Linux Filesystem Operations

ese2025

Unix File Name Specifiers a.k.a “File Globbing”

- In Linux, you have command-line tools to search, create or modify batches of files. These are powerful features of the command line that have no counterpart with GUI interfaces! You can use them in your BASH scripting as well.
- For example, you can create a number of files to experiment with. Let's begin with:

```
$ cd ~
$ mkdir temp
$ cd temp
$ touch file{0..9}{a..c}.txt
$ ls -l
```

File Globbing Cont’d

- The long directory listing should produce a set of 30 files, with various names, but with a particular convention. In each file name, there is "file" followed immediately by a number from 0 to 9, followed immediately by a letter from a to c, followed by the file extension, .txt. The curly braces are a BASH feature, allowing you to specify arrays. Quite often, we want to search for certain files, because one those files contain the data that we are interested in. For example, to get all the "a" files, one could use:

```
$ ls -l file?a.txt
```

- where the question mark is a single "wild card" character that stands for any valid file-name alphanumeric character.

File Globbing Con’t S’more

- Or, to see only the 4, 5, or 6 files with a "b" suffix, one could use:

  $ ls -l file[4-6]b.txt

- where the square brackets are used to indicate a range. Now, how about the 5 to 7 range, with only b and c files?

  $ ls -l file[5-7][b-c].txt

- Or, how about any file that starts with "file1":

  $ ls -l file1*

Even more File Globbing!

- where the * wildcard character represents any valid string that might follow "file1" including an extension.

- Try out a few of your own!

Unix Pipes and Tees

- One the Linux command line, you can redirect the output of a command from standard output to a number of possible destinations. Usually, we send the data to another command. Go to a temporary directory in your home directory:

```
$ cd ~
$ cd temp    # create “temp/” if it doesn’t already exist
```

- Try the following command, which dumps the kernel ring buffer:

```
$ dmesg
```

Pipes & Tees Cont’d

- There is lots of text produced with this command, and it would be more helpful to see the information on a page-by-page basis. We can send the output of dmesg to the more command instead:

```
$ dmesg | more
```

- Now, the information is displayed in manageable chunks. Using a "tee" to borrow a plumbing analogy, we can send the text to both a file as well as to standard output via more:

```
$ dmesg | tee dmesgout.txt | more
```

Pipes & Tees Cont’d

- which creates the file (or overwrites it if it does not exist), named dmesgout.txt. If we wish to append data to a file using tee, rather than overwrite, you can use tee -a:

```
$ touch file0a.txt
$ echo "this is some data" | tee -a file0a.txt | more
$ echo "this is even more data" | tee -a file0a.txt | more
```

- You can also use file name specifiers to send the same data to multiple files.

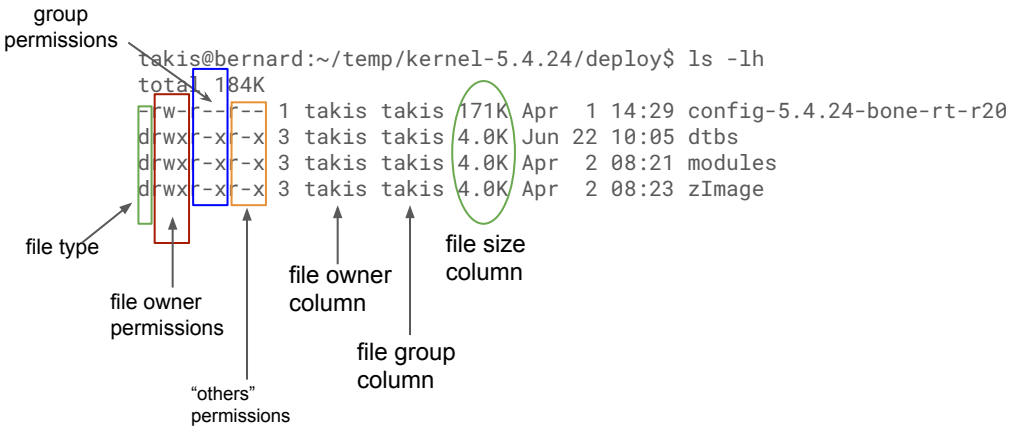
```
$ touch file{0..9}{a..c}.txt
$ echo "have some data!" | tee -a file[4-7][b-c].txt | grep "data"
```

Pipes & Tees Cont’d

- where, in this example, we have sent the stream to the "grep" command, described in class.
- Try some of these features on your own!

Unix File Ownership and Permissions

- Unix-like operating systems (such as Linux and MacOS), are structured with users, groups and permissions. This provides inherent security to the OS, since only certain users are able to modify critical aspects of the filesystem
- when performing a long-format directory listing, you will see many characters preceding the actual file name, as in:



Ownership & Permissions Cont’d

- change ownership of a file:

  $ chown new_owner:new_group <file or directory name>

- change permissions of a file (many ways to do this!)

  $ chmod go+x <file name> # make a file for the group or others
  $ chmod 777 <file name> # give all read, write and executable permissions
  $ chmod ugo-rw <file name> # take away read-write privileges from all
  $ chmod 644 <file name> # owner: rw, group: r, others: r