## Scripting

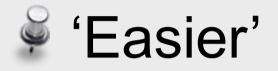
- Least Privilege Principle
- Unix scripting
- Examples
- Other solutions

Ş

## **Least Privilege Principle**

- No process or file should be given more privileges than it needs to do its job.
- Setuid programs: don't set unless necessary
- Run programs under special user id such as www and nobody if possible
- Some applications such as httpd can change its user id from root to nobody after opening the privileged port number 80.
- Temporary files shouldn't be in /tmp

## Scripting is...









## **Cons of Unix scripting**

- "Prayerful parsing"
- I/O is expensive due to process communications
- Interpretation slower than compiled code
- Interface inconsistency
- Security: TOCTTOU
  - rm /tmp/\*/\* (find /tmp -not-accessedrecently | xargs rm)

## Who scripts?

Users

#### Power users

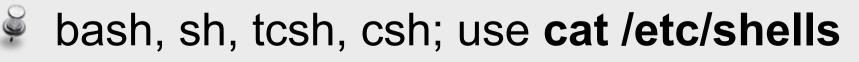
Administrators

Developers

#### Testers

## Developments

- Job Control Language
- 9 1960s Unix pipe
- 1993 Applescript
- 2005 Automator
- 2006 Windows PowerShell
- Available shells in Linux





## **Unix Shell Scripting**



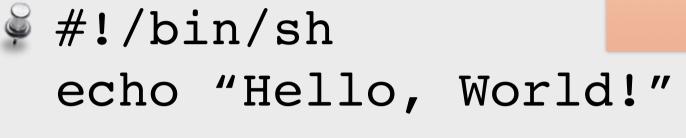
## **Unix Philosophy**

Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface.

#### Doug McIlroy Inventor of the | construct

## ./hello

No extension



\$ chmod +x ./hello \$ ./hello Hello, World! \$ sh ./hello Hello, World!

The correct UNIX way

Ş

echo -e "no newline\c"

## **Another example**

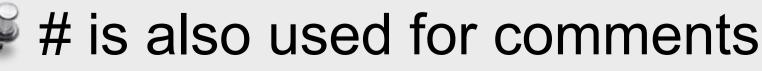
```
#!/bin/bash
clear
echo "This is information provided by mysystem.sh. Program starts
now."
echo "Hello, $USER"
echo
echo "Today's date is `date`, this is week `date +"%V"`."
echo
echo "These users are currently connected:"
w | cut -d " " -f 1 | grep -v USER | sort -u
echo
echo "This is `uname -s` running on a `uname -m` processor."
echo
echo "This is the uptime information:"
uptime
echo
echo "That's all folks!"
  COSC301 Lecture 4: Scripting
                                   10
```

## #! "Sh-Bang"

## **First** line

- ₩ #!/bin/sh
- #!/usr/bin/perl -wnl
- #!/usr/bin/env python
- Default is /bin/sh

## SetUID not honoured



## **Design Patterns**

#### 🖌 Source: 1s

Filter: sort
Filter: sort

Fread from stdin and write to stdout
Sink: less

will read from stdin and write to file
will "Cantrip": rm

do something but return nothing

#### 🖉 Compiler: tar

read from file and write to another file

## **Good scripts**

- A sensible name
  - don't clash with existing commands and programs
- No errors
- Perform the intended task
- Have a clear logic
- Efficient, no unnecessary work
- Informative, notifying users about what it is doing
- Reusable

## **BASH** basics

- Files read by bash
  - /etc/profile, .bash\_profile, .bashrc
  - depending on login, interactive, non-interactive, or use sh directly
- Built-in commands like cd and eval, exit, exec, export, ...
- Three types of commands
  - built-in, function, executable programs
- debugging a script: **bash -xv script\_file**
- Some self-study required. Read Bash Beginners Guide

## **BASH basics (cont.)**

Environment variable

A variable with name and value used by shells and processes

Use printenv or env to find them

Frey can be set by

Globally, /etc/profile, /etc/bash.bashrc

Per user, ~/.bash\_profile,~/.bashrc, ~/.profile

Solution Normal Non Interactive shell (shell scripts)

/etc/profile, ~/.bash\_profile, ~/.bash\_logout

Used by login shells

✓/etc/bash.bashrc, ~/.bashrc

Sused by interactive, non-login shells

For details: https://wiki.archlinux.org/index.php/

environment variables COSC301 Lecture 4: Scripting

## **I/O Channels**

# stdin previous pipe or terminal, ^D to 'end-of-file'

## stdout next pipe or terminal

stderr not piped

## FD 0,1,2 respectively

## Redirection

- command > file-overwriting
- command >> file-appending
- 🗳 command 2> file
  - 🗳 redirect stderr to file
- echo "Warning to stderr" >&2
  - 🖗 redirect stdout to stderr
- echo "To black hole" 2> /dev/null >&2



## "Pipe"

- Communication channel between programs
- § 5 biggest dirs in the current dir

All I/O via kernel, slow

## Variables

varname=value no spaces around '=' Assignment

💡 \$varname

Deference

- Global and local variables
  - Environment variables are global variables.
- Seen by subshell/child processes if export PATH=\$HOME/bin:\$PATH
- Beware white-space! varname="foo bar"

## Interpolation

- A built-in command in a string can be executed and the execution output will replace the original command.
- 'non-interpolated string'

```
🏺 `command`
```

"interp. string \$varname `command`"

```
$ foo=`command \`command\``
foo=$(command $(command))
```

(Bash specific)

## **Conditions**—if

#### see test(1)

#### 

*if-otherwise* 

fi

## if\_!\_grep -q ...; then if-grep-did-not-find fi

## **Conditionals**—case

```
weight case "$fo proc" in
   'fop')
     command;;
   'xep')
     command1; commandN;;
   *)
     default-command >&2
     exit 1;;
 esac
```

## Loops—for

# for i in foo bar baz do echo \$i done (( ...; ...; ...)) is a Bash-ism

## for ((i=128; i<160; i++)); do printf "ip%03d\tA\t192.168.1.%d\n" \$i \$i done</pre>

## Loops—while

# \$\overlinessimpless | while read filename do do do stuff with "\$filename" done

# while true do infinite loop body done

## Subshells

```
    I39.80.32.2 - - [26/Mar/2007:17:28:34 +1200] ↔
    "GET /path/to/file.html HTTP/1.0" 304 -
```

```
    (echo "IP Freq";
        (cat access_log;
        gzcat access_log.*.gz)
        | cut -d' ' -f1 | sort | uniq -c
        | sort -rn | awk '{print $2,$1}'
    ) | column -t
```

getting ugly, start making functions

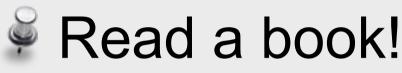
## Arithmetic

Bash-ism

expr 2 \\* 8 16

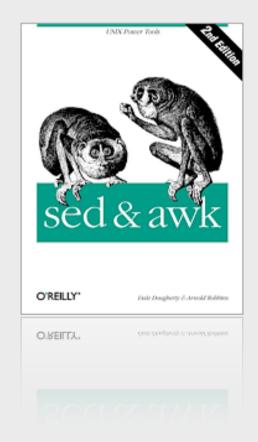
echo 'ibase=10; obase=2; 192' | bc
11000000

## Sed and Awk



### Regular expressions!

- Takes a while to learn
- A few recipes are useful



## sed—Stream Editor

- Delete header on first line sed –e 1d
- Disable FTP service in inetd
  sed -e 's/^ftp/#&/' < inetd.conf > \
   inetd.conf.new
  mv inetd.conf{,~}; mv inetd.conf{.new,}
- What requests got a 404?
  gzcat access\_log.\*.gz | sed -ne '/ 404
  [0-9]\*/s/^.\*"[A-Z]\* \(.\*\) HTTP\/
  [0-9.]\*".\*\$/\1/p'

### awk

#### Re-order fields echo 'a b c a c b' | tr ' '\n' | sort \ uniq -c | awk '{print 2, \ sort -r -k2

#### Collation

```
echo -e '1\n2\n3\n4' awk '
BEGIN{sum=0;max="?"}
max == "?" \{max = \$1\}
\{sum + = $1\}
1>\max{max=1}
END{print "Avg:" sum/NR "\nMax:" max}'
```

## A command a day...

# List descriptions of system commands

find /bin /usr/bin /sbin /usr/sbin \
 -type f -perm /111 | \
 xargs -L1 basename | \
 xargs -L1 whatis | grep '([18])'



## **Other Systems**



## Applescript example

Is 10% of disk available?

<u>https://developer.apple.com/library/mac/documentation/applescript/conceptual/</u> <u>applescriptlangguide/conceptual/ASLR\_lexical\_conventions.html#//apple\_ref/doc/uid/</u> <u>TP40000983-CH214-SW1</u>

tell application "Finder"

set the percent\_free to ¬

(((**the** free space **of the** startup disk) / ¬

(the capacity of the startup disk)) \* 100) div 1

end tell

if the percent\_free is less than 10 then

**tell** application (path to frontmost application **as** text)

display dialog "The startup disk has only " &  $\neg$ 

the percent\_free & ¬

" percent of its capacity available." & **return & return** & ¬ "Should this script continue?" **with** icon 1

end tell

end if

## **PowerShell examples**

These examples from *Monad Manifesto* 

What is filling up my application logs?

Get-EventLog application Group source Select - first 5 Format-Table

Not text, but objects are passed around

## Why is Msilnstaller filling my log?

## Get-EventLog application |Where {\$\_.source -eq "MsiInstaller"}|Group Message |Select \_\_first 5 |Format-Table

counter Message

344 Detection of product '{90600409-6E45-45CA-BFCF-C1E1BEF5B3F7}... 344 Detection of product '{90600409-6E45-45CA-BFCF-C1E1BEF5B3F7}... 336 Product: Visual Studio.NET 7.0 Enterprise - English - Inter... 145 Failed to connect to server. Error: 0x800401F0 8 Product: Microsoft Office XP Professional with FrontPage --...

\_\_\_\_\_

Change Format-Table to output XML, CSV, LIST, HTML, Excel...

Is my eventlog usage regular across the week?

Get-EventLog application | Group
{\$\_.Timewritten.DayOfWeek}

counter DayofWeek

- 1,333 Tuesday
- 1,251 Wednesday
  - 744 Thursday
  - 680 Monday
  - 651 Friday
  - 556 Sunday
  - 426 Saturday

All these commands run in the same run-time environment (.NET) so I/O is cheap.

The shell can validate properties etc. using *reflection*, meaning it can look at what methods etc. are available.



- What is the least privilege principle?
- List a few pros and cons of shell scripting compared with other programming languages like C/C++.

## **Suggested Reading**

- The Art of Unix Programming Eric S. Raymond
- The Unix Hater's Handbook Simson Garfinkel, Daniel Weise, and Steven Strassmann
- Monad Manifesto Jeffrey P. Snover
- Scripting: Higher Level Programming for the 21st Century John K. Ousterhout (father of Tcl)

#### Bash Guide for Beginners Machtelt Garrels

[Reference] bash(1)