PURDUE UNIVERSITY®

CS 25200: Systems Programming

Lecture 2: More *NIX Commands, Shell Scripting in Bash

Prof. Turkstra

Announcements

- Midterm exam will be Thursday, October 17, 8:00pm – 10:00pm
  - WALC 1055
- Makeups will be granted only in the *most extreme* circumstances

Lecture 02

- Commands wrap-up
- Getting started with Bash
  - Data types
  - Reading and writing
  - Control loops
  - Decision making

whereis

- Locates the binary, source, and manual files for the given command
  `whereis [options] name`
- Only works if the requisite db has been generated
- Example...
  `whereis ls`

which

- Shows the full path to a command

```
which [options] programname
```

- Examples…

```
which ps
which ls
which sudo
```

head

- Collects the first n lines of a file with n defaulting to 10 if unspecified

```
head [-n] file
```

- Examples…

```
head -30 yuk # top 30 lines
head yuk     # top 10 lines
head *       # top 10 lines of every file
```

tail

- `tail [+/-[n] [b|c|l] [-f]] file`
  delivers n units from the file
  - `+n` counting from the top
  - `-n` counting from the end
  - `n` defaults to -10 if unspecified

  counting by
  - `b` blocks
  - `c` characters
  - `l` lines (default)

  `-f` means follow - infinite trailing output (use `ctrl-c` to stop)
- Has buffer limitations - see the man page

Examples

- `tail +10 yuk` # all lines beyond line 10
  `tail -30 yuk` # last 30 lines
  `tail -30c yuk` # last 30 characters
  `tail -30f yuk` # last 30 lines, continuing outputting any added lines
  `tail -30 *` # last 30 lines of all files

cut

- Used to make vertical cuts across a file
  `cut -flags columns or field filename`
- Useful flags
  `-c` characters
  `-d` field delimiter
  `-f` fields
- See man page for more information

Examples

- Given a file `data`,

```
12345 7890 abcd efgb
This is line one
this is no big deal
```

- ...and this script,

```
#! /bin/bash
cut -c1-5,8- data
echo '------'
cut -d' ' -f2-3 data
exit 0
```

- We get this output:

```
12345890 abcd efgb
This  line one
this  no big deal
------
7890 abcd
is line
is no
```

Another example

- Here's another example:

```
#! /bin/bash
DAY_OF_WEEK="$(date | cut -d' ' -f1-1)"
MONTH="$(date | cut -d' ' -f2-2)"
DAY="$(date | cut -d' ' -f3-3)"
YEAR="$(date | cut -d' ' -f6-)"
echo "Date: $(date)"
echo "Month: ${MONTH}"
echo "Day: ${DAY}"
echo "Year:  ${YEAR}"
echo "Day of the week: ${DAY_OF_WEEK}"
exit 0
```

- Which outputs...

```
Date: Mon Jul 22 16:01:17 EST 1996
Month: Jul
Day: 22
Year: 1996
Day of the week: Mon
```

paste

- Used to combine lines from two files together
  `paste [-dlist] file1 file2 …`
- By default concatenates corresponding lines of the files together using a tab as the separator
- Example:
  `paste -d" " x y z`
  concatenates the corresponding lines of the files `x`, `y`, and `z` together using the list of separators circularly. In this case the list only contains a single space.

More examples

- `paste -s [-d list] file1 file2 …`
  merges lines together serially (one file at a time)
- `paste -s -d" \n" yuk`
  pastes each pair of lines in the file `yuk` together
  - the list specified with `-d` is a space followed by a newline
- See man page for more options and information

wc

- Word count
  `wc -[c|w|l] file`
- Used to count
  `-c` characters
  `-l` lines
  `-w` words (separated by whitespace)
- Default is all three

Example

```
#! /bin/bash
wc x.c
wc -l x.c
wc -w x.c
wc -c x.c
NL=$(wc x.c)
echo ${NL}
echo "\"${NL}\""
LL=$(wc -l < x.c)
echo "\"${LL}\""
echo ${LL}
exit 0
```

Output:

```
  301   878   8382  x.c
301 x.c
878 x.c
8382 x.c
301 878 8382 x.c
"    301    878    8382 x.c"
"    301"
301
```

A quick look at `sort`

- Read the man page for more information
- A few very useful flags:
  `-u` unique lines only
  `-tx` field separator `x` (default is whitespace)
  `-b` ignore leading blanks
  `-r` reverse sort
  `-n` numbers not characters
  `-k` sort on fields (up to 10 `-k` options allowed)
  - Note: field numbers begin with 1

sort example

- This example,

```
#! /bin/bash
cat sdata
echo
sort sdata
exit 0
```

- Yields this output:

```
1  a  5                0  b  3
2  b  4                1  a  5
3  a  4                1  a  5
1  b  4                1  b  4
0  b  3                2  b  4
1  a  5                3  a  4
```

Another example

- This example,

```
#! /bin/bash
cat sdata
echo
sort -u sdata
exit 0
```

- Yields this output:

```
1  a  5                0  b  3
2  b  4                1  a  5
3  a  4                1  b  4
1  b  4                2  b  4
0  b  3                3  a  4
1  a  5
```

-u and -k example

- This example,

```
#! /bin/bash
cat sdata
echo
sort -u -k 2 sdata
exit 0
```

- Yields this output:

```
1  a  5              3  a  4
2  b  4              1  a  5
3  a  4              0  b  3
1  b  4              2  b  4
0  b  3
1  a  5
```

Another -u and -k example

- This example,

```
#! /bin/bash
cat sdata
echo
sort -u -k 2,2 sdata
exit 0
```

- Yields this output:

```
1  a  5              1  a  5
2  b  4              2  b  4
3  a  4
1  b  4
0  b  3
1  a  5
```

Specifying field order

- This example,

```
#! /bin/bash
cat sdata
echo
sort -ur -k 2,2 -k 3,3 -k 1,1 sdata
exit 0
```

- Yields this output:

```
1  a  5              2  b  4
2  b  4              1  b  4
3  a  4              0  b  3
1  b  4              1  a  5
0  b  3              3  a  4
1  a  5
```

Beating the dead horse

- This example,

```
#! /bin/bash
cat sdata
echo
sort -k 3,3 -k 1,1 -k 2,2 sdata
echo
sort -k 3bn,3 -k 1bn,1 -k 2b,2 sdata
exit 0
```

- Yields this output:

```
1  a  5          3  a  14          11 b  4
11 b  4          11 b  4           1  a  5
12 c  40         12 c  40          3  a  14
2  a  40         2  a  40          2  a  40
21 c  51         1  a  5           12 c  40
3  a  14         21 c  51          21 c  51
```

awk

- Pattern scanner and processing language
- Sequence of rules…

```
pattern {action}
…
default {action}
```

- Pattern is a regular expression
- Action is a sequence of statements to execute when pattern is matched

awk

- This example,

```
#! /bin/bash
cat adata
echo
awk ‘{print $1}’ adata
echo
awk ‘$2 ~ /c/’ adata
exit 0
```

- Yields this output:

```
1  a  5          1          12 c  40
11 b  4          11         21 c  51
12 c  40         12
2  a  40         2
21 c  51         21
3  a  14         3
```

More awk

- This example,

```
#! /bin/bash
cat data
echo
awk ‘$1 ~ /2/’ data
echo
awk ‘$1 ~ /^2/’ data
exit 0
```

- Yields this output:

```
1  a  5          12 c  40          2  a  40
11 b  4          2  a  40          21 c  51
12 c  40         21 c  51
2  a  40
21 c  51
3  a  14
```

Even more awk

- This example,

```
#! /bin/bash
cat data
echo
awk ‘/^2/ {print $2}’ data
echo
awk ‘$3 == “40”’ data
exit 0
```

- Yields this output:

```
1  a  5          a            12 c 40
11 b  4          c            2  a 40
12 c  40
2  a  40
21 c  51
3  a  14
```

Math!

- This example,

```
#! /bin/bash
cat data
echo
awk '{ sum += $3 } END { print sum }' data
echo
awk '/^2/ { sum += $1 } END { print sum }' data
echo
exit 0
```

- Yields this output:

```
1  a  5          154          23
11 b  4
12 c  40
2  a  40
21 c  51
3  a  14
```

sed

- Stream editor
  - Performs basic text transformations on input
    - Reads line into buffer
    - Applies commands
    - Outputs (revised) line

sed

- This example,

```
#! /bin/bash
cat data
echo
sed ‘s/a/f/’ data
echo
sed ‘s/1/5/’ data
echo
exit 0
```

- Yields this output:

```
1  a  5          1  f  5          5  a  5
11 b  4          11 b  4          51 b  4
12 c  40         12 c  40         52 c  40
2  a  40         2  f  40         2  a  40
21 c  51         21 c  51         25 c  51
3  a  14         3  f  14         3  a  54
```

More sed

- This example,

```
#! /bin/bash
cat data
echo
sed ‘s/1/5/g’ data
echo
sed -e ‘2d’ -e ‘4d’ data
echo
exit 0
```

- Yields this output:

```
1  a  5          5  f  5          1  a  5
11 b  4          55 b  4          12 c  40
12 c  40         52 c  40         21 c  51
2  a  40         2  f  40         3  a  14
21 c  51         25 c  55
3  a  14         3  f  54
```

sed

- Can also specify commands in a file

```
sed -f commandfile somefile
```

find

- Search for files in a directory hierarchy

```
find [options] starting_dir expression
```

- Search directory tree for expression, applying optional tests and actions
- Examples…

```
find . -name “*.conf” -print
find . -name “*.conf” -exec chmod o+r ‘{}’ \;
```

I/O redirection - reading

- To redirect input for a program or command,
  `< file` `n < file` n is the file descriptor
  `<< file` `n << file` n is the file descriptor
- Example:

```
mail jeff@purdue.edu < my_document
```

I/O redirection - writing

- We can redirect the output from a program or command too!
  - `> file` and `n > file` Redirect output to `file`
  - `>> file` and `n >> file` Appends output to `file`
  - `>| file` and `n >| file` Overrides the noclobber option, if set
  - `>& number` Redirects the output to file descriptor `number`

More

- Redirect output and error to different files…

```
$ command > out.txt 2> err.txt
```

- Redirect stdout to out.txt, and stderr to stdout

```
$ command > out.txt 2>&1
```

I/O redirection - pipes

- Pipes enable a series of programs to work together
  `command_1 | command_2 | … | command_n`
- Functions a lot like `>` except stdout from `command_n-1` is redirected to stdin of `command_n`.
- Example:
  ```
  $ ls -l | wc -l
  46
  ```
  counts how many lines of text `ls` just output

`tee` command

- Check out the unix `tee` command...

```
any_command | tee save_out
```

  - Saves a copy of all output (sent to standard out) in the file `save_out`

```
tee save_in | any_command
```

  - Saves a copy of all input (sent to standard in) in the file `save_in`

Purdue trivia

- "Harvey Washington Wiley was the first professor of chemistry, the first state chemist, the first ROTC instructor, the first baseball coach, and the 'father of the U.S. pure food and drug law.' Yet, the Purdue board of trustees once censored him for riding a bicycle - considered unseemly conduct by a faculty member."

  - A Century and Beyond, by Robert W. Topping

UNIX Organization

- *nix has multiple components…
  - Scheduler – decides when and for how long a process should run
  - File system – provides a persistent storage mechanism
  - Virtual memory – address space isolation
  - Networking
  - Windowing system
  - Shells and applications

Shells

- There are many different shells available for *NIX-based computers. Some of them even run under that *other* OS.
- A shell is basically a command interpreter. It provides an interface between the user and the computer (operating system).
- Shells may be graphical (explorer.exe, for instance) or text-based - often times called a CLI or command line interface

Shells cont...

- When we write a shell script, the first line of the file tells the operating system which shell to use.
- Some common *NIX shells include:
  - `#! /bin/sh` Bourne shell
  - `#! /bin/csh` C-Shell
  - `#! /bin/ksh` KornShell
  - `#! /bin/bash` Bourne-Again SHell

Bash

"Bash is the GNU Project's shell. Bash is the Bourne Again SHell. Bash is an sh-compatible shell that incorporates useful features from the Korn shell (ksh) and C shell (csh). It is intended to conform to the IEEE POSIX P1003.2/ISO 9945.2 Shell and Tools standard. It offers functional improvements over sh for both programming and interactive use. In addition, most sh scripts can be run by Bash without modification." - https://ww.gnu.org/software/bash/

Diving in - variables

- Declaration and definition:
  `NAME="whatever"`
  with no spaces between or after the "="
- Accessed as `$NAME`, or better, `${NAME}`
- Examples:

```
MagicVariable1=7
MagicVariable2="Hello!"
MagicVariable3=3.1415
MagicVariable4=${MagicVariable1}
```

Special Bash "variables"

- Bash has some built-in variables that allow us to get at some important information...
  `$#` Number of command line arguments
  `$*` Command line arguments
  `$0` The name of the shell script
  `$$` Current process ID number
  `$?` Return value from last executed command
  `$1` to `$N` Command line parameters (as separated by whitespace)

`echo` command

- `echo <options> Arguments`
  - Data is not formatted
  - Writes its arguments, separated by blanks (spaces) and ending with a newline, to standard output.
  - Whitespace can be protected with double or single quotes as needed.
- Example:

```
$ echo Hello, World!
Hello, World!
$
```

Common `echo` options

- See man page for more detail.

`-n` do not add a newline at the end

`-e` turn on the special meaning of the backslash `\` character

`-E` turn off backslash (default)

Examples

```
echo -n "Hello"
echo ", World!"
echo -5 degrees!
echo "-n hmm"
```

Results in the following output:

```
Hello, World!
-5 degrees!
-n hmm
```

Questions?