CST8177 – Linux II

bash startup files, awk, stty

Todd Kelley
kelleyt@algonquincollege.com

Topics

- midterms (Feb 11 and March 1)
- bash startup files
- awk
- stty

Configuring Bash Behavior

- When we customize our shell behavior by
  - setting environment variables (for example, `export PATH=/bin:/usr/bin:/sbin`)
  - setting aliases (for example `alias ll="ls -l"`)
  - setting shell options (for example, `shopt -s failglob` or `shopt -s dotglob`)
  - setting shell options (for example, `set -o noclobber`)

  we make these customizations permanent using bash startup files

Bash Startup Files

- http://teaching.idallen.com/cst8207/12f/notes/210_startup_files.html
- `~/.bash_profile` is sourced by your login shell when you log in
  - the things we set up here are done only once when we log in
  - `export`-ed variables here are inherited by subshells
  - we `source ~/.bashrc` here because login shells do not source it
- `~/.bashrc` is sourced by each non-login subshell, interactive or not
  - if the subshell is invoked with the `--norc` option, this file is NOT sourced
  - here we set up things that are not inherited by the login shell
  - inside this file, at the top, we check whether it's an interactive or non-interactive shell:

    ```
    [ -z "$PS1" ] && return
    ```
  - we set aliases in this file
  - we set options configured with `shopt` and `set` in this file

Startup File Sequence

- When a login shell starts
  1. execute commands from `/etc/profile`, if that file exists
  2. execute commands from the first of these that is readable (in order):
     1. `~/.bash_profile`
     2. `~/.bash_login`
     3. `~/.profile`
- The `--noprofile` command line option inhibits this behavior
- When a login shell exits
  1. read and execute commands from the file `~/.bash_logout`, if it exists

Startup File Sequence (cont'd)

- When an interactive non-login shell starts
  1. execute commands from `~/.bashrc`, if that file exists
- The `--norc` command line option to bash inhibits this behavior
- The `--rcfile file` option specifies that `file` should be used instead of `~/.bashrc`

System Wide Shell Configuration

- Configuration in `/etc/profile` applies to all users on the system
- The files in `/etc/skel/` are copied to newly created user accounts (can give new users a default copy of `.bash_profile` and `.bashrc`)

Non-Interactive Shells

- The bash process used to execute a shell script is non-interactive
- `stdin` and `stdout` not connected to a terminal (more details in bash manpage)
- In this case, bash will look for a filename in the variable `BASH_ENV` and source that file
- `if [ -n "$BASH_ENV" ]; then . "$BASH_ENV"; fi`

New Commands: awk

- we used awk to extract the first column of a space delimited input stream:
  - `awk '{print $1}'`
- `{print $1}` is actually an awk program
- the single quotes prevent the shell from interpreting the `{ }` and `$`
- This program prints the first field of every line of the input
- The input can come from `stdin` or from a file given as an argument

awk (cont'd)

- more generally, we have

`pattern{action}`

- awk reads its input line by line, and for each line that matches `pattern`, the `action` is taken
- If no pattern is specified, then every line matches
- if no action is specified, the default action is `print` (so `awk /this/` is like `grep this`)

awk (cont'd)

- `BEGIN` is a special pattern that matches just before the first actual input line
- `END` is a special pattern that matches just after the last actual input line
- `$0` denotes the whole input line
- `$1` denotes the first field in the input line
- `$2` denotes the second field in the input line, and so on
- `NF` denotes the number of fields
- `FS` denotes the field separator (default whitespace)

awk (cont'd)

- two main ways to set the input field separator
- as an argument on the command line

```
awk -F: '/tgk/{print $7}' /etc/passwd
```

  - this would print field 7, the user's shell, for any password record that contains `tgk`
- Or, we could set the FS variable in a BEGIN action

```
awk 'BEGIN{FS=":"}/tgk/{print $7}' /etc/passwd
```

  - notice that this uses two `pattern{action}` pairs

more awk examples

- for all lines of output from wc, print the first field

```
wc /etc/passwd | awk '{print $1}'
```

- for all lines in the /etc/passwd file, print the number of fields

```
awk -F: '{print NF}' /etc/passwd
```

- for all lines in the /etc/passwd file, print the last field – note difference between NF above and $NF here

```
awk -F: '{print $NF}' /etc/passwd
```

more awk examples

- print “RIGHTMOSTFIELD” as a header at the top, and then print the last (rightmost) field of every line in the `/etc/passwd` file – notice there are two `pattern{action}` pairs, and the second one has no pattern

```
awk -F: 'BEGIN{print “RIGHTMOSTFIELD"}   {print $NF}' /etc/passwd
```

- Same as above, but this time ignore lines that begin with # (this one uses a regular expression – we will learn regular expressions later)

```
awk -F: 'BEGIN{print "LASTFIELD"} !/^#/{print $NF}' /etc/passwd
```

New Commands: stty

- recall the effect of these control characters:
  - `^Z` suspend the current foreground process
  - `^C` terminate the current foreground process
  - `^D` end of file character
  - `^U` kill character to erase the command line
- these are actually properties of the terminal
- they can be set with the stty command
- `stty -a` : print out the current tty settings
- `stty susp ^X` :(that's a caret ^, shift-6 on my keyboard, followed by capital X) means set the susp character to CTRL-X instead of CTRL-Z

stty (cont'd)

- if you accidentally dump the contents of a binary file to your screen, and all the control characters reconfigure your terminal on you, you can reset it to sane values with

```
stty sane
```