COE 322: More Unix & Customizing your Environment

Victor Eijkhout

Susan Lindsey

Environment Customizations

- .bashrc - shell prompt
- .alias - bash shell aliases
- editor customizations
  - .vimrc
  - .emacs

Environment - Shell Prompt

Customize your command-line/shell prompt

Place this line in your .bashrc file:

```
export PS1="[\u@\h \w]\$ <\!> \]"
```

Then `source` the file to effect the changes:

```
$ source ~/.bashrc
```

Now your command-line prompt should look something like this:

```
[slindsey@isp02 ~]$ <66>
```

Change directories and see what happens.

See http://tldp.org/HOWTO/Bash-Prompt-HOWTO/bash-prompt-escape-sequences.html for more customizing options.

Environment - Bash: Aliases

Use aliases to give a new, usually abbreviated, name to a command:

```
$ alias ls='ls -F'
$ alias rm='rm -i'
$ alias h='history'
```

Environment - Bash: .alias file

- Create a text file in your home directory: .alias
- Add aliases and save
- Source .alias

```
source .alias
```

File: **`.alias`**

```
alias ..="cd .."
alias m="more"
alias rm="rm -i"
alias mv="mv -i"
alias h="history"
alias j="jobs"
alias lt='ls -lt'
alias ll="ls -l"
```

Environment - customize editor

You can customize your editors, vi and/or emacs, through the use of two respective “dotfiles” in your home, `$HOME`, directory.

- vi/vim editor - `.vimrc`

  sample: https://gist.github.com/simonista/8703722

- emacs editor - `.emacs`

  sample: https://gist.github.com/rplzzz/11258794

Quick Unix Review

- Directories
- Paths
- Redirection
- Pipes
- Permissions

Basic stuff: `ls`, `touch`, `man`

- List files command: `ls`
  - useful options: `-1`, `-a`, `-l` or in any combination

    ```
    $ ls -la mynewfile
    ```

    - This command, like many, has tons of options:

      ```
      $ ls --help
      ```

- Create or `touch` a file. `touch` also updates file access time

  `touch mynewfile`, then `ls` again.
- Use the `man` command to explore options for other commands:

  ```
  $ man ls
  $ man touch
  ```

More Basic Stuff: `cp`, `mv`, `rm`

- Copy: `cp file1 file2`
- Do this, check that it's indeed a copy.
- Rename or 'move': `mv file1 file2`
- Confirm that the original file doesn't exist any more.
- Remove:

`rm myfile` This is irrevocable!

Displaying File Contents

- Display the contents of a file with `cat`:

  ```
  $ cat myfile
  ```

- Other file display utilities: `less`, `more`, `head`, `tail`

  ```
  $ less bigfile
  $ more bigfile
  $ head bigfile
  $ tail mybigfile
  ```

- Explore options:

  ```
  man head; head -n 5 yourfile
  ```

- Put something in a file:

  ```
  cat > myfile
  ```

  end input with `Control-D`.

Directories

- Make a subdirectory ‘folder’:

  `mkdir newdir`

- Check where you are (print working directory):

  `pwd`

- Change directories:

  `cd newdir`

- Back to your home directory:

  `cd` (no arguments) or `cd ~`

Paths

A path is a location of a file in a directory structure.

```
$ ls $HOME/newuser/myfile
```

The *path* to **`myfile`** is: **`$HOME/newuser/myfile`**

- Relative path: does not start with slash
- Absolute path (such as **`pwd`** output): starts at root

Paths

- Path to your home directory: tilde **`cd ~`**
- Going out of a directory: **`cd ..`**
- You can use these symbols in paths:

```
ls newdir/subdir1/../subdir2
```

Paths : Exercise

From your home directory:

```
mkdir -p sub1/sub2/sub3
cd sub1/sub2/sub3
touch myfile
```

You now have a file `sub1/sub2/sub3/myfile`

1. From your homedir - how do you move `myfile` to `sub1/sub2/myfile`?
2. Go: `cd sub1/sub2`

   How do you now move the `myfile` to `sub1/myfile`?
3. Go to your home directory: `cd`

   How do you move `sub1/myfile` to here?

Paths - Exercise

After the following commands:

```
mkdir somedir
touch somedir/somefile
```

Give at least two ways of specifying the path to somefile from the current directory for instance for the `ls` command.

Same after doing `cd somedir`

Redirection

There are three standard files:

'standard input/output/error' (available in C/C++ as `stdin`, `stdout`, `stderr`)

- Normally connected to keyboard, screen, and screen respectively.
- Redirection: standard out to file:

```
ls > directorycontents
```

(actually, screen is a file, so it is really a redirect)

- Standard in from file: `mail < myfile`

(actually, the keyboard is also a file, so again redirection)

Redirection - Input/Output

Output into a file:

```
ls -l > listing
```

Append:

```
ls dir1 > dirlisting
ls dir2 >> dirlisting
```

Input:

```
myprogram < myinput
```

Redirection - splitting out and err

Sometimes you want to split standard and error outputs:

- Use `stdout`= 1 and `stderr`= 2:

```
myprogram 1>results.out 2>results.err
```

- Very useful: get rid of errors:

```
myprogram 2>/dev/null
```

Pipes

Redirection is command-to-file.

The pipe character: |

- Pipe: command-to-command

```
ls | wc -l
```

(what does this do?)

- Unix philosophy: small building blocks, chain together.

Pipes - More command sequencing

More complicated case of one command providing input for another:

```
echo The line count is wc -l foo
```

where `foo` is the name of an existing file.

Use backquotes or command macro:

```
echo The line count is `wc -l foo`
echo "There are $( wc -l foo ) lines"
```

Permissions

Basic permissions

- Three degrees of access: user/group/other
- three types of access: read/write/execute

```
user group other
rwx  rwx   rwx
```

Example: `rw-r-----`:

owner read-write, group read, other nothing

Permissions Setting

- Add permissions

```
chmod g+w myfile
```

- recursively:

```
chmod -R o-r mydirectory
```

- Permissions are an octal number:

```
chmod 644 myfile
```

Permissions - the **x** bit

The **x** bit has two meanings:

- For regular files: executable.
- For directories: you can go into them.
- Make all directories viewable:

```
chmod -R g+X,o+X rootdir
```

Unix help

- The `man` command - short for “manual” - Built-in Unix online documentation

  `man cp`

- Each command also has its own built-in help (usually) (note the double dash)

  `ps –help; ls –help`

- Bash Command history
- Job management

Bash: Command History

Use ! + command number to repeat a command:

```
$ history 5
66 cat hello.cpp
67 icpc hello.cpp
68 ./a.out
69 vi hello.cpp
70 history
$ !67
icpc hello.cpp
$ !68
vi hello.cpp
```

Use ! ! to repeat the last command

```
$ !!
vi hello.cpp
```

Bash: Command History

While programming, you'll often repeat the same command or variations on a command many times in a single session. Bash's history capabilities can greatly reduce the amount of typing and greatly increase development speed.

The shell keeps a history of all your previous commands in the `.bash_history` file. The history command displays this list:

```
$ history 5     #display the last five commands
$ history -c #clear the history
```

Bash: Command History

The `history` command prints your command-line history:

```
$ history 5
  65 cp test.cpp hello.cpp
  66 cat hello.cpp
  67 icpc hello.cpp
  68 ./a.out
  69 vi hello.cpp
  70 history
```

Use the ! operator + a string-pattern to match past commands:

```
$ !ic
  icpc hello.cpp
  $ !cat
  cat hello.cpp
  $ !v
  vi hello.cpp
```

Job Control Basics

Victor Eijkhout, Susan Lindsey

Job Control - Processes

Process: an application the computer is running, e.g.: `vim`, the bash shell, window manager.

- Use the **`top`** command to view **all** processes currently running.
- Use the **`ps`** command to view just your own processes.
- Each process has its own id, the `pid`
- You can manage processes using job control commands

```
$ ps
  PID TTY          TIME CMD
 6146 pts/7    00:00:00 bash
10137 pts/7    00:00:00 vim
11189 pts/7    00:00:00 ps
```

Job Control - Jobs

Job: a group of processes started from the current shell.

- Jobs can contain processes, but processes don't contain jobs.
- Use the `jobs` command to display a list of active background processes.
- Each job has its own job id, the `jobid`
- You can manage jobs using job control commands

```
$ jobs
[1]   Stopped                 vim myprogram.cpp
[2]-  Stopped                 vim xer
[3]+  Stopped                 more pascal.cpp
```

Job Control - Commands

Command	Description	Example
`ctrl-z`	Suspend the current/foreground job	within `vim` or `emacs`
`ctrl-c`	Kill the current/foreground job	an `a.out` that's hanging
`fg <jobid>`	resume a background job	`$ fg %1`
`bg <jobid>`	place a job in the background	`$ bg %2`
`kill <pid | jobid>`	kill a job or process	`$ kill 10137`
`ps`	display processes - `man ps`	`$ ps`
`jobs`	display jobs	`$ jobs`

Job Control - Example Workflow

Development without job control

```
$ <322> vi myprogram.cpp
$ <323> icpx myprogram.cpp
$ <324> ./a.out
$ <325> !322
$ <326> !ic
```

repeat forever

Development with job control

```
$ <363> vi myprogram.cpp
[1]+  Stopped                 vim myprogram.cpp
$ <364> !ic
icpc myprogram.cpp
compilation aborted for myprogram.cpp    #compilation error
$ <365> fg                                #resume editor
vim myprogram.cpp
[1]+  Stopped                 vim myprogram.cpp
$ <366> !ic && ./a.out
```

Job Control - Example Workflow 2

Edit & suspend, view jobs, resume & kill

```
$ <335> vi myprogram.cpp          #suspend the vi session
[1]+  Stopped                     vim myprogram.cpp
$ <336> icpx myprogram.cpp
$ <337> ./a.out                   #run an infinite loop
Hello, world that is beautiful!
Hello, world that is beautiful!
^Z                                #suspend program execution
[2]+  Stopped                     ./a.out
$ <338> jobs                      #list current jobs
[1]-  Stopped                     vim myprogram.cpp
[2]+  Stopped                     ./a.out
$ <339> %1                        #resume the vi session
vim myprogram.cpp
[1]+  Stopped                     vim myprogram.cpp
$ <340> kill %2                   #kill the endless loop
[1]+  Stopped                     vi endlessloop.cpp
$ <341> fg                        #resume the vi session
```

Job Control - 3 Minute Exercise

Practice some job control. Create and manage multiple edit sessions.

In your terminal:

- Edit one of your source files
- Save your edits and suspend the session
- Edit a brand new file, save, and suspend the session
- List your jobs
- Resume editing the first file