l` option just lists.

`unzip -l` function just lists

can specify files

```
unzip -l ../playground.zip playground/dir-087/file-z
```

`-v` makes it more verbose.

can use stdio: in this case it was the `-@` option that makes it take a list of file names

```
find playground -name "file-A" | zip -@ file-A.zip
```

can write to stdout but it's not that great. unzip doesn't accept stdin

can do more normal stdin:

```
ls -l /etc/ | zip ls-etc.zip -
```

unzip can be sent to stdout though

```
unzip -p ls-etc.zip | less
```

zip is mostly used for window systems

synchronizing files and directories

`rsync options source destination`

- local: local file
- remote: `[user@]host:path`
- rsync server: `rsync://[user@]host[:port]/path`

one must be local

`-a` is for archiving; recursion and preservation of file attributes to make a mirror of the playground directory
`-v` is for verbose output

`rsync -av playground foo`

it only does as much work as it needs to.

`rsync source destination`

copies source into destination

`rsync source/ destination`

copies contents of source into destination

imagine an external drive at `/media/BigDisk`

```
mkdir /media/BigDisk/backup
sudo rsync -av --delete /etc /home /usr/local /media/BigDisk/backup
```

delete option deletes files that are no longer there

could alias this whole command

rsync over a network

there is also an rsync server.

19 Regex:

`-i --ignore-case Ignore case. Do not distinguish between uppercase and lowercase characters.`

`-v --invert-match Invert match. Normally, grep prints lines that contain a match. This option causes gr`

`-c --count Print the number of matches (or non-matches if the -v option is also specified) instead of t`

`-l --files-with-matches Print the name of each file that contains a match instead of the lines themselv`

`-L --files-without-match Like the -l option, but print only the names of files that do not contain matc`

```
-n --line-number
Prefix each matching line with the number of the line within the file.
```

`-h --no-filename For multi-file searches, suppress output of filenames`

make directories

```
grep bzip dirlist*.txt
grep -l bzip dirlist*.txt # just list directories
```

`grep -L bzip dirlist*.txt # just ones that that don' have match`

regex metacharacters inclue

`^ $ . [ ] { } - ? * ( ) | \`

`grep -h '.zip' dirlist*.txt`

here the `.` means any character.

the `^` and `$` sign only if the regex is at the beginning or ending of the line.

bracket expansions: `grep -h '[bg]zip' dirlist*.txt` bracket expansions (with netation): `grep -h '[^bg]zip' dirlist*.txt`

- in this case the caret is negation but only if it is the first one.
- there still has to be a character

character range:

`grep -h '^[A-Z]' dirlist*.txt`

there can be multiple ranges

`grep -h '^[A-Za-z0-9]' dirlist*.txt`

to actuallyn match a dssh make it the first one `grep -h '^[-AZ]' dirlist*.txt`

there is a whole list of printable characters classes.

useful because range in shell expansion (not regex) is based on dictionary collation order.

example: [:digit:]

basic vs extended regular expressions

use `--E` option for extended regex.

basic regec characters are `^ $ . [ ] *`

extended regex characers are `() { } ? + |`

alternation

`echo "AAA" | grep -E 'AAA|BBB' echo "AAA" | grep -E 'AAA|BBB|ccc' grep -Eh '^(bz|gz|zip)' dirlist*.txt`

quantifiers

the `?` makes it optional here is a crazy regex for a phone number: `^\(?[0-9][0-9][0-9]\)? [0-9][0-9][0-9]-[0-9][0-9][0-9][0-9]$`

the parehtnese are optional.

there is also the `*` which means match zero or more times, whereas `?` just matches once.

here is a crude way to match a sentence: `[[:upper:]][[:upper:][:lower:] ]*\`

+ matches one or more times

`^([[:alpha:]]+ ?)+$`

{} match a specific number of times

`{n}, {n,m} {n,} {,m}`

so

`^\(?[0-9][0-9][0-9]\)? [0-9][0-9][0-9]-[0-9][0-9][0-9][0-9]$`

becomes

```
\(?[0-9]{3}\)? [0-9]{3}-[0-9]{4}$
```

regex with find

grep wants lines that contain a match, find wants things that are exact match

```
find . -regex '.*[^-_./0-9a-zA-Z].*'
```

will match messy file names

regex with locate

```
locate --regex 'bin/(bz|gz|zip)'
```

Ch. 20 Text Processing

cat

`cat -A` displays the control sequences. unix ends with linefeed (*ASCII 10*) while msdos use *ASCII 13*

there is `dos2unix` and `unix2dos`.

sort

sorts contents of standard input and sends results to standard output.

or gan go

```
sort file1.txt file2.txt file3.txt > final_sorted_list.txt
```

`-b` is ignore blanks. `-f` is ignore case. `-n` is numerics. `-r` is reverse. `-m` don't do sorting; it is `merge`. `-o` is `--outpuf-file` `-t`field separator.

```
ls -l /usr/bin | sort -nrk 5 | head
du -s /usr/share* | sort -nr | head
```

sort uses whitespace as delimeters by default.

can take multiple keys:

```
sort --key=1,1 --key=2n distros.txt
```

to include the malformed date:

```
sort -k 3.7nbr -k 3.7nbr -k 3.4nbr -k 3.4nbr distros.txt
```

`b` suppresses leading spaces. the dots specify the character to start at in a field, and b ignores leading spaces; apparently only one whitespace character is the delimeter.

`/etc/passwd` uses colons as delimetes.

can sort it with `sort -t ':' -k 7 /etc/passwd | head`

uniq

removes adjacent duplicate outputs.

`-c` output number of time line occurs `-d` output only the repeated ones `-g n` skip n fields but no option to change field from whitespacce `-i` ignorrecase `-s n` skip n characters `-u` onply print unique lines.

cut

get portions of each line: `-c` for characters, `-f` for fields, `-d` is delimter, `--complement` to.

in my `distros.txt` is space delimited so we can go:

cut -f 3 -d ” ” distros.txt | cut -c 7-10

expand

converts tabs to appropriate number of spaces

25 Starting a Project

Anything you can output to a echo you can assign to a variable, with all the same expansion rules.

variables are created whenever they are used so we have to be careful when we are making something

can surround variables with braces, which disappear after expansion, so that only what is intendet to be expanded is expanded.