Unix2
Writing Csh Scripts

Slides:
bit.ly/unix2

Ynon Perek
ynon@ynonperek.com

Course Goals

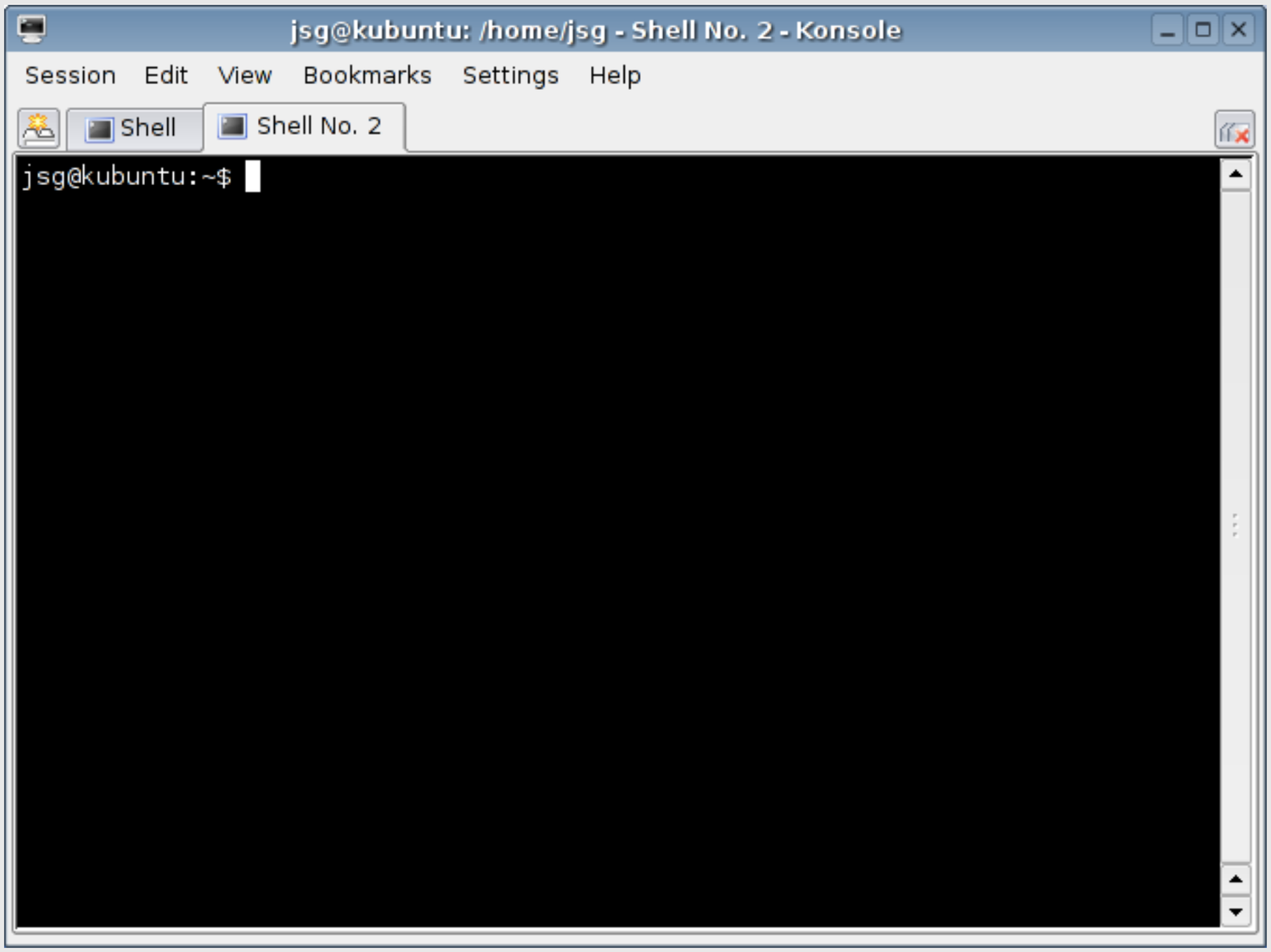
- At the end of the course, you'll
- Automate common tasks using shell scripts
- Customise your environment using rc files

Course Resources

- Labs: https://github.com/ynonp/csh-course/blob/master/labs/lab.md

- Extra Reading: http://star-www.rl.ac.uk/docs/sc4.htx/sc4.html

This Is Your Shell



Available Shells

- sh, bash
- ash
- csh, tcsh
- dash, ksh, zsh
- rush, fish
- and a lot more…

Shells vs. Programming

- Builtin file tests
- Builtin filesystem operations
- We’re already doing it

We'll Use Csh

- It's Easy to learn
- It's the default at Intel

Csh Alert

- Csh is not considered the best shell out there
- It's not backward compatible to sh
- Its parser is complex
- Other limitations: http://www.grymoire.com/unix/CshTop10.txt

Hello Shell

```
% echo "Hello Unix"
Hello Unix

% echo $0
tcsh

% echo $version
tcsh 6.17.00 (Astron) 2009-07-10 (x86_64-apple-darwin) options
wide,nls,dl,al,kan,sm,rh,color,filec

% echo $user
ynonperek
```

Shell Basics

- Wildcards
- Quoting
- Pipes & Filters

Wildcard Expansion

- If a word has a wildcard, csh will replace that word with a list of matching filenames
- Assume files a.txt, b.txt and c.txt are in the current folder

```
# Delete all .txt files
rm a.txt b.txt c.txt

# Same as
rm *.txt
```

Wildcard Expansion

Template	Matches
*	zero or more characters
?	exactly one character
[abcd]	exactly one character listed
[a-e]	exactly one character in range
[^a-egh]	exactly one character NOT in the specified range

nomatch

- Running a command with glob first checks the glob
- If nothing matches, command is not executed

```
% echo *.nomatch
echo: No match.
```

nomatch

- set nonomatch to run the command anyways

```
% set nonomatch
% echo *.nomatch
*.nomatch
```

Quoting

- Prevent parsing with single quotes

```
echo $user
ynonperek

echo '$user'
$user
```

Quoting

- Prevent wildcard parsing with double quotes

```
echo *.txt
a.txt b.txt

echo "*.txt"
*.txt
```

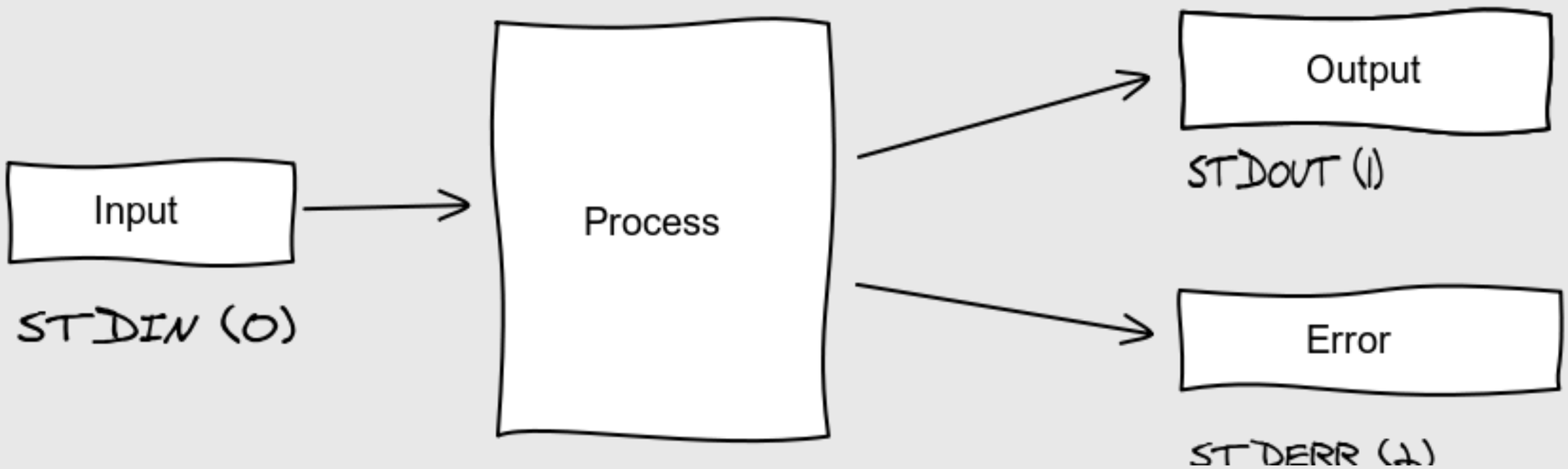
Command Substitution

- Use `command` to run a command and use its output as part of the command line

```
rm `cat oldfiles.txt`
```

Pipes & Filters

File Descriptors



Redirecting Output

```
# print to screen
% cowsay Hello

# print to file
% cowsay Hello > cow.art

# append to file
% cowsay Hello >> cow.art
```

Redirecting Errors

```
% touch main.c
% ls main.c nofile.txt
ls: nosuchfile.txt: No such file or
directory main.c

% ls main.c nofile.txt > filelist
ls: nosuchfile.txt: No such file
or directory
```

Redirecting Errors

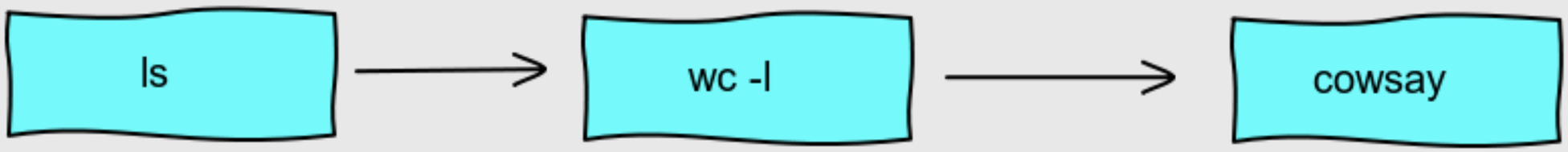
```
# send both STDOUT and STDERR to all
ls main.c nosuchfile.txt >& all

# split STDERR and STDOUT to two files
(ls main.c nosuchfile.txt > ok) >& error

# send errors to /dev/null printing only
# STDOUT
( ls main.c nosuchfile.txt > `tty` ) >& /dev/null
```

Clobbering

- Redirecting to files in scripts is dangerous
- Set noclobber to protect your files

```
touch file
set noclobber
echo > file

file: File exists.
```

Pipes




```
 ____
< 40 >
 ----
        \     ^    ^
         \      ___
          \    (oo)\_________
               (__)\         )\/\
                   ||----w   |
                   ||        ||
```

Q & A

Csh Scripts

Agenda

- Hello csh scripts
- Exit status
- Using variables
- If / Loop
- Arithmetics
- Arrays
- Woking with multiple files

Our First Shell Script

```
#!/bin/tcsh -f

echo All your base are belong to us
```

Our First Shell Script

- Use shbang line
- Add execute permission with **`chmod +x hello.csh`**
- Run script with **`./hello.csh`**

Setting Variables

- Assign values using set command
- Use quotes for multiple words

```
set foo = 10
set bar = "hello world"
set buz = 'show me the $$$'
```

Variable Types

- Strings:

```
% set a = 01
% echo $a

01
```

Variable Types

- Lists:

```
% set a = (foo bar buz)
% echo $a[1]
foo

% echo $#a
3

% echo $a[$#a]
buz
```

Variable Types

- Lists:

```
% set b = (foo bar buz)
% echo $b[*]
foo bar buz

% echo $b[2-]
bar buz

% echo $b[1-2]
foo bar
```

Variable Modifiers

```
% set name=/home/ynon/demo.png

% echo $name:h
/home/ynon

% echo $name:t
demo.png

% echo $name:r
/home/ynon/demo

% echo $name:r:t
demo
```

Variable Modifiers

```
% set uc=HELLO
% set lc=hello

% echo $uc:l
hELLO

% echo $uc:al
hello

% echo $lc:u
Hello

% echo $lc:au
HELLO
```

Variable Modifiers

```
% set name="I can haz cheezburger"

% echo $name:s/ /_/
I_can haz cheezburger

% echo $name:as/ /_/
I_can_haz_cheezburger
```

Variable Info

- Use ${%…} to get the length of a variable

```
% set name = 'lil bub'

% echo ${%name}
7
```

Script Parameters

- Pass data to script using command line arguments
- Inside script use $1, $2, $3, … to access the data
- Parameter count is saved in $#
- All parameters is saved in $* (which is also called $argv)

Reading User Input

- The special variable $< returns a single line of text from the user

- Save it in a named variable with the command set

```
#!/bin/tcsh -f

echo "Who are you?"
set name = "$<"

echo "Welcome, $name"
```

Lab: Getting Parameters

Csh If

```
#!/bin/tcsh -f

echo "Who are you?"
set name = "$<"

if ( $name =~ [A-Z]* ) then
  echo "That's a lovely name"
endif
```

If Structure

- Use `if ( … ) then … endif` for conditional blocks
- Use `if ( … ) command` for single line if
- Can add else blocks

Csh Expressions

- Inside if we can use csh special expressions:
- Logical, arithmetic and comparison
- Command exit status
- File inquiry operators

Operators

- These are all supported:

```
|| && | ^ & == != =~ !~ <= >=
< > << >> + - * / % ! ~ ( )
```

- Result is a string, but operands are numbers

Command Exit Status

```
#!/bin/tcsh -f

echo "What are you looking for?"
set text = "$<"

if ({ grep "$text" /etc/passwd }) then
  echo "Found It"
else
  echo "Sorry, text not found"
endif
```

Exit Status

- Each program has an exit status
- 0 => success
- anything else => failure
- Value is saved in $status (or $?)

Exit Status

- Set exit status from shell script

```
#!/bin/tcsh -f

echo "hello"

# fail
exit 7
```

Exit Status

- Set exit status from c program

```c
#include <stdio.h>

int main(int argc, char **argv)
{
  printf("Hello World");

  // fail
  return 7;
}
```

Checking Exit Status

```
#!/bin/tcsh -f

% grep foo nosuchfile
grep: nosuchfile: No such file or directory

% echo $?
2

% grep foo /etc/passwd
% echo $?
1
```

Exit Status and If

- Use {…} to run a command and branch according to its exit status
- Success => then block
- Failure => else block

Redirections

- Redirections don’t work inside if

- For noisy commands use $? explicitly

Redirections

```
#!/bin/tcsh -f

grep root /etc/passwd >& /dev/null

if ( $? == 0 ) then
  echo "Found root in /etc/passwd"
else
  echo "root not found"
endif
```

File Inquiry Operators

```
#!/bin/tcsh -f

if ( -r /etc/passwd ) then
  echo "/etc/passwd is readable"
endif
```

Other File Tests

Operator	Meaning
-r, -w, -x	readable, writable, executable
-X	executable in the path or shell builtin
-z / -s file	file is empty / non-empty
-f, -d, -l	path is a file, a dir or a link

File Info

Operator	Meaning
-M	Last file modification time
-L	Name of the file pointed to by a symbolic link
-P	file permissions (octal)
-Z	file size (in bytes)

Q & A

Csh Switch/Case

```
#!/bin/tcsh -f

switch ($1)
  case *.png:
     echo "It's a png image"
     breaksw

 default:
     echo "It's another file"
endsw
```

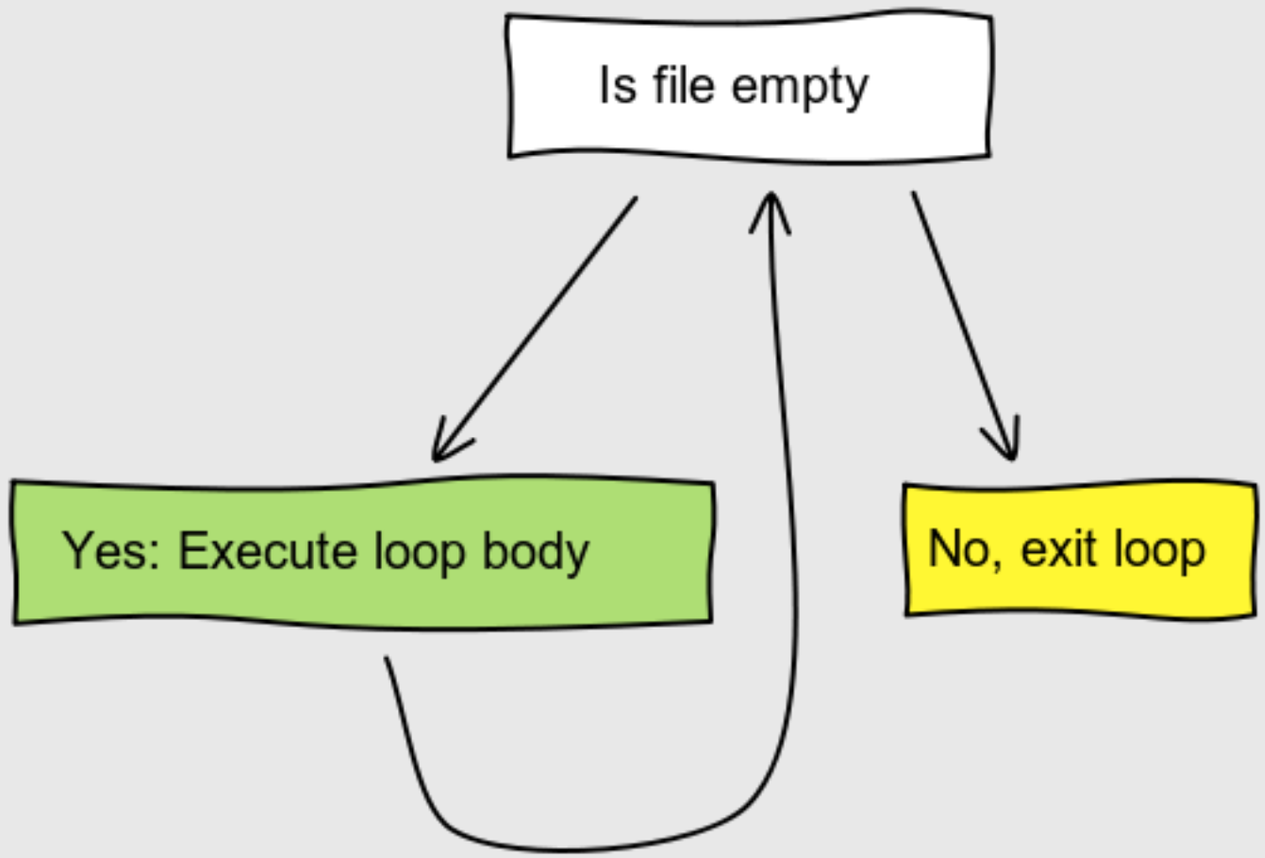
Useful Commands

- `realpath` prints the real path (from relative)
- `repeat` executes a command n times

Lab: Part 3 Conditionals

Csh Loops

Let’s start with while



while syntax

```
#!/bin/tcsh -f

while ( -f /tmp/run )
   echo File /tmp/run still exists.
   sleep 1
end
```

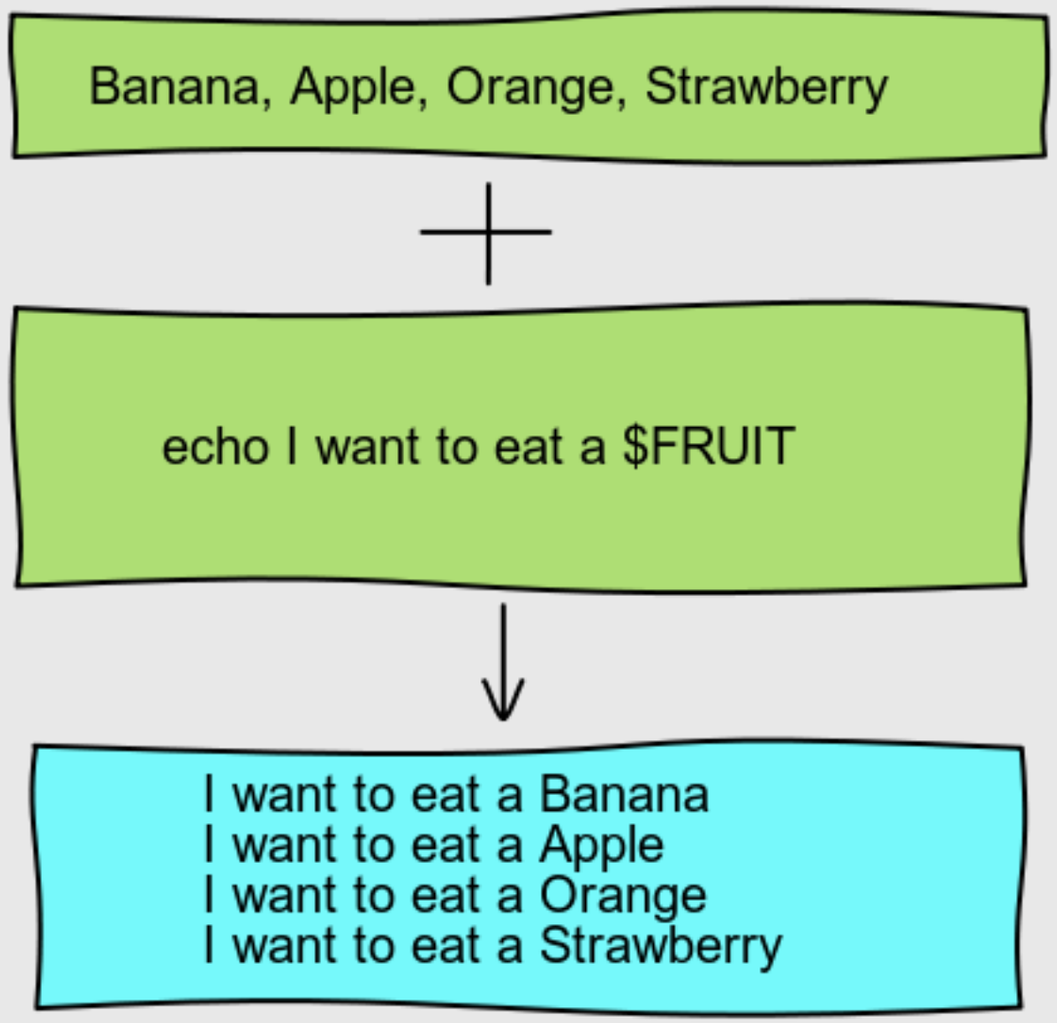
while expressions

- All if expressions work:
- Arithmetics and logic
- File tests
- Command exit status

while user is logged in

```
while ( 1 )
   who | grep jimmy >& /dev/null
   if ( $? != 0 ) then
      # grep failed => user not found
      break
   endif

   sleep 2
end

echo bye bye
```

foreach loops



foreach loops

```
#!/bin/tcsh -f

set fruits = (apple banana orange)

foreach item ($fruits)
  echo "I can has $item"
end
```

foreach file

```
#!/bin/tcsh -f

foreach file (*.txt)
  cp "$file" "${file}.old"
end
```

foreach argument

- Arguments with spaces are not handled here

```
#!/bin/tcsh -f


foreach arg ($argv)
  echo "Got: $arg"
end
```

foreach line

```
#!/bin/tcsh -f

foreach line ("`cat /etc/shells`")
  echo "> $line"
end
```

Arithmetics

- Use @ to evaluate a numeric expression in csh

```
@ name = expr
@ name++
@ name--
```

Arithmetics

- Count how many text files have more than 10 lines

```
set files = 0

foreach txtfile (*.txt)
  echo checking $txtfile
  set lines=`wc -l < "$txtfile"`
  echo lines = $lines
  if ( $lines > 10 ) then
    @ files++
  endif
end

echo "$files files have more than 10 lines"
```

Q & A

Lab: Part 4

Loops

Parsing Arguments

- Let's write a script that handles command line arguments

- The script should:
  - Print "hello!"
  - If passed "-c" it should also print "nice to meet you"
  - If passed "-d" it should also print "I can has cheezburger"

Parsing Arguments

```
#!/bin/tcsh -f

echo "Hello!"

foreach arg ($argv)
  switch ($arg)
    case -c:
      echo "nice to meet you"
      breaksw

    case -d:
      echo "I can has cheezburger"
      breaksw
  endsw
end
```

Parsing Arguments

- But what if you tried to run

```
./hello -cd
```

getopt to the rescue

- getopt normalises command line arguments

```
% getopt cd -c -d
 -c -d --

% getopt cd -cd
 -c -d --
```

getopt demo

- Normalise with getopt

- Parse with foreach/switch loop

getopt demo

```
#!/bin/tcsh -f

echo "Hello!"

foreach opt (`getopt cd $argv`)
  switch ($opt)
    case -c:
      echo "nice to meet you"
      breaksw

    case -d:
      echo "I can has cheezburger"
      breaksw
  endsw
end
```

getopt + normal objects

- Many commands take both switches and normal objects
- To parse the “remaining” objects:
  - Take the output of getopt
  - shift it per each switch

getopt + normal objects

```
#!/bin/tcsh -f

set count=1

set remaining = (`getopt cd $argv`)
foreach opt (`getopt cd $argv`)
  switch ($opt)
    case -c:
      echo "nice to meet you"
      @ count++
      breaksw

  endsw
end

repeat $count shift remaining
echo $remaining
```

Q & A

Lab: Part 5

Getopt

Scheduled Tasks

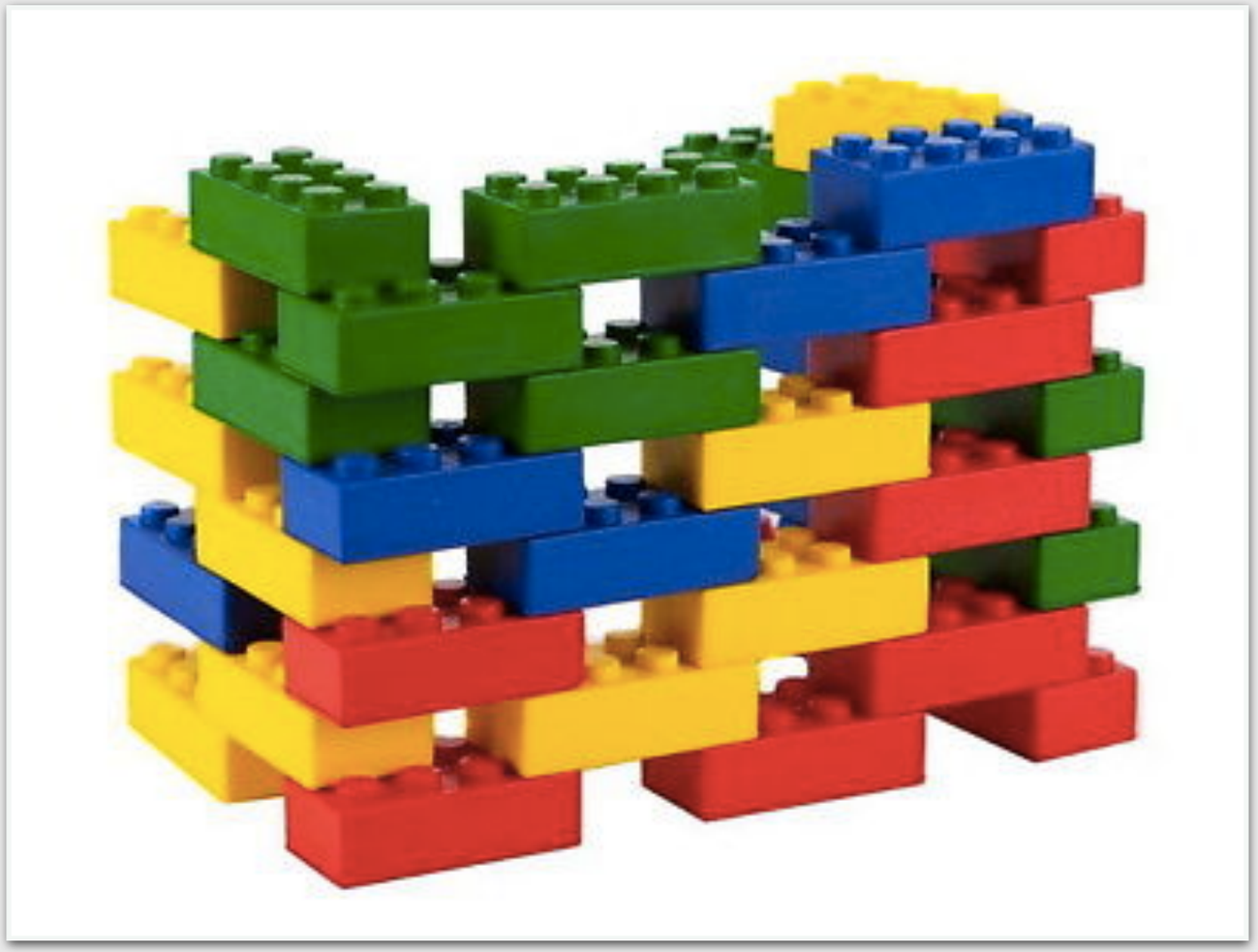
- Use crontab to schedule a repeating task
- Edit tasks with:
  - `crontab -e`
- View tasks with:
  - `crontab -l`

Scheduled Tasks

```
######################
# Cron fields
#   minute          0-59
#   hour            0-23
#   day of month    1-31
#   month           1-12
#   day of week     0-7 (0 or 7 is Sun)
#

# Backup the server daily
0 0 * * * /bin/backup.sh

# Backup the database every hour
0 * * * * /bin/backup_db.sh

# Send email once a week
0 0 * * 4 /bin/send_report.sh

# Check data every 5 minutes
0-59/5 * * * * /bin/check_data.sh
```

Multiple Script Files

- Like Lego, shell scripts are small and composable



Multiple Script Files

- Use **`./script`** to run an external script

- Use **`source script`** to “include” external file

source demo

common.csh

```
set name  = 'ynon'
set email = 'ynon@ynonperek.com'
set blog  = 'www.tocode.co.il/blog'
```

demo.csh

```
#!/bin/tcsh -f

source common.csh

echo welcome, $name
```

Things You Can Source

- cd
- alias
- set / setenv

source and arguments

- When passing arguments to source they are sent to sourced file as its $1, $2, $3
- $0 is the original file

```
source common.csh 10 20
```

source and arguments

- Source without arguments sends $*
- $0 is still the original file

```
source common.csh
```

Q & A

Named Pipes

Named Pipes

- A simple IPC mechanism based on files
- Processes read from or write to the pipe

Create a named pipe

- Use mkfifo to create a pipe

- `mkfifo pipename`

Working with pipes

Process A:

```
echo hello world > pipe
```

Process B:

```
cat pipe
```

Use named pipes

- Control long running scripts
- Write simple “services”

Lab: Part 6 Named Pipes

Thanks For Listening

- Ynon Perek
- ynon@ynonperek.com
- www.tocode.co.il/blog