EECS 2031 E 3.0

Software Tools

Week 11: November 21, 2018

Arrays

- Infrequently used, but they exist in bash
- Can declare them explicitly
  - declare -a foo=(a b c)
- Or implicitly
  - foo=(a b c)
- ${foo[2]} - element 2
- ${foo[@]} - all of foo
- ${#foo[@]} - size of foo
- Arrays are 0 offset

```
#!/bin/bash
foo=(monday tuesday wednesday thursday friday saturday sunday)
echo "${foo[0]}"
echo "${foo[1]}"
echo "${foo[2]}"
echo "${foo[3]}"
echo "${foo[@]}"
echo "${#foo[@]}"
```

```
wanderereecsyorkuca:t jenkin$ ./a.sh
monday
tuesday
wednesday
thursday
monday tuesday wednesday thursday friday saturday sunday
7
```

Arrays

- Array elements can be null (so they will not print)
  - Unset foo[2]
  - foo[0]=

```
#!/bin/bash
foo=(monday tuesday wednesday thursday friday saturday sunday)
echo "${foo[@]}"
unset foo[1]
echo "${foo[@]}"
foo[0]=
echo "${foo[@]}"
```

Arrays

- Can create arrays from arrays
- Can select parts of arrays
  - ${foo[@]:2:1}

```
#!/bin/bash
foo=(monday tuesday wednesday thursday friday saturday sunday)
echo "${foo[@]}"
foo=(${foo[@]} holiday extraday)
echo "${foo[@]}"
foo=(${foo[@]:3:2})
echo "${foo[@]}"
```

```
wanderereecsyorkuca:t jenkin$ ./a.sh
monday tuesday wednesday thursday friday saturday sunday
monday tuesday wednesday thursday friday saturday sunday holiday extraday
thursday friday
```

Bash and subshells

- It is easy (too easy) to generate subshells in bash
  - Invoke another command
  - Use a programming construct which bash implements by using subshells
- In either event you need to be aware that the (sub) shell will have its own local variables that will vanish when the sub-shell is exited

Bash and sub-shells

- If you put a command in () it is executed in a subshell.

```
#!/bin/bash
x=7
echo before $x
(echo "in parenthesis $x";x=8;echo "in parenthesis $x")
echo "after $x"
```

```
wanderereecsyorkuca:t jenkin$ ./d.sh
before 7
in parenthesis 7
in parenthesis 8
after 7
```

Bash and sub-shells

- You can cause Bash to spawn subshells whenever you pipe the output of a command

```
#!/bin/bash
x=7
echo before $x
for i in 1 2 3 4 5 6 7 8; do
  let "x=$x+1"
  echo "inside $x"
done
echo "after $x"
```

```
./d.sh
before 7
inside 8
inside 9
inside 10
inside 11
inside 12
inside 13
inside 14
inside 15
after 15
```

```
wanderereecsyorkuca:t jenkin$ cat d.sh
#!/bin/bash
x=7
echo before $x
for i in 1 2 3 4 5 6 7 8; do
  let "x=$x+1"
  echo "inside $x"
done | cat >/dev/null
echo "after $x"
```

```
./d.sh
before 7
after 7
```

Bash and subshells

```
#!/bin/bash
x=7
echo before $x
for i in 1 2 3 4 5 6 7 8; do
  let "x=$x+1"
  echo "inside $x"
done >/dev/null
echo "after $x"
```

```
wanderereecsyorkuca:t jenkin$ ./d.sh
before 7
after 15
```

Summary

- Bash - a CLI (shell) based on sh before it
  - There are other shells. Bash is free so commonly used.
- Supports standard programming language constructs, untyped variables (int, string) and arrays
- Supports functions and the ability to invoke other programs (including other bash programs)
- Utilizes value of exit (very unix-friendly) to pass a single small value integer between processes
- Variables are ‘complex’ in that different programming features can lead to the spawning/use of subshells or separate processes with their own namespace.

Rest of the course

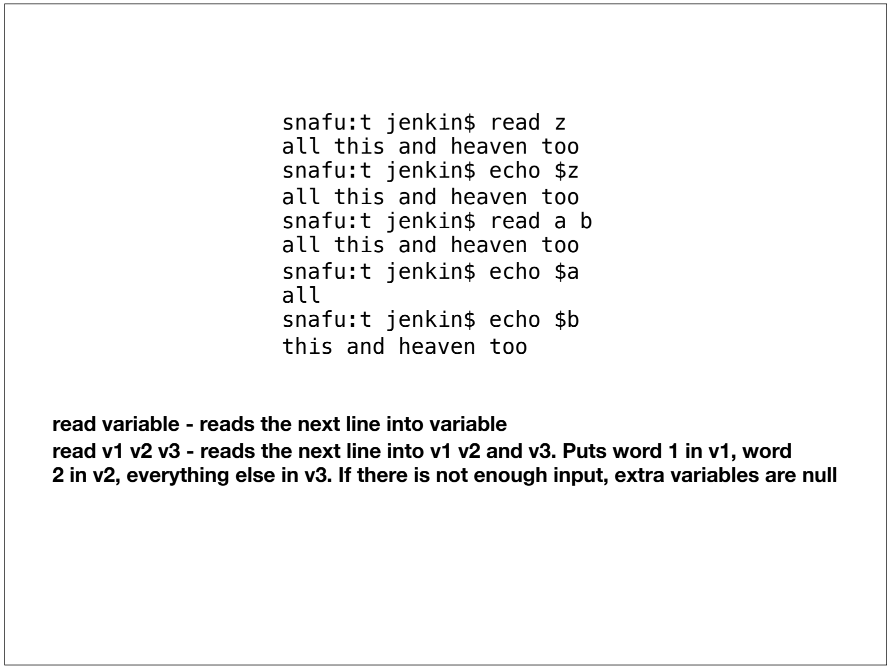
- Advanced topics the rest of today
- Lab 10 next week (labs)
- Next week, last quiz, the entire course in 2 hours
- Lab 10 cannot be handed in during the lab test. But at the last office hours the following week.

Advanced topics in Bash

- Yes, on the last lab test.
- Yes, on the final.

read

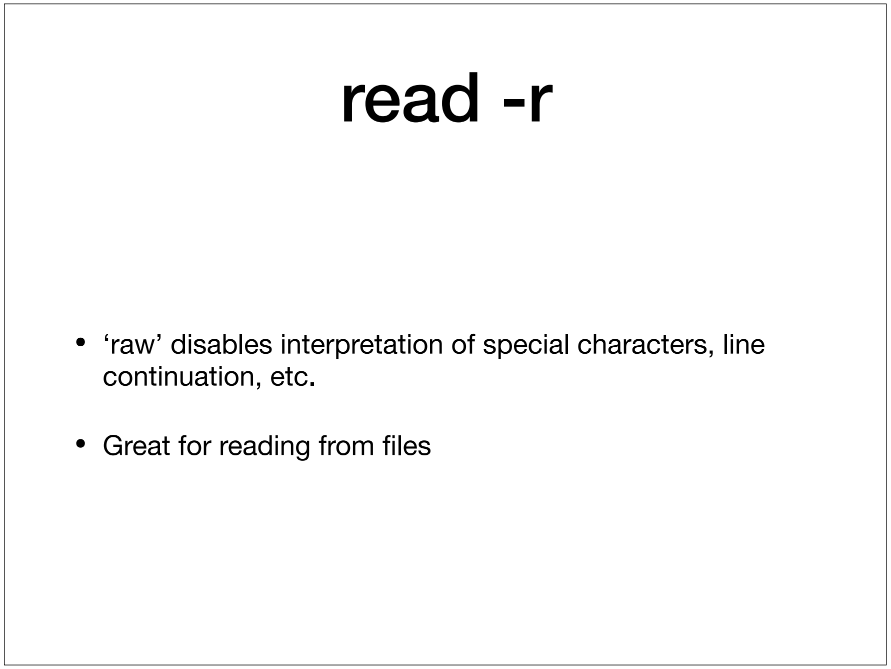
- read is a builtin command that (d’oh) reads from the standard input.
- It has a large number of options
  - RTFM



```
snafu:t jenkin$ read z
all this and heaven too
snafu:t jenkin$ echo $z
all this and heaven too
snafu:t jenkin$ read a b
all this and heaven too
snafu:t jenkin$ echo $a
all
snafu:t jenkin$ echo $b
this and heaven too
```

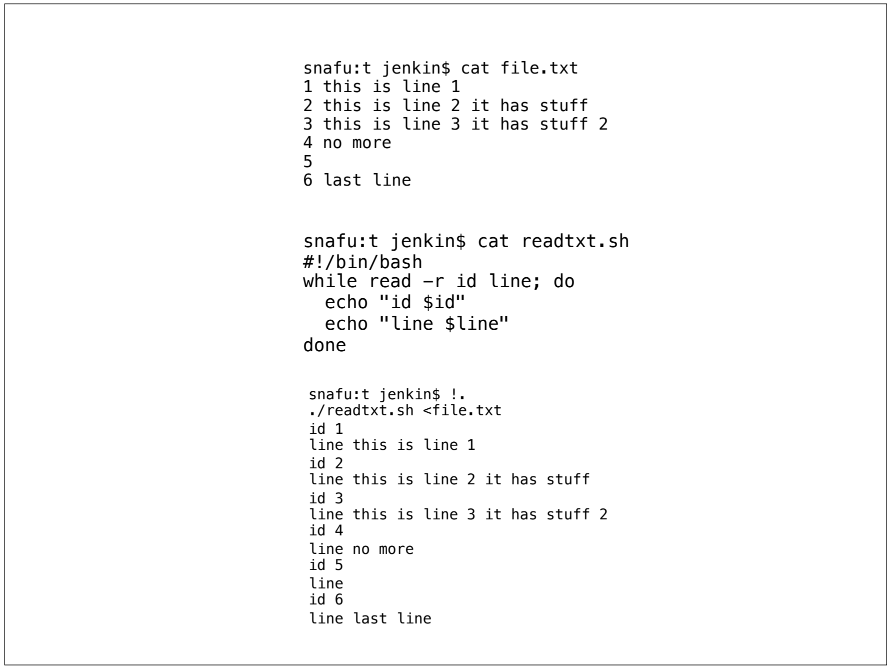
read variable - reads the next line into variable

read v1 v2 v3 - reads the next line into v1 v2 and v3. Puts word 1 in v1, word 2 in v2, everything else in v3. If there is not enough input, extra variables are null



read -r

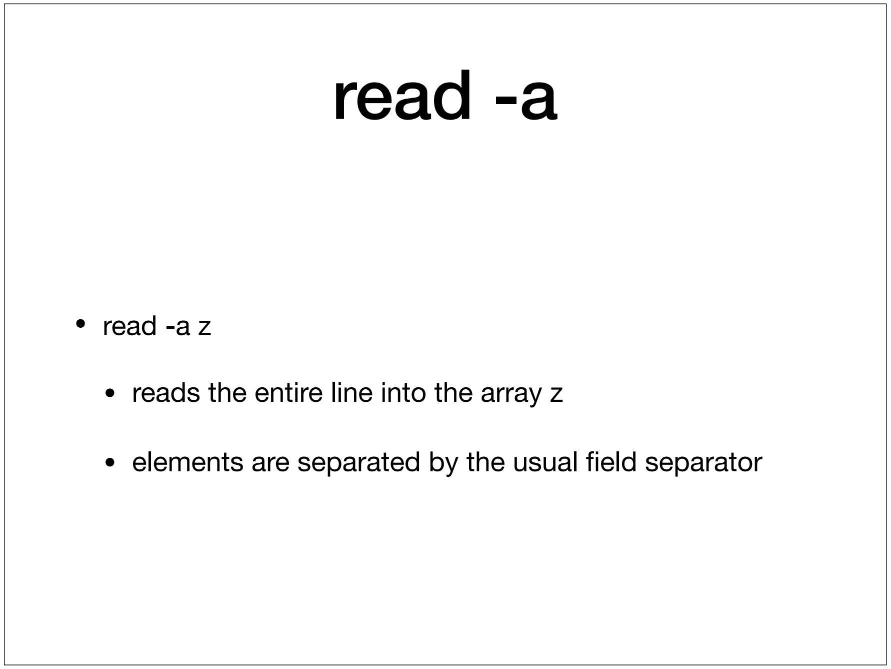
- ‘raw’ disables interpretation of special characters, line continuation, etc.
- Great for reading from files



```
snafu:t jenkin$ cat file.txt
1 this is line 1
2 this is line 2 it has stuff
3 this is line 3 it has stuff 2
4 no more
5
6 last line
```

```
snafu:t jenkin$ cat readtxt.sh
#!/bin/bash
while read -r id line; do
  echo "id $id"
  echo "line $line"
done
```

```
snafu:t jenkin$ !.
./readtxt.sh <file.txt
id 1
line this is line 1
id 2
line this is line 2 it has stuff
id 3
line this is line 3 it has stuff 2
id 4
line no more
id 5
line
id 6
line last line
```



read -a

- read -a z
  - reads the entire line into the array z
  - elements are separated by the usual field separator

```
snafu:t jenkin$ cat readtxt.sh
#!/bin/bash
while read -r -a tokens ; do
  echo "there were ${#tokens[@]} tokens"
  for z in ${tokens[@]}; do
    echo "$z"
  done
done
```

```
snafu:t jenkin$ cat file.txt
1 this is line 1
2 this is line 2 it has stuff
3 this is line 3 it has stuff 2
4 no more
5
6 last line
```

```
./readtxt.sh <file.txt
there were 5 tokens
1
this
is
line
1
there were 8 tokens
2
this
is
line
2
it
has
stuff
there were 9 tokens
3
this
is
line
3
it
has
stuff
2
there were 3 tokens
4
no
more
there were 1 tokens
5
there were 3 tokens
6
last
line
```

IFS

- IFS is the Internal field separator. Its how Bash recognizes the boundaries between fields
    - Normally its white space (tabs, blanks, new lines)
- But we can set it to other things to modify the way in which the shell works
- So lets play with read and IFS

Comma separated files

- Common way of turning spreadsheets into text
- If we could only take text files and turn them into things separated by comma's we'd be good to go, but that's difficult to do.
- Easier - get read to treat the separator as a , rather than a white space

```
snafu:t jenkin$ cat readcsv.sh
#!/bin/bash
IFS=,
while read -r -a x; do
  echo "Line" ${x[@]}
  for z in ${x[@]}; do
    echo "item " $z
  done
done
```

```
a,b,c,123
4, 5,6 123,
all this
too
```

```
snafu:t jenkin$ ./readcsv.sh <foo.csv
Line a b c 123
item  a
item  b
item  c
item  123
Line 4  5 6 123
item  4
item   5
item  6 123
Line all this
item  all this
Line too
item  too
```

Setting IFS

- You can do it anywhere (and live with the consequences).
- If you want to set it and unset it
    - Spawn a sub shell
    - Save the old value and restore it afterwards

Another example

- Randomizing lines in a file
- Given a printable text file, generate another text file such that
    - Both files contain the same set of lines
    - The orders are different (2nd is randomized, permuted version of first)
- To make it easier, assume | does not appear in the file

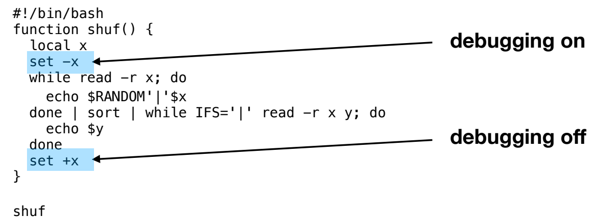
```
#!/bin/bash
function shuf() {
  local x
  while read -r x; do
    echo $RANDOM'|'$x
  done | sort | while IFS='|' read -r x y; do
    echo $y
  done
}

shuf
```

```
snafu:t jenkin$ cat file.txt
1 this is line 1
2 this is line 2 it has stuff
3 this is line 3 it has stuff 2
4 no more
5
6 last line
snafu:t jenkin$ !.
./randomize.sh <file.txt
6 last line
4 no more
5
1 this is line 1
2 this is line 2 it has stuff
3 this is line 3 it has stuff 2
snafu:t jenkin$ !.
./randomize.sh <file.txt
1 this is line 1
6 last line
2 this is line 2 it has stuff
4 no more
3 this is line 3 it has stuff 2
5
snafu:t jenkin$
```

Debugging bash scripts

- /bin/bash -x foo.sh
- set -x and set +x

```
snafu:t jenkin$ /bin/bash -x ./randomize.sh  <file.txt
+ shuf
+ local x
+ read -r x
+ echo '7900|1' this is line 1
+ read -r x
+ echo '22174|2' this is line 2 it has stuff
+ read -r x
+ echo '31100|3' this is line 3 it has stuff 2
+ read -r x
+ sort
+ echo '32368|4' no more
+ read -r x
+ echo '6175|5'
+ read -r x
+ echo '28754|6' last line
+ read -r x
+ IFS='|'
+ read -r x y
+ echo 2 this is line 2 it has stuff
2 this is line 2 it has stuff
+ IFS='|'
+ read -r x y
+ echo 6 last line
6 last line
+ IFS='|'
+ read -r x y
+ echo 3 this is line 3 it has stuff 2
3 this is line 3 it has stuff 2
+ IFS='|'
+ read -r x y
+ echo 4 no more
4 no more
+ IFS='|'
+ read -r x y
+ echo 5
5
+ IFS='|'
+ read -r x y
+ echo 1 this is line 1
1 this is line 1
+ IFS='|'
+ read -r x y
```



```
#!/bin/bash
function shuf() {
  local x
  set -x
  while read -r x; do
    echo $RANDOM'|'$x
  done | sort | while IFS='|' read -r x y; do
    echo $y
  done
  set +x
}

shuf
```

debugging on

debugging off

Chaining commands

```
indigo 305 % false && echo "hello"
indigo 306 % true && echo "hello"
hello
indigo 307 % false||echo "hello"
hello
indigo 308 % true||echo "hello"
indigo 309 %
```

source and .

```
#!/bin/bash
echo "in A"
y=hello
x=world
echo "before x is $x"
echo "before y is $y"
./b.sh
echo "after x is $x"
echo "after y is $y"

indigo 342 % cat b.sh
#!/bin/bash
echo "in B"
y=hello
x=moon
echo "in b x is $x"
echo "in b y is $y"
```

```
indigo 343 % a.sh
in A
before x is world
before y is hello
in B
in b x is moon
in b y is hello
after x is world
after y is hello
```

```
#!/bin/bash
echo "in A"
y=hello
x=world
echo "before x is $x"
echo "before y is $y"
. ./b.sh
echo "after x is $x"
echo "after y is $y"
```

```
indigo 350 % a.sh
in A
before x is world
before y is hello
in B
in b x is moon
in b y is hello
after x is moon
after y is hello
```

So lets look at something more complex

- When linux boots, it must configure the system and people (administrators) must be able to do that configuration.
- At a point a fair way along the process, linux looks in /etc/rc.* directories and executes things there.
- Almost all are shell scripts.
- Lets look at one

```
#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.

# Print the IP address

_IP=$(hostname -I) || true
if [ "$_IP" ]; then
  printf "My IP address is %s\n" "$_IP"
fi
```

Execute hostname -I and set value to _IP

```
#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.

# Print the IP address

_IP=$(hostname -I) || true
if [ "$_IP" ]; then
  printf "My IP address is %s\n" "$_IP"
fi
```

Throw away any bad return value

```
#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.

# Print the IP address

_IP=$(hostname -I) || true
if [ "$_IP" ]; then
  printf "My IP address is %s\n" "$_IP"
fi
```

If _IP has a valid value (not null)

```
#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.

# Print the IP address

_IP=$(hostname -I) || true
if [ "$_IP" ]; then
  printf "My IP address is %s\n" "$_IP"
fi
```

Fancy printing

Export

- Builtin function of bash
  - Exports variables from one instance of bash to another
  - Variable is available to any process you run from this shell
- Normally used for ‘global’ state variables

Export usage

- export variable
  - Exports the variable
- export
  - Displays all exported variables

```
declare -x EDITOR="vi"
declare -x GROUP="faculty"
declare -x HOME="/cs/home/jenkin"
declare -x HOST="indigo"
declare -x HOSTNAME="indigo"
declare -x HOSTTYPE="x86_64-linux"
declare -x KDEDIRS="/usr"
declare -x LANG="en_US.UTF-8"
declare -x LD_LIBRARY_PATH="/cs/home/ivy/ve-2.2/lib"
declare -x LESSOPEN="||/usr/bin/lesspipe.sh %s"
declare -x LOADEDMODULES=""
declare -x LOGNAME="jenkin"
declare -x
LS_COLORS="rs=0:di=38;5;27:ln=38;5;51:mh=44;38;5;15:pi=40;38;5;11:so=38;5;13:do=38;5;5:bd=48;5;232;38;5;11:cd=48;5;232;38;5;3:or=48;5;232;38;5;9:mi=05;4
8;5;232;38;5;15:su=48;5;196;38;5;15:sg=48;5;11;38;5;16:ca=48;5;196;38;5;226:tw=48;5;10;38;5;16:ow=48;5;10;38;5;21:st=48;5;21;38;5;15:ex=38;5;34:*.tar=38
;
5;9:*.tgz=38;5;9:*.arc=38;5;9:*.arj=38;5;9:*.taz=38;5;9:*.lha=38;5;9:*.lz4=38;5;9:*.lzh=38;5;9:*.lzma=38;5;9:*.tlz=38;5;9:*.txz=38;5;9:*.tzo=38;5;9:*.t7
z=38;5;9:*.zip=38;5;9:*.z=38;5;9:*.Z=38;5;9:*.dz=38;5;9:*.gz=38;5;9:*.lrz=38;5;9:*.lz=38;5;9:*.lzo=38;5;9:*.xz=38;5;9:*.bz2=38;5;9:*.bz=38;5;9:*.tbz=38;
5;9:*.tbz2=38;5;9:*.tz=38;5;9:*.deb=38;5;9:*.rpm=38;5;9:*.jar=38;5;9:*.war=38;5;9:*.ear=38;5;9:*.sar=38;5;9:*.rar=38;5;9:*.alz=38;5;9:*.ace=38;5;9:*.zoo
=38;5;9:*.cpio=38;5;9:*.
7z=38;5;9:*.rz=38;5;9:*.cab=38;5;9:*.jpg=38;5;13:*.jpeg=38;5;13:*.gif=38;5;13:*.bmp=38;5;13:*.pbm=38;5;13:*.pgm=38;5;13:*.ppm=38;5;13:*.tga=38;5;13:*.xb
m=38;5;13:*.xpm=38;5;13:*.tif=38;5;13:*.tiff=38;5;13:*.png=38;5;13:*.svg=38;5;13:*.svgz=38;5;13:*.mng=38;5;13:*.pcx=38;5;13:*.mov=38;5;13:*.mpg=38;5;13:
*.mpeg=38;5;13:*.m2v=38;5;13:*.mkv=38;5;13:*.webm=38;5;13:*.ogm=38;5;13:*.mp4=38;5;13:*.m4v=38;5;13:*.mp4v=38;5;13:*.vob=38;5;13:*.qt=38;5;13:*.nuv=38;5
;
13:*.wmv=38;5;13:*.asf=38;5;13:*.rm=38;5;13:*.rmvb=38;5;13:*.flc=38;5;13:*.avi=38;5;13:*.fli=38;5;13:*.flv=38;5;13:*.gl=38;5;13:*.dl=38;5;13:*.xcf=38;5;
13:*.xwd=38;5;13:*.yuv=38;5;13:*.cgm=38;5;13:*.emf=38;5;13:*.axv=38;5;13:*.anx=38;5;13:*.ogv=38;5;13:*.ogx=38;5;13:*.aac=38;5;45:*.au=38;5;45:*.flac=38;
5;45:*.mid=38;5;45:*.midi=38;5;45:*.mka=38;5;45:*.mp3=38;5;45:*.mpc=38;5;45:*.ogg=38;5;45:*.ra=38;5;45:*.wav=38;5;45:*.axa=38;5;45:*.oga=38;5;45:*.spx=3
8;5;45:*.xspf=38;5;45:"
declare -x MACHTYPE="x86_64"
declare -x MAIL="/var/spool/mail/jenkin"
declare -x MANPATH="/cs/home/jenkin/man:/cs/local/man:/cs/fac/man:/cs/local/share/man:/usr/local/man:/usr/local/share/man:/usr/share/man"
declare -x MODULEPATH="/usr/share/Modules/modulefiles:/etc/modulefiles"
declare -x MODULESHOME="/usr/share/Modules"
declare -x OLDPWD
declare -x OSTYPE="linux"
declare -x PAGER="/cs/local/bin/less"
declare -x PATH="/cs/home/jenkin/bin:/cs/local/bin:/cs/fac/bin:/cs/local/packages/xfce/bin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/usr/X11R6/
bin:."
```

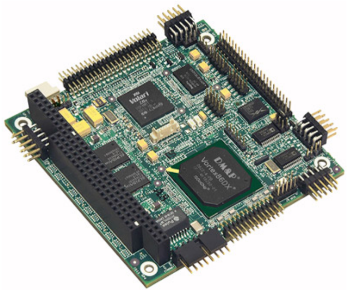
Partial list

C, Bash and embedded systems

- Perhaps most common place that you end up needing languages like C is when you interact with low level infrastructure.
- Large number of commercial and recreational systems like this.

PC104

- Refers to a form factor (104).
- Very much like traditional computer (Intel) hardware.
- Additional support for hardware.



In the 'hobby' space

- Raspberry Pi <- you know all about this now
- Beagleboard
- Others
- Typically designed (initially) for the hobby market but have been re-purposed elsewhere.

All run C, all have Bash

- Almost all have reasonably high level libraries to access output pins on the device.
- WiringPi/GPIO - provides low level access

At a higher level

- Almost all devices speak a protocol at a higher level than just turning a wire on or off.
- There are a number of standard protocols for talking to external devices

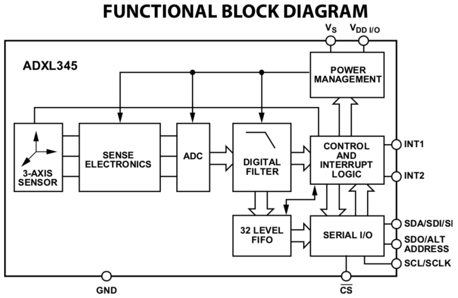


Figure 1.

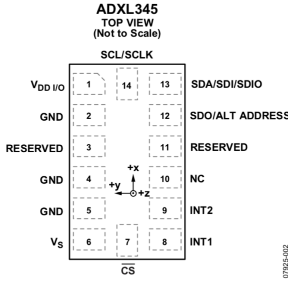


Figure 2. Pin Configuration

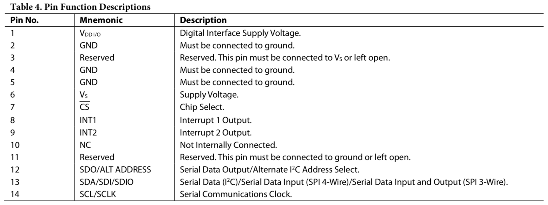
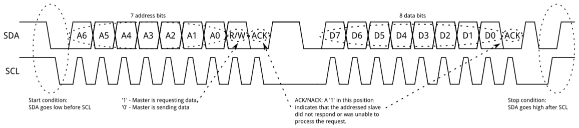


Table 4. Pin Function Descriptions

Pin No.	Mnemonic	Description
1	$V_{DD\ I/O}$	Digital Interface Supply Voltage.
2	GND	Must be connected to ground.
3	Reserved	Reserved. This pin must be connected to V_S or left open.
4	GND	Must be connected to ground.
5	GND	Must be connected to ground.
6	V_S	Supply Voltage.
7	$\overline{CS}$	Chip Select.
8	INT1	Interrupt 1 Output.
9	INT2	Interrupt 2 Output.
10	NC	Not Internally Connected.
11	Reserved	Reserved. This pin must be connected to ground or left open.
12	SDO/ALT ADDRESS	Serial Data Output/Alternate I^2C Address Select.
13	SDA/SDI/SDIO	Serial Data (I^2C)/Serial Data Input (SPI 4-Wire)/Serial Data Input and Output (SPI 3-Wire).
14	SCL/SCLK	Serial Communications Clock.

So wiring

- I2C devices need power and GND (3.3V, usually)
- Have a serial protocol (I2C) running over two lines SDA and SCL.
- Sophisticated protocol allowing multiple units to communicate on the same bus.


7 address bits
8 data bits
SDA
SCL
A6
A5
A4
A3
A2
A1
A0
R/W
ACK
D7
D6
D5
D4
D3
D2
D1
D0
ACK
Start condition:
SDA goes low before SCL
'1' - Master is requesting data
'0' - Master is sending data
ACK/NACK: A '1' in this position indicates that the addressed slave did not respond or was unable to process the request.
Stop condition:
SDA goes high after SCL

But generally you don't have to care

- People have written libraries that talk to the devices
- People have written standard tools to interact with devices.

```
//----- OPEN THE I2C BUS -----
char *filename = (char*)"/dev/i2c-1";
if ((file_i2c = open(filename, O_RDWR)) < 0)
{
    //ERROR HANDLING: you can check errno to see what went wrong
    printf("Failed to open the i2c bus");
    return;
}

int addr = 0x5a;          //<<<<<The I2C address of the slave
if (ioctl(file_i2c, I2C_SLAVE, addr) < 0)
{
    printf("Failed to acquire bus access and/or talk to slave.\n");
    //ERROR HANDLING; you can check errno to see what went wrong
    return;
}


//----- READ BYTES -----
length = 4;            //<<< Number of bytes to read
if (read(file_i2c, buffer, length) != length) {
    //ERROR HANDLING: i2c transaction failed
    printf("Failed to read from the i2c bus.\n");
}
else
{
    printf("Data read: %s\n", buffer);
}


//----- WRITE BYTES -----
buffer[0] = 0x01;
buffer[1] = 0x02;
length = 2;            //<<< Number of bytes to write
if (write(file_i2c, buffer, length) != length)        {
    /* ERROR HANDLING: i2c transaction failed */
    printf("Failed to write to the i2c bus.\n");
}
```

Software development

- C is but one of a number of potential programming languages.
  - “Every tool is a hammer, except a screwdriver. Its a chisel.”
- Choose the right tool for the right job.

C

- Low memory footprint
- Close to the hardware
- Very strong control over actions in the language
- Poor set of standard libraries
- Poor set of standard datatypes
- No support for non-ascii languages

So you decide C is the right choice

- “Unix is user friendly. Its just particular about who its friends are”
- It lacks support to prevent you from making silly and sometimes catastrophic errors.

Moog Bug

- Moog speaks through a network interface
- Expects packets in real time representing (x,y,z,p,q,r) state.
- Moves there as fast as possible from its current state.
- Wrote a library (in C) to provide support from another machine

Moog Bug

- Code had a timer loop, estimated desired (x,y,z,p,q,r) and output them.
- Tested in simulation - good
- Run - no motion at all.
- Killed program - moved at blinding speed through a complex motion.
- Cause identified -> buffering in network code caused data to be buffered at the Unix side.

Moog Bug

- Run again
  - Robot moved, but moved very very quickly from its starting state to some random state.
- Cause turned out to be a bigendian/littleendian number format difference between the Moog computer and the host.

C

- There exist many IDE's (OSX, Windows, etc.)
- Often one of the reasons you choose C as the implementation language was due to hardware requirements
  - Often implies poor support for IDE's
- The command line tools (almost) always work, (almost) always work in the same way.

So some basic tools

- gcc -Wall - your friend
- make - standard mechanism to automate the process of building images
- git - standard tool for version control, and with tools like GitHub also a standard mechanism for supporting off-site backup and storage.
- vi/emacs - programmers editors. (Almost) always available.
- adb, gdb, dbx - command line debugging tools
- prof, gprof - profiling tools
- ar - make static libraries (.a)
- gcc - make shared libraries

Typical gotcha’s

- Unix (Linux) assumes things go in specific places
  - /include /usr/include /lib /usr/lib others
- If you are writing code for non-unix machines this may not be so true.

But for many environments & tasks, C is the right tool for the job

Brian Kernighan

- Controlling complexity is the essence of computer programming.
  - *Software Tools* (1976), p. 319 (with P. J. Plauger).
- The most effective debugging tool is still careful thought, coupled with judiciously placed print statements.
  - "Unix for Beginners" (1979).
- Everyone knows that debugging is twice as hard as writing a program in the first place. So if you're as clever as you can be when you write it, how will you ever debug it?
  - "The Elements of Programming Style", 2nd edition, chapter 2.

Bash (or any shell)

- Typically built for programmers, for programmers.
- Designed to make your task easier (not harder)
-

What’s left

- Lab10 (next week), Last quiz (next week), Lab test (the week afterwards), final exam.
- Final exam is set by the registrar.
- Unofficial grades will appear on Moodle, likely before Christmas, but if not, before the new year.

Questions?