

The Shell and shell programming: BASH

Disclaimer: The opinions expressed in this presentation are the views of the author and do not reflect the official policy or position of any agency of the U.S. government.

Copyright © 2015 Marc Ronell
email: mronell@alumni.upenn.edu

Newton Free Library
October 14, 2015

The GNU Copyleft statement

<http://www.gnu.org/copyleft/copyleft.en.html>

This information is free; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This work is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this work; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

Two most important programs for UNIX users:

- The Shell
- An Editor (like EMACS)

At the interactive Shell

- A command ends with a newline or a ; (semicolon)
 - *ls* or *pwd*;
 - *ls* - list directory
 - *pwd* - print working directory
 - *ls; pwd* is the same as separately typing the commands on 2 lines

GNU/Linux

Shell pipes: send commands from one command to the next

The Pipe character | allows you to direct the stdout (output) of one command to the stdin (input) of the next command

```
$ date '+%c'
```

```
Mon 12 Oct 2015 08:57:32 AM EDT
```

```
$ date '+%c' | cut -d' ' -f3,4
```

```
Oct 2015
```

- For the second line we pipe the output of date, Mon 12 Oct 2015 08:57:32 AM EDT
- Into the function cut
- Cut splits the line based on delimiter spaces (-d' ')
- and prints only the 3rd and 4th fields (-f3,4)
- Giving us the output: Oct 2015
- See “man cut” or “info cut” or from emacs “C-h i” and search for the cut documentation.

The `grep` command lets us search for strings within a file.
Search for all lines in a file containing Fred

```
$ grep -e Fred data1.txt
```

```
123-45-6789 Smith George 123 Park Place Newton MA Samantha  
Joe,Fred Emily,Sue  
876-54-3210 McCabe Fred 52 Elm Street Lexington MA Winifred  
Sam,Ed Nancy
```

GNU/Linux

Shell Grep: eliminate lines from output, filter

- Grep can be used to filter lines out of a file
- Strip out all lines starting with a # character
- `grep -v '^#' data1.txt`
- The ^ tells grep to look for all # at the line beginning.
- -v tells grep invert the match or eliminate matching lines
- See “man grep”; info grep; In emacs, C=h i;

Combinations of grep and sort can be used to sort the file

- First, bring over the header information
- Pull over all data which starts with a #
- Get rid of the -v which inverts the search; Keep the lines which match; > write output to file.
- `grep '^#' data1.txt > data1_sorted.txt`

```
## Data file for Newton Free Library GNU/Linux course Homework  
1
```

```
## Filename: data1.txt
```

```
## copyright Marc Ronell 2015
```

```
##
```

```
## SS Number Last First Address Town State Wife Sons Daughters
```

```
##
```

Sort an output stream and append (» appends) to file.

- Pull all lines which are not comments; comments are gone
- `grep -v '^#' data1.txt`
- Then pipe the results to sort
- `grep -v '^#' data1.txt | sort`
- Send the results to an output file.
- `grep -v '^#' data1.txt | sort » data1_sorted.txt`

Similarly, sort data2.txt to produce data2_sorted.txt

In order to use the join command, we must eliminate the comments in both files:

- Instead of appending the data below the comments in the file
- Just over write the file contents (use > not »)
- `grep -v '^#' data1.txt | sort > data1_sorted.txt`

Similarly, sort data2.txt to produce data2_sorted.txt without comments

Once the data files are sorted, they can be joined on matching fields

- `join -1 1 -2 1 data1__sorted.txt data2__sorted.txt`
- join the two files using field 1 of file 1 (-1)
- And field 1 of file 2
- where file 1 is `data1__sorted.txt`
- and file 2 is `data2__sorted.txt`
- Set the field separator to be a single tab character (`-t $'\t'`)
- `join -t $'\t' -1 1 -2 1 data1__sorted.txt data2__sorted.txt > output.txt`

- Which car does Karen's parents own and what is its license plate?
- Here, the tab character is being treated as an instantiated variable

```
$ grep -e Karen output.txt | cut -d$'\t' -f15,16  
Toyota URF634
```

- List all of the kids whose family owns a Toyota.

```
$ grep -i -e Toyota output.txt | cut -d$'\t' -f8,9  
Joe,Fred Emily,Sue  
Ian Karen  
Alex Zoey,Nina
```

Write a code which loops through a list or set of filenames:

```
for var in list of words
do
commands
done
```

```
$ for i in *
> do
> echo $i
> done
data1__sorted.txt
data1.txt
data2__sorted.txt
data2.txt
```

Compute two weeks from today

```
$ date -date="today"
```

```
Mon Oct 12 20:33:26 EDT 2015
```

```
$ date -date="today + 2 months"
```

```
Sat Dec 12 19:33:36 EST 2015
```

```
$ date -date="today + 2 months + 2 days"
```

```
Mon Dec 14 19:40:30 EST 2015
```

```
$ date -date="today + 2 years"
```

```
Thu Oct 12 20:33:47 EDT 2017
```

- The Shell is a script interpreter.
- Can create and manipulate files.
- Can pipe output of one program as input to the next.
- Pipe together functions to accomplish work.
- When you write conventional programs, they can call shell scripts to do much of the work.