The Linux Command Line Notes

Jordan Mandel

June 29, 2022

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`

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'
```

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}
```

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