Session: Shell Programming
Topic: Advanced Commands

Daniel Chang

Text File Processing

Reading and filtering text files

- `cut` - Print specific columns from a text file
- `awk` - Print specific lines from file based on filter
- `grep` - Print lines or filenames filtered by Regular Expressions (regex)
- `find` - Locate files based on pattern and perform commands

Modifying text files

- `sed` - Edit portions of text file based on filter
- `|, >, >>` - Redirect desired output to original or new file

Reference

- `awk` - http://www.vectorsite.net/tsawk.html
- `sed` - http://www.grymoire.com/Unix/Sed.html
- Regular Expressions - http://www.regular-expressions.info/

Application - Simple Database

```
> cat dvd.txt
1994  12  19.99  Action     True Lies
2003  10  24.99  Adventure  Pirates of the Carribean
1990  3   9.99   Comedy     Kindergarten Cop
1990  10  14.99  action     Total Recall
1996  6   14.99  Comedy     Jingle All the Way
```

Reading Fields

```
    > cut -c 11-17 dvd.txt
    19.99
    24.99
    9.99
    14.99
    14.99

    > cut -c -28 dvd.txt
    1994  19.99  Action
    2003  24.99  Adventure
    1990  9.99   Comedy
    1990  14.99  action
    1996  14.99  Comedy
```

Filtered Reading - awk

- "awk" is a program that expects a text file containing "awk commands" (can contain shell commands)
- Commands passed through the command line must be sent as-is (single-quoted string)

awk Command Generally

```
    > awk <search pattern> {<awk commands>}
```

Example

```
> awk '/[Aa]ction/' dvd.txt
1994  12  19.99  Action     True Lies
1990  10  14.99  action     Total Recall

> awk '/[Aa]ction/ {print $5}' dvd.txt
True
Total

> awk '/[Aa]ction/ {print $0 | "cut -c 18-"}' dvd.txt
Action     True Lies
action     Total Recall

> awk '/[Aa]ction/ {if ($1 > 1992) print $0 | "cut -c 29-"}' dvd.txt
True Lies
```

- Commas can be used to space outputs
- 'BEGIN' and 'END' occur before and after the searching of all the lines

```
> awk 'END {print NR,"DVDs"}' dvd.txt
5 DVDs
```

awk Program File

- Typically awk commands are stored as a program in a file and executed with

```
awk -f <filename>
```

awk Program File Generally

```
# comments do not work on all systems
BEGIN      {<initialization commands>}
<pattern1> {<commands1>}
<pattern2> {<commands2>}
...
END        {<final commands>}
```

- Multiple commands must be separated by ";" (semicolon)

User Variables

- Can be declared by simply using a new variable name
- Common operations: `+, -, *, /, ++, +=, --, -=`
- Use similar to C or C++ variables
- Referenced by simply using name (no special character)

Predefined Variables

- NR - Count of the number of input lines (real-time value)
- NF - Count of the number of words in an input line ($NF corresponds to the last field)
- FILENAME - Name of input file
- FS - "Field Separator" character used to divide fields on the input line (default is all "white space"). FS assigned another character to change the field separator.
- RS - "Record Separator" character delimiting records, which by default are single lines separated by a "newline".
- OFS - "Output Field Separator" used when printing (default is a "space").
- ORS - "Output Record Separator" used when printing (default is a "newline" character).

awk Program File Example

```
> cat prog.awk
# process dvd.txt
BEGIN { action_num = 0; adventure_num = 0;
        action_cost = 0; adventure_cost = 0 }
/[Aa]ction/    { action_num++; action_cost += $2 * $3}
/[Aa]dventure/ { adventure_num++; adventure_cost += $2 * $3 }
END { print "DVD Inventory";
      printf("\n");
      printf("Action Movies:        %2d\n", action_num);
      printf("Inventory Value: %7.2f\n", action_cost);
      printf("Adventure Movies:     %2d\n", adventure_num);
      printf("Inventory Value: %7.2f\n", adventure_cost);
      printf("\nTotal DVDs             %d\n", NR) }


> awk -f prog.awk dvd.txt
DVD Inventory

Action Movies:         2
Inventory Value:  389.78
Adventure Movies:      1
Inventory Value:  249.90

Total DVDs             5
```

Filtered File Editing - `sed`

```
sed [flags|range] '<command>' [< oldfile > newfile]
sed [flags|range] '<command>' [filename]
```

- *oldfile* - File to be used as input is redirected into command
- *newfile* - Output redirected into this file
- *filename* - If redirection is not used *filename* can be used to specify the input file
- Typically <*command*> must be literalized (single quotes)

Substitution Command

```
sed s/<pattern>/<newpattern>/ [filename]
```

- "&" can be used in <newpattern> to refer to pattern matched
- Patterns are actual Regular Expressions
- Wildcards refer to quantities of the preceding character or set only (they do not standalone)

Example - Substitution

```
> sed 's/Adventure/Adv      /' dvd.txt
1994  12  19.99  Action     True Lies
2003  10  24.99  Adv        Pirates of the Carribean
1990  3   9.99   Comedy     Kindergarten Cop
1990  10  14.99  action     Total Recall
1996  6   14.99  Comedy     Jingle All the Way

> sed 's/[0-9]*/&Y/' dvd.txt
1994Y  12  19.99  Action     True Lies
2003Y  10  24.99  Adventure  Pirates of the Carribean
1990Y  3   9.99   Comedy     Kindergarten Cop
1990Y  10  14.99  action     Total Recall
1996Y  6   14.99  Comedy     Jingle All the Way
```

- `sed` actions can be restricted to specific lines
- Ranges are specified using ',' (not '-'). '$' specifies last line

```
$ sed '3 s/[0-9]*/&Y/' dvd.txt
1994  12  19.99  Action     True Lies
2003  10  24.99  Adventure  Pirates of the Carribean
1990Y  3   9.99   Comedy     Kindergarten Cop
1990  10  14.99  action     Total Recall
1996  6   14.99  Comedy     Jingle All the Way

$ sed '3,$ s/[0-9]*/&Y/' dvd.txt
1994  12  19.99  Action     True Lies
2003  10  24.99  Adventure  Pirates of the Carribean
1990Y  3   9.99   Comedy     Kindergarten Cop
1990Y  10  14.99  action     Total Recall
1996Y  6   14.99  Comedy     Jingle All the Way
```

Deletion Command

`sed` **`/<pattern>/ d`** `[`*`filename`*`]`

Example

```
$ sed '/[Aa]ction/ d' dvd.txt
2003  10  24.99  Adventure  Pirates of the Carribean
1990  3   9.99   Comedy     Kindergarten Cop
1996  6   14.99  Comedy     Jingle All the Way
```

Print Command

`sed` **`-n /<pattern>/ p`** `[`*`filename`*`]`

- Will print all lines matching patterns
- "-n" prevents normal printing (of matched lines)

Example

```
$ sed -n '/[Aa]ction/ p' dvd.txt
1994  12  19.99  Action     True Lies
1990  10  14.99  action     Total Recall
```

`cut [-d` *`char`*`**] -c|-f <**`***`range`***`**>**`** *`filename`*

Description: Prints selected columns from a text file.

Options:

- **`[-c]`** - Print characters range
- **`[-f]`** - Print field range (this can be incompatible with use of cut)
- **`[-d]`** - Use specified delimiter instead of TAB (specify single character)
- *filename* - Specifies text file to read (by default will not be modified)

Range:

- **`"n"`** - Single character or field position
- **`"n-"`** - From position to end of line
- **`"n-m"`** - Range of positions
- **`"-m"`** - From start to position

Example:

```
# print characters 10-20 from all lines
cut -c 10-20 table.txt

# print first four fields
# use single space as delimiter, not tab
cut -d ' ' -f -2 table.txt
```

`grep [-i] [-l] [-n][-v]` *`text filename`*

Description: Finds characters or lines of text in one or more files and displays the information to the screen.

Options:

- **`[-i]`** - ignores case
- **`[-l]`** - Displays only names of files not actual lines.
- **`[-n]`** - Displays the line numbers
- **`[-v]`** - Look for lines that don't have text
- *`text`* - word or phrase that contains text you want to search for. If there are spaces or things that confuse UNIX enclose them in quotation marks. Actually a "regular expression", which can be very complex
- *`filename`* - File(s) you want to search.

Example:

```
grep -i "smith" *

alias finger "ypcat passwd|grep -I"
finger dchang
```

find *directories* **[name** *filename***] [-user** *username***] [-atime** *+days***] [-mtime** *+days***] [-print] [-exec** *command* {} **\:][ok** *command* {}\;]

Description: Finds one or more files, based upon rules you give, and then allows you to execute a command on those files. Totally cryptic.

Options:

- *directories* - list of directories you want to search
- name *filename* - file(s) you want to search for
- user *username* - user who owns the files
- atime +*days* - Files that have not been accessed in +days. A minus sign instead of a + sign you get the files that were looked within those number of days.
- mtime +days - Files that have not been modified in those number of days. A minus sign instead of a + signs gets you files that were modified within those number of days.
- print - Displays names of files. Always use this.
- exec *command* {} \; - Runs the *command* when it finds the files. Puts the found filename inside the {}. Be sure and use the \; as a separator.
- ok *command* {}; - Same as exec only it asks before it runs the command.

Example:

```
find ~dchang\wishlist -name dvd.txt -exec cat {} \;
```