BASH Essentials 2013

Introduction

What Is?

Bourne Again Shell

Installing
1. configure
2. make
3. make tets
4. make install => /usr/local

Starting
Check $PATH
$bash

Custom dir
-prefix = $HOME configure
-prefix = $HOME/bash configure

FAQ
http://www.faqs.org/faqs/unix-faq/shell/bash/.
Korn Shell (ksh) pdksh
Perl - Practical Extraction and Report language (sed/awk)
Python – interpreted language

File System
ext2 – second extended filesystem (ext2)
4TB partitions
2GB giles
255 byte filenames
ext3 – special features for error recovery
NTFS
HFS
NFS

Suffixes
.sh
.txt
.log
.html
.tgz / tar.gz – tarball / ????

Directories
/ = root
/home /lancer/file.dat = ~/file.dat
^$HOME dir ^ user dir

. = current dir
.. = parent dir
/dev – device drivers
/bin & /usr/bin – Linux commands
/lib & usr/lib - Linux libraries
/var – config and logfiles
/etc – default config
/usr/local/bin – commands not in distro, but added by admin
/opt – commercial software
/tmp – tempfiles
/sbin & /usr/sbin – "safe" bin for admin only

Inodes / Links
- Links: hard link ref in diff or current dir
- Symbolic link – only pathname of another file

Inode – Index node
- each file gets ID#
- inode – Inoees to disk space
- density – 1/4096 is Linux default

Pipe file
- buffer between 2 progs
- once big syste sill stop app writing until reading app catches up
- 'Unix Socket' file – network sockets <-> socket buffer

Device Files
- character device file

- /dev dir
  - has all configured devices, present or not
- /dev/null
  - black hole to send unwanted data
- /dev
  - us usually admin only
- /dev/zero
  - endless stream of zeros
  - used to create new files
- /de
- /tty
  - terminal window (console)
- /dsp
  - i/f that play AU sound files on soundcard
- /fd0
  - 1st floppy deive
- /hdal
  - 1st IDE partition
- sdal
  - 1st SCSI partition
-

BASH Keywords
! esac select case fi then do for until fdone function while elif if time else in { } [[ ]]

Command Basics
- Utilities – general purpose (ie: returning the date, # of lines
- Filters – take command results + modify them
- Prompt - $, >
- $
- $date
- Arguments – change command behavior ($date"+%H:%M)
- Switches – option enable command features **($date-u)**
- Parameters
  - short: -y
  - long: --universal
- most commands support
  - - -help
  - - -verbose
  - - -version
- Comments
  - # $date - -universal #dhow the date in UTC format
  - - -: last switch in list of switches

Command-Line Editing
- vi mod – mimcis Vi + vim editors
- Emacs – emacs, nano, pico
- chunk mode: (only 1 on at a time)
  - shopt –o emass
  - shopt –o vi
- Vi mode: up, down, left, right, Typeing Inserts
- Emacs: vtrl-b
  -

Command-Line Editing

There are special key combinations to edit what you type or to repeat previous com-mands. Bash has two editing modes.These modes emulate the keys used in two popular Linux text editors. Vi modemimics the vi and vim editors.Emacs mode works like emacs, nano or pico.
The current editing mode can be checked with the shoptcommand.shopt -o emacsis on if you are in emacs mode. shopt -o vi is on if you are in vi mode. Only one mode can be on at a time.

```
$ shopt -o emacs
emacs           on
$ shopt -o vi
vi              off
```

Regardless of the mode, the arrow keys move the cursor and step through the most recently executed command:
n Left arrow —Moves back one character to the left. No characters are erased.
n Right arrow—Moves forward one character to the right.
n Up arrow—Moves to the previous command in the command history.
n Down arrow—Moves to the next command in the command history (if any).

Using the left and right arrows, the cursor moves to any position in the command. In the middle of a line, new text is inserted into the line without overwriting any old typing.

Emacs mode is the default mode on all the major Linux distributions.The most com-mon emacs keys are as follows:

n control-b—Moves back one character to the left. No characters are erased.
n
control-f —Moves forward one character to the right.
n control-p—Moves to the previous command in the command history.
n control-n—Moves to the next command in the command history (if any).
n Tab key—Finds a matching filename and completes it if there is one exact match

The filename completion feature attempts to find a matching filename beginning with the final word on the line. If a matching filename is found, the rest of the filename

```
is typed in by Bash. For example,
$ dat
is completed when the Tab key is pressed to
$ date
```

if date is the only command that can be found starting with the characters dat .
The vi mode key combinations are as follows:
n Esc—Enters/exits editing mode.
n h—Moves back one character to the left. No characters are erased.
n
l—Moves forward one character to the right.
n k—Moves to the previous command in the command history.
n
j—Moves to the next command in the command history (if any).
n Esc twice—Finds a matching filename and completes it if there is one exact match.
A complete list of key combinations (or bindings) is listed in the Bash man page in the Readline section.The default key combinations can be changed, listed, or reassigned using the bind command.To avoid confusion, it is best to work with the defaults unless you have a specific application in mind.
Other editing keys are controlled by the older Linux stty (set teletype) command. Running stty shows the common command keys as well as other information about your session. Use the -a(all) switch for all settings.

```
$ stty
speed 9600 baud; evenp hupcl
intr = ^C; erase = ^?; kill = ^X;
eol2 = ^@; swtch = ^@;
susp = ^Z; dsusp = ^Y;
werase = ^W; lnext = ^@;
-inpck -istrip icrnl -ixany ixoff onlcr
-iexten echo echoe echok
-echoctl -echoke
```

Many of these settings are used only when you're working with serial port devices and can be ignored otherwise.The other settings are control key combinations marked with a caret (^) symbol. Keys with ^@(or ASCII 0) are not defined.The keys are as fol-lows:
n erase(usually ^?, which is the backspace key on IBM-style keyboards)—Moves left and erases one character.
n intr (usually ^C)—Interrupts/stops the current program or cancels the current line.
n kill (usually ^X)—Erases the current line.
n rprnt(usually ^R)—Redraws the current line.
n stop (usually ^S)—Pauses the program so you can read the results on the screen.
n start(usually ^Q)—Resumes the program.
n susp (usually ^Z)—Suspends the current program.
n werase (usually ^W)—Erases the last word typed.
To change the suspend character to control-v, type

```
$ stty susp '^v'
```

Changing key combinations can be very difficult. For example, if you are running an X Windows server (the software that runs on a client computer) on a Microsoft Windows computer to access a Linux computer, key combinations can be affected by the following:

```
n
Microsoft Windows
n The X server software
n
The Linux window manager
n
The stty settings
Each acts like layers of an onion and they must all be in agreement. For example,
shift-insert, often used to paste text, might be handled by your X Window server
before
your Linux computer or your shell have a chance to see it.
```

Variable Assignments & Displaying Messages

Variable Assignments and Displaying Messages
Variables can be created and assigned text using an equals sign. Surround the text with

double quotes.

```
$ FILENAME="info.txt"
```

The value of variables can be printed using the printf command.printf has two arguments: a formatting code, and the variable to display. For simple variables, the for-matting code is "%s\n" and the variable name should appear in double quotes with a dollar sign in front of the name

```
$ printf "%s\n" "$FILENAME"
info.txt
printf can also display simple messages. Put the message in the place of the format-
ting code.
$ printf "Bash is a great shell.\n"
Bash is a great shell.
```

printf and variables play an important role in shell scripting and they are described in greater detail in the chapters to come

Multiple Commands

Multiple commands can be combined on a single line. How they are executed depends on what symbols separate them.

If each command is separated by a semicolon, the commands are executed consecu-tively, one after another.

```
$ printf "%s\n" "This is executed" ; printf "%s\n" "And so is this"
This is executed
And so is this
```

If each command is separated by a double ampersand (&&), the commands are execut-ed until one of them fails or until all the commands are executed.

```
$ date && printf "%s\n" "The date command was successful"
Wed Aug 15 14:36:32 EDT 2001
The date command was successful
```

If each command is separated by a double vertical bar (||), the commands are execut-ed as long as each one fails until all the commands are executed.

```
$ date 'duck!' || printf "%s\n" "The date command failed"
date: bad conversion
The date command failed
```

Semicolons, double ampersands, and double vertical bars can be freely mixed in a sin-gle line.

```
$ date 'format-this!' || printf "%s\n" "The date command failed" && \
printf "%s\n" "But the printf didn't!"
date: bad conversion
The date command failed
But the printf didn't!
```

These are primarily intended as command-line shortcuts:When mixed with redirec-tion operators such as >, a long command chain is difficult to read and you should avoid it in scripts.