

Bash Script

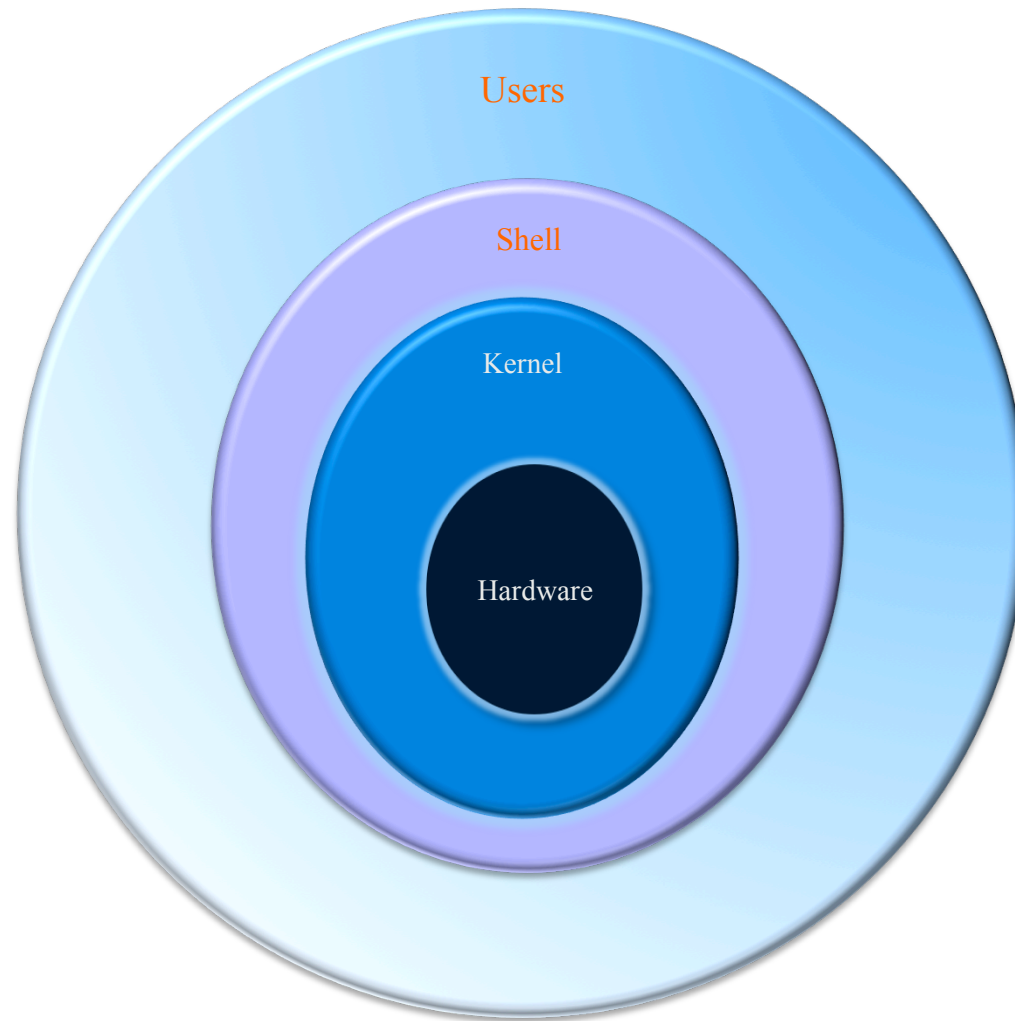
CIRC Winter Boot Camp 2015

Baowei Liu



UNIVERSITY *of* ROCHESTER

Review of Linux



From: torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)
Newsgroups: comp.os.minix
Subject: What would you like to see most in minix?
Summary: small poll for my new operating system
Message-ID: <1991Aug25.205708.9541@klaava.Helsinki.FI>
Date: 25 Aug 91 20:57:08 GMT
Organization: University of Helsinki

Hello everybody out there using minix -

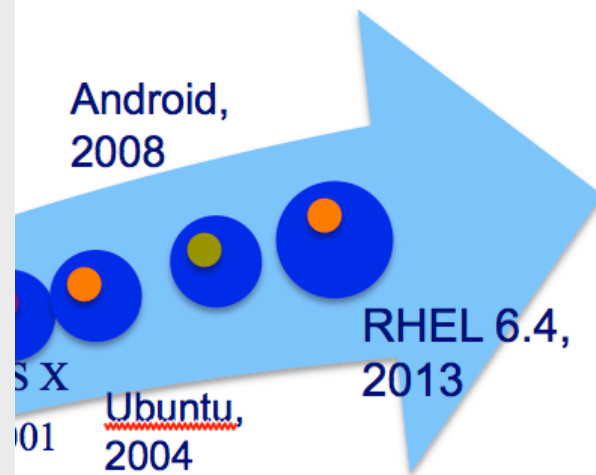
I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing since april, and is starting to get ready. I'd like any feedback on things people like/dislike in minix, as my OS resembles it somewhat (same physical layout of the file-system (due to practical reasons) among other things).

I've currently ported bash(1.08), and gcc(1.40), and things seem to work. This implies that I'll get something practical within a few months, and I'd like to know what features most people would want. Any suggestions are welcome, but I won't promise I'll implement them 😊

Linus (torvalds@kruuna.helsinki.fi)

PS. Yes – it's free of any minix code, and it has a multi-threaded fs. It is NOT protable (uses 386 task switching etc), and it probably never will support anything other than AT-harddisks, as that's all I have :-).

hells



<https://en.wikipedia.org/wiki/>



UNIVERSITY of ROCHESTER

Bash Script

- Unix/Linux commands in a text file – batch mode
- An executable program to be run by the shell (Bash)



Shell Script VS. Command Lines

- Easy to edit: long command lines
- Reusable: many command lines or options
- Flexible and Efficient: Run with different arguments.
- Easy to use with job scheduler



Shell Script VS. Other Script Language

- Easy to program: the commands and syntax are exactly the same as those directly entered at the command line. Quick start.
- Slow run speed comparing with other programming languages
- Not easy for some tasks like floating point calculations or math functions



Linux Commands

- ls: list directory contents
- cd: change directory
- man: manual
- echo



Linux Command echo

- Display a line of text
- Example: echo hello world
- “...” or ‘...’



To Write a Bash Script

- An editor: vi emacs, nano,....
- Specify interpreter as bash: `#!/bin/bash`
- Some Linux commands
- Comments: `#` (single line)
- Set executable permission



File permissions

```
-rw-r-----@ 1 liu  staff  446317 Jan 20 14:08 TcshAndShScreenCapture.png  
drwxr-xr-x@ 9 liu  staff    306 Jan 23 12:31 Tests
```

- Three scopes and permissions
- Bash script has to have execute permission to allow the operating system to run it.
- Check permissions: `ls -l`
- Change permissions: `chmod`



Bash Variables

- Create a variable: name=value
- No data type
- No need to declare but can be declared with “declare command”
- No space allowed before and after =
- Use \$ to refer to the value: \$name



Environment Variables

- env
- \$SHELL
- \$PATH
- \$LD_LIBRARY_PATH
- \$RANDOM
- \$USER



Variable Value

- Assign value: `a=2`
- Pass value: `b=$a`
- Display value: `echo $a`
- Multiple Variables
- Strong quoting & weak quoting



Assign Variable Value

- Parameter expansion $\${}$
- Command Substitution: $\$()$, or $`$
- Arithmetic expansion: $\$((\dots))$



Arithmetic Expression

- Arithmetic operators: $+$ $-$ $*$ $/$
- Integer only
- Arithmetic Expansion $((...))$
- Floating point calculation: other tools like `bc`, or `awk`



Basic calculator: bc

- An arbitrary precision calculator language
- Simple usage: `echo $a+$b | bc`
- Can use math library: `echo "s(0.4)" | bc`



Stdin, stdout and stderr

- Stdin: standard in, data stream that is going into a process
- Stdout: data stream coming from a running process
- Stderr: data stream of error messages being generated by a process



Redirect and Pipes

- Redirect between files including the three file descriptors, stdin, stdout and stderr: > >>
- Pipes takes the output of one command as the input of another command |



Bash Script

CIRC Winter Boot Camp 2015

Baowei Liu



UNIVERSITY *of* ROCHESTER

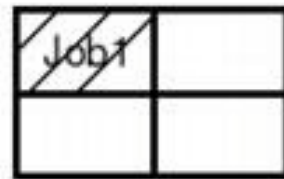
Questions after Day 1 Class

- Why we need bash script?
- Introduction to job schedulers: slurm, cron

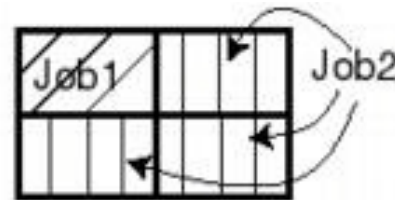


Scenarios & Examples Using Bash Script

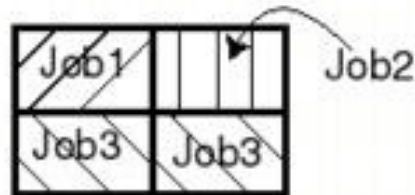
- Mul
com
- Syst
- Run



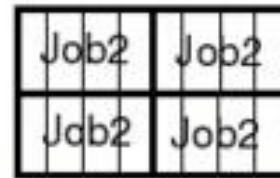
(a) Job1 started at 8:00 am.
Will finish at 10:00 am.



(b) Job2, submitted but can't start
since it needs 4 processors.
Remaining 3 reserved by Job2.



(c) At 8:30 am Job3 submitted.
Job3 backfills Job2.



(d) At 10:00 am, Job2 starts.

[http://
www.ccs.miami.edu/hpc/
lst/7.0.6/admin/
parallel.html](http://www.ccs.miami.edu/hpc/lst/7.0.6/admin/parallel.html)



Job Scheduler Slurm

- Slurm

1. Free and open-source job scheduler
2. Arbitrate resources by managing a queue of pending jobs
3. Examples for submitting jobs to our local systems can be found on info.circ.rochester.edu

http://en.wikipedia.org/wiki/Slurm_Workload_Manager



Job Scheduler Cron

- Time-based job scheduler
- Schedule the command to run with crontab
 - e
- Each line of a crontab file represents a job



Cron and Crontab

- Specify the time:

* * * * * script/command
min hr dom m dow(0-6)

- Specify every five hour

* */5 * * * script/command
0 0 1 1,6 * script/command

- Non standard macros

@yearly @reboot ...



Conditional Expression and if

- if [[conditional expression]]
 then

 else

fi



Conditional Expression: Integers

- `[[...]]`
 1. White spaces around “[[“ and “]” are necessary
 2. `[[$a -eq $b]]`: `-eq`, `-ne`, `-gt`, `-ge`, `-lt`, `-le`



Conditional Expression: Strings

- Compare strings: `[["$a" = "$b"]]`
- operators = or ==, !=, >, < (alphabetical)
- White spaces around `[[]]` are necessary!!
- -n (not null), -z(null)



Conditional Expression

- Integers (Numeric Comparison): -eq, -ne, -lt, -gt, -le, -ge
- Strings (Alphabetical Comparison): =, !=, <, >, <=, >=
- > < for Integers ?



Compound Operators

- $\&\&$, \parallel

if $[[\dots]] \&\& [[\dots]]$

then

....

fi



Feature	new test [[old test [
string comparison	>	\> (*)
	<	\< (*)
	= (or ==)	=
	!=	!=
	<=, >=	\<=, \>= (*)
integer comparison	-gt	-gt
	-lt	-lt
	-ge	-ge
	-le	-le
	-eq	-eq
	-ne	-ne
conditional evaluation	&&	-a (**)
		-o (**)

■ Arithmetic Expansion ((\$a > \$b))

==, !=, <, <=, >, >=

■ Old but more portable way [] or test

Arithmetic

<http://mywiki.woledge.org/BashFAQ/031>



Compare Floating Point Numbers

- Use Basic Calculator: `bc`
`compare_results=`echo "$a>$b" | bc``
- Operators: `==`, `!=`, `>`, `>=`, `<`, `<=`
- Convert to integer (Return 1 for True and 0 for False)
- Always check the command before using it!



Conditional Executions

- Command 1 && Command 2
- Command 1 || Command 2



Conditional Expression

- true and false commands
- Return values of Linux commands
\$?
- Shell true/false VS. Arithmetic true/false in
Arithmetic Expansion(())



Returns of Arithmetic Expansion

if the arithmetic expression brings up a value not 0 (arithmetic true), it returns 0 (shell true)

if the arithmetic expression evaluates to 0 (arithmetic false), it returns 1 (shell false)



Bash Script

CIRC Winter Boot Camp 2015

Baowei Liu



UNIVERSITY *of* ROCHESTER

Shell Expansions Review

- Parameter Expansion: `$variable`, `${variable}`
- Arithmetic Expansion: `$((expression))`
- Command Substitution: `$()` or `` ``



Brace Expansion

- Brace expansion is used to generate an list.
- {string1, string2, ...,stringN}
space not allowed between braces!!!
- Range {<start>..<end>}: {1..20}
space not allowed between braces!!
- Very first expansion to do !!
{ \$a..\$b }



Brace Expansion

- Preamble and Postscript

$a\{1,3,4\}b$

space is important!!!

- Combining and nesting

$\{a,b,c\}\{1..3\}$

$\{\{a,b,c\},\{1..3\}\}$

- Escaping backslash



Loop Constructs: for loop

- Basic Syntax

for arg in [list]

do

.....

done

- [list]:

1. Brace Expansion (string or integer): {1..5}
2. Command Substitution: `ls`
3. Arithmetic Expansion?



for loop –Arithmetic Expansion

- Basic Syntax

```
for (( expr1; expr2; expr3 ))
```

```
do
```

```
...
```

```
done
```

- Examples:
- White space are not important for Arithmetic Expansion



Loop Constructs –while loop

- Conditional Expression

while [[conditional expression]]

do

....

done

- Arithmetic Expansion

while ((arithmetic expression))

do

...

done



Loop Constructs –until loop

- Conditional Expression

until [[conditional expression]]

do

....

done

- Arithmetic Expansion

until ((arithmetic expression))

do

...

done



Other Flow Control Constructs: case

```
case expression in  
  pattern1)  
    statement;;  
  pattern2)  
    statement;;  
  ....  
esac
```

;; and *



Filename Expansion / Globbing

- Expanding filenames containing special characters
- Wild cards * ?, not include . . .
- Square brackets [set]: “-”
- Special characters: ! (other than)
- Quote special pattern character if they are to be matched literally
- Escaping backslash: protect a subsequent special character



grep and Regular Expression

- grep: search for matches to a pattern in a file and print the matched line to stdout
grep PATTERN file
- Regular Expression: globbing pattern used for text



Regular Expression

- . : Equivalent to ? in filename expansion
- .* : any string. Equivalent to * in filename expansion
- * : zero or more times, a* will match a,aa,... but not ab
- ^ : starting with, ^ab
- \$: ending with, ab\$



Regular Expression

- `[]`: “-”
- `\< \>`: exact word



sed and Regular Expressions

- sed 's/abc/xyz' File: All occurrences
- sed '5,10s/abc/xyz' File: specified lines
- sed '0~2 s/abc/xyz/' File: only in the even lines
- More complicated examples



Bash Script

CIRC Winter Boot Camp 2015

Baowei Liu



UNIVERSITY *of* ROCHESTER

sed and Regular Expressions

- Word Characters: Alphanumeric characters plus “_” [A-Za-z0-9_]
- Replace all occurrences in a line



awk

- A text-processing programming language in Linux
- awk ‘{print \$1}’
- Floating number calculations



WC

- wc: print the number of bytes, words and lines in a file.
- -c
- -l
- -w



Arrays

- Array is a numbered list
- One-dimensional only
- Create an array with = and (), or declare –a
- Array element: ArrayName[index]
- Access elements: \${ArrayName[n]} @, *
- Array size:\${#ArrayName[@]}, \${#ArrayName[*]}
- Initialize an array with brace expansion
- Delete array or element: unset ArrayName[n]
- Add element without key: ArrayName+=(...)



Strings and Manipulation

- Create a string
- Display a string
- Length of a string
- Substring: a Bash string just holds one element
- Compare strings
- Concatenate of string
- Substring extraction
- Substring replacement



Compare Strings

- `=:` `[["$a" = "$b"]]`, white space are important!!
- `!=`
- `>:` alphabetically
- `<:` alphabetically
- `-z` check if the string is null /zero-length
- `-n`



Substring Extraction

- $\$ \{ \text{string:position:length} \}$
- $\$ \{ \text{string:position} \}$



Substring Removal

- $\$ \{ \text{string} \# \text{substring} \}$: regular expression
- $\$ \{ \text{string} \# \# \text{substring} \}$
- $\$ \{ \text{string} \% \text{substring} \}$
- $\$ \{ \text{string} \% \% \text{substring} \}$



Substring Replacement

- `${string/substring/replacement}`
- `${string//substring/replacement}`
- `${string/#substring/replacement}`
- `${string/%substring/replacement}`



Functions

- Syntax

```
Function funcname{  
    commands....  
}
```

```
Function funcname(){  
    commands....  
}
```

- Pass Arguments
- Returning Values



File Manipulation

- Examine the status of a file
 - a file: True if file exists
 - s file: True if file exists and has a size greater than zero
 - f file: True if file exists and is a regular file
- Compare files
 - file1 -nt file2: newer than
 - file1 -ot file2: older than



Merge files

- join: merge files by a common column
- cat: merge files by rows



head and tail

- $-n$
- $-c$
- $-f$



Some Examples

- File name modifications
- Wrappers

