Linux 101 - Bash Scripting

Evan Widloski - 2014-09-30

The Bash Shell

- shell - interactive interface to system

csh, sh, zsh, ksh, dash, ash, busybox, etc.

- 'Bourne-again' Shell - 1989, FSF
- GNU replacement to Bourne shell - 1977, AT&T
- Default to nearly all GNU/Linux distributions

Simple Example

```
#!/bin/bash                 <-- Hashbang / Magic Number
                                Used to identify interpreter

#print text                 <-- Comment
echo 'Hello World!'

echo 'Hi';echo 'there'      <-- Separate commands with ;

/path/to/myscript           <-- Call other scripts

exit 0                      <-- Close with exit status 0
```

Variables

```
cookies=3
cookies="yummy"
```

<-- Assign number/string to variable

```
echo $cookies
yummy
```

<-- Print variable contents

```
declare -r readonly=25
readonly=50
bash: test: readonly variable
```

<-- Make a read-only variable

man bash --> SHELL BUILTIN COMMANDS

```
echo $SHELL
```

<-- Some variables are pre-set

$PWD,$USER,$TERM,$?

Strings and Variables

```
animal="dog"


echo "I love my $animal"      <-- Substitution
I love my dog


echo "I love my ${animal}s"   <-- Access variables 'safely'
I love my dogs

echo ${#animal}
3
```

Types of Quotes

```
echo "I love my \$animal"   <-- Escaped $ character
I love my $animal                others: $ ` \ * @ > < |

echo 'I love my $animal'    <-- Literal interpretation
I love my $animal
```

if Statements

```
if command                          <-- Check exit status of
then                                    command
  echo 'exit status was 0'
else
  echo 'status was not 0'
fi
```

```
if grep 'spaghetti' file            <-- Check for spaghetti in
then                                    file
  echo 'found spaghetti'
  more commands
else
  echo 'no spaghetti :('
fi
```

Tests

((*EXPRESSION*)) <-- Used for arithmetic

[[*EXPRESSION*]] <-- Used for files and strings

```
(( a = 3**2 + 4 ))          <-- Integer math

(( a > 10 ))

[[ $str1 == 'mystring' ]]   <-- String testing

[[ -a filename ]]           <-- File exists?
```

man bash --> CONDITIONAL EXPRESSIONS

[] vs [[]]

[] - old syntax for Bourne shells

[[]] - new implementation

New	Old
`[[ 3 > 1 ]]`	`[ 3 \> 1 ]`
`[[ 10 > 1 && 2 < 3 ]]`	`[ 10 \> 1 -a 2 \< 3 ]`
`[[ $test == 'space example' ]]`	`[ "$test" = 'space example' ]`

Logical Operators

```
&& <-- and
|| <-- or
```

```
if (( 10 > 5 )) && (( 2 > 1 ))
then
  echo 'True'
fi
```

```
if [[ $USER == 'evan' ]] || command
then
  echo 'Success'
fi
```

Short Circuits

```
command  && echo 'Success'          <-- Direct parallel to C
command  || echo 'Failure'              short circuits
```

&& <-- evaluate 2nd if 1st succeeds

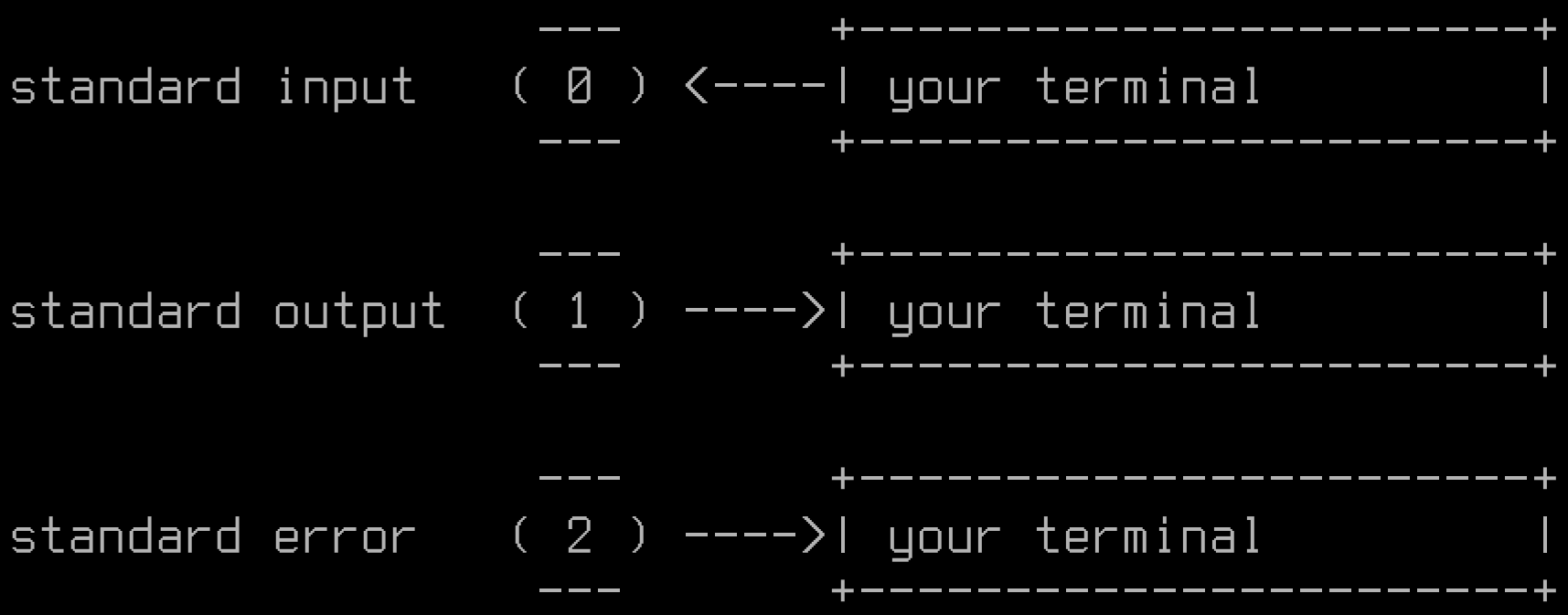
|| <-- evaluate 2nd if 1st fails

```
if [[ -a filename ]]                <-- long form
  then rm filename
fi
```

```
[[ -a filename ]] && rm filename    <-- Alternate form
```

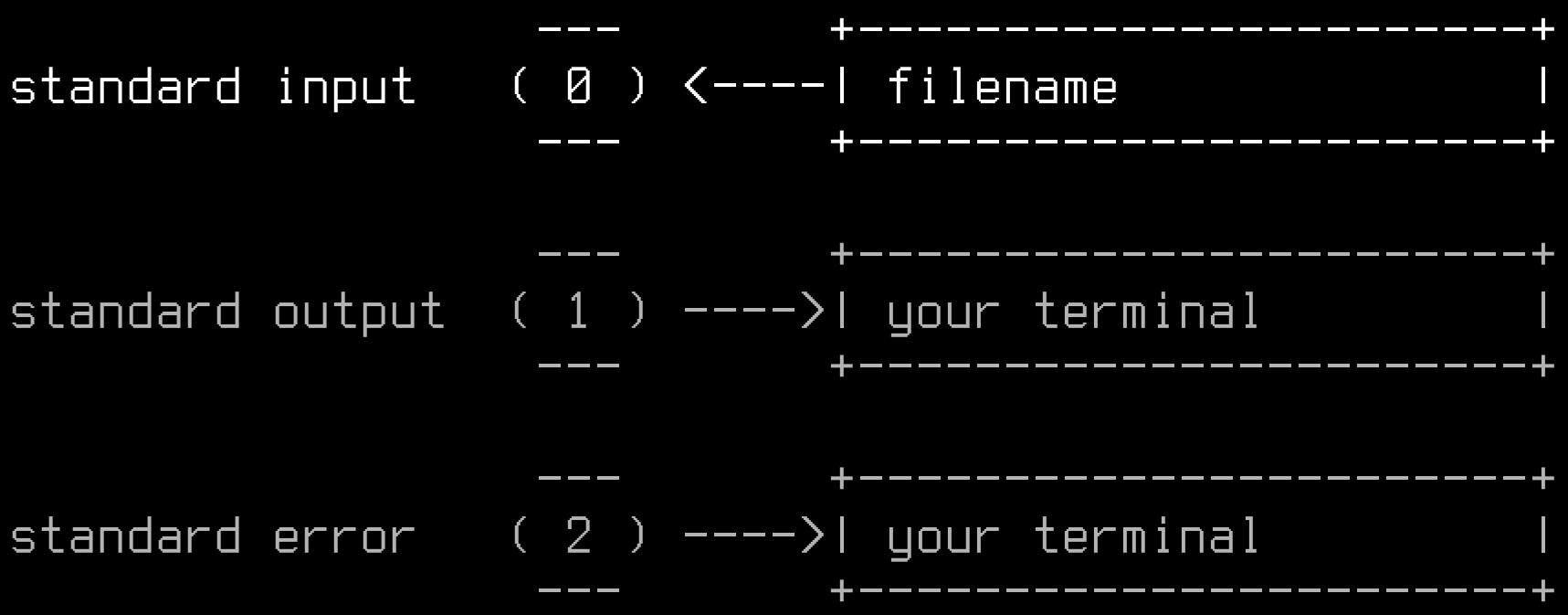
Redirection

3 input and output streams setup by bash



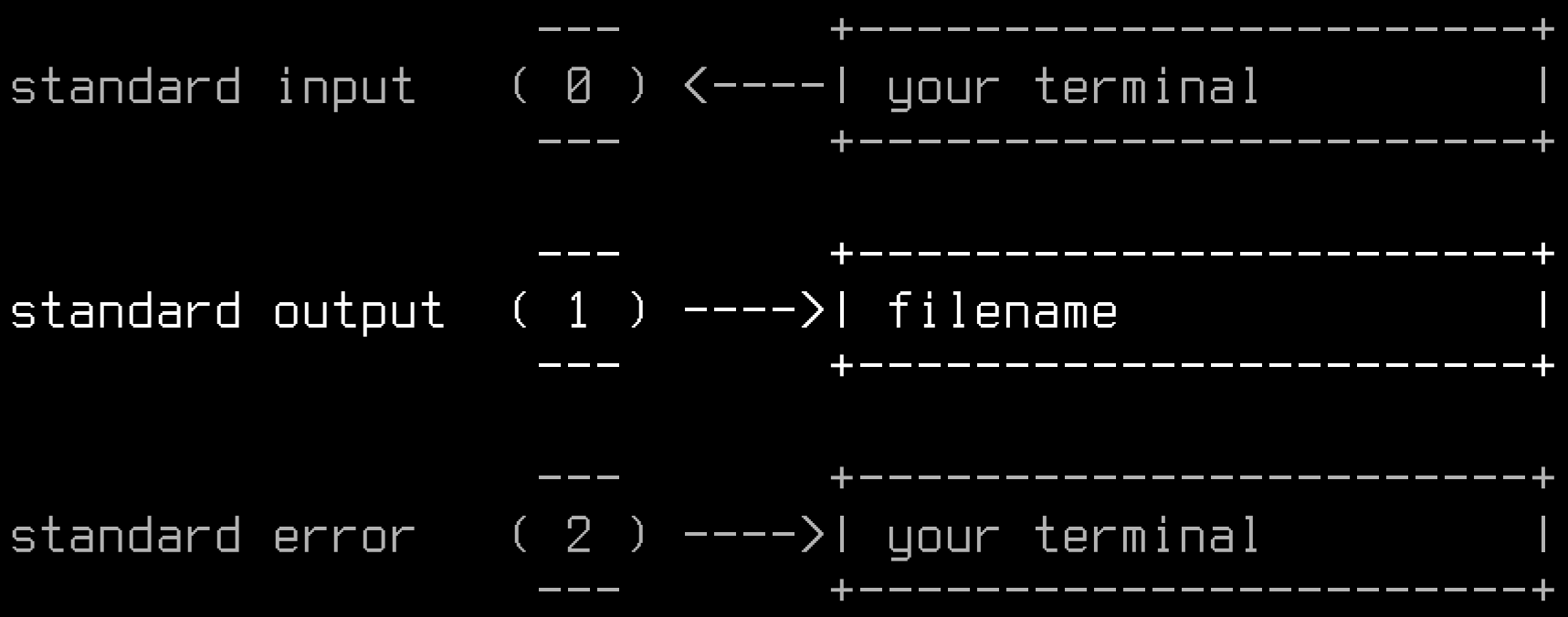
Redirection

command 0< filename



Redirection

```
echo 'Chili Dogs' 1> filename
```



Redirection

```
command 1> fileout        <-- these are the same
command > fileout

command 0< filein         <-- these are the same
command < filein


command | command2       <-- stdout to stdin

command >> filename      <-- append to file

command 2> errorlog      <-- redirect stderr to log
```

Globbing Files

`*` - match all, many times
`?` - match all, once

```
ls ?2.txt
12.txt a2.txt d5.txt 22.txt

ls hello.*
hello.jpg hello.o hello.c
```

Loops

```
for file in *.jpg
  if [[ $file =~ ^2014 ]]
    echo $file
  fi
done
```

<-- No " necessary

```
for file in *.jpg
  ls -l "$file"
done
```

<-- Prevent whitespace expansion

.bashrc

PATH variable tells bash where to look for commands

```
PATH=$PATH:~/bin:~/resources/bin
export PATH
```

alias creates a shortcut to a command

```
alias ls='ls --color=auto'
alias ll='ls -la'
alias ..='cd ..'
```

.bashrc

```
PS1="\[\e[00;32m\]\u\[\e[0m\]\[\e[00;34m\]\h\[\e[0m\]\
[\e[00;33m\]\@\[\e[0m\]\[\e[00;37m\]:\[\e[0m\]\
[\e[00;36m\]\w\[\e[0m\]\[\e[00;37m\]>\v\[\e[0m\]\
[\e[00;32m\]:\[\e[0m\]\[\e[00;31m\]\W\[\e[0m\]"
```

Change Prompt Statement (PS1)

evan@computer:07:23 AM:/usr/local/src>bash:src:

bashrcgenerator.com

Useful Resources: mywiki.wooledge.org

wiki.bash-hackers.org

freenode: #bash

Stay updated at purduelug.org/calendar

freenode:#purduelug

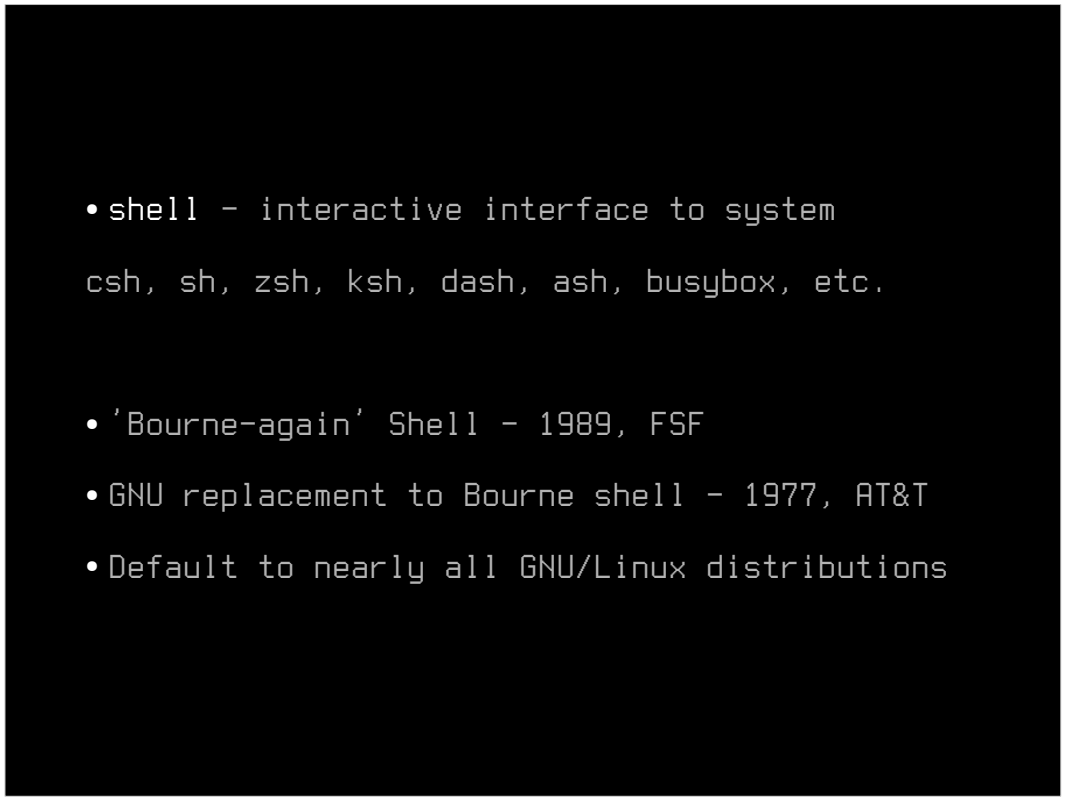
Please provide feedback: tinyurl.com/plug2014bash

Next event: Mini Techtalks

Saturday, Oct. 4, 11:30am

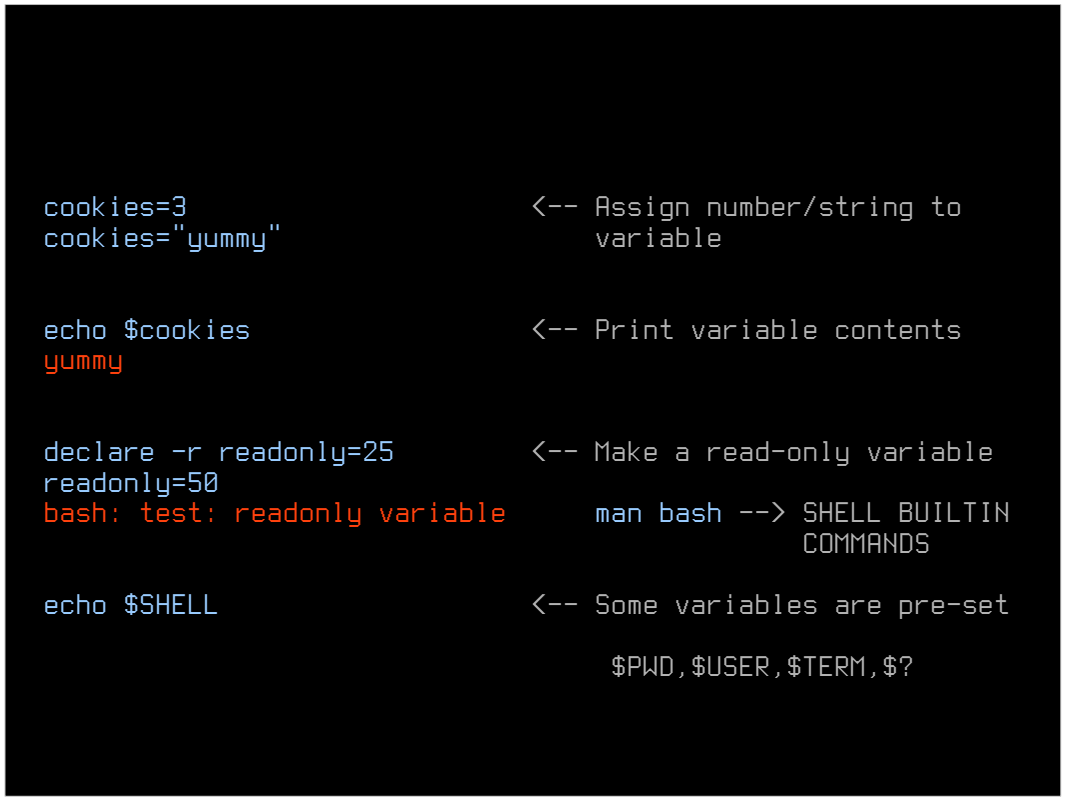
ARMS1109

things to add
- arrays
- backticks
- wget text vs del file example script



Bourne Shell - Stephen Bourne, Unix Version 7

```
#!/bin/bash             <-- Hashbang / Magic Number
                            Used to identify interpreter

#print text             <-- Comment
echo 'Hello World!'

echo 'Hi';echo 'there'  <-- Separate commands with ;

/path/to/myscript       <-- Call other scripts

exit 0                  <-- Close with exit status 0
```



other declare flags -i:integer -l:lowercase

make specific mention of $?, it will be used later

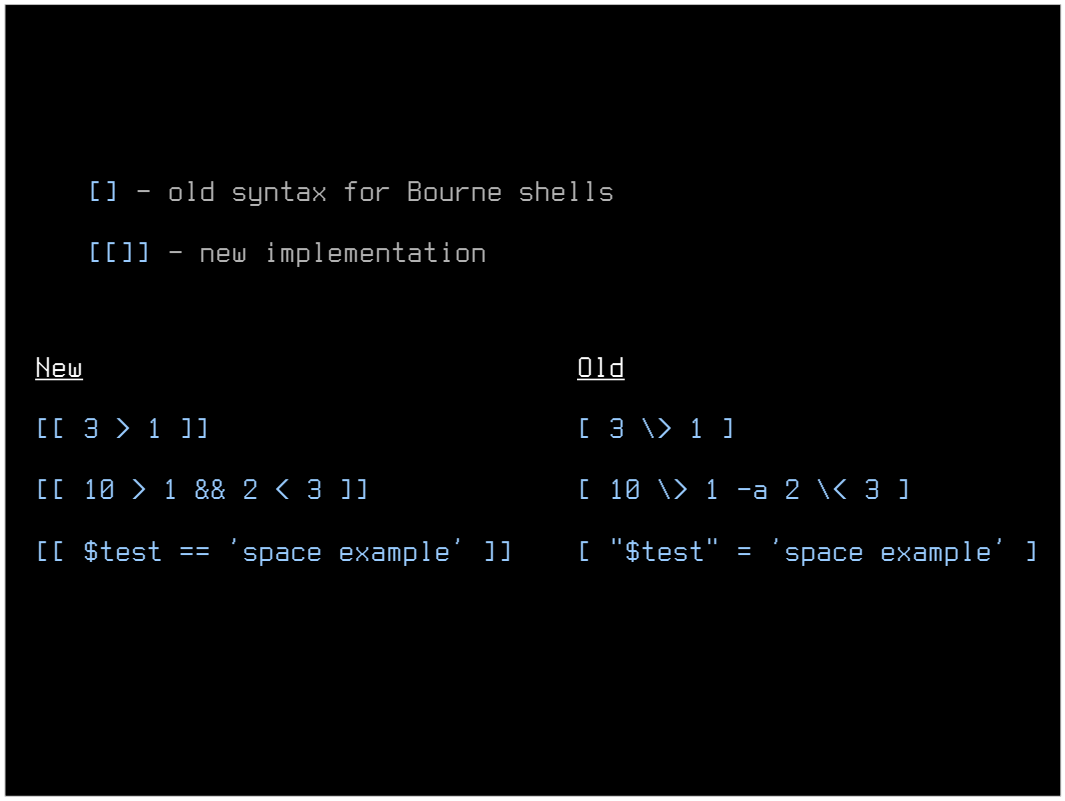
```
animal="dog"


echo "I love my $animal"      <-- Substitution
I love my dog


echo "I love my ${animal}s"   <-- Access variables 'safely'
I love my dogs

echo ${#animal}
3
```

double quotes- interprets special chars by default
 - need to get a literal character but keep else default, use backslash

single quote - don't interpret anything, literal string
  - use for fixed strings

```
if command                    <-- Check exit status of
then                              command
  echo 'exit status was 0'
else
  echo 'status was not 0'
fi


if grep 'spaghetti' file      <-- Check for spaghetti in
then                              file
  echo 'found spaghetti'
  more commands
else
  echo 'no spaghetti :('
fi
```

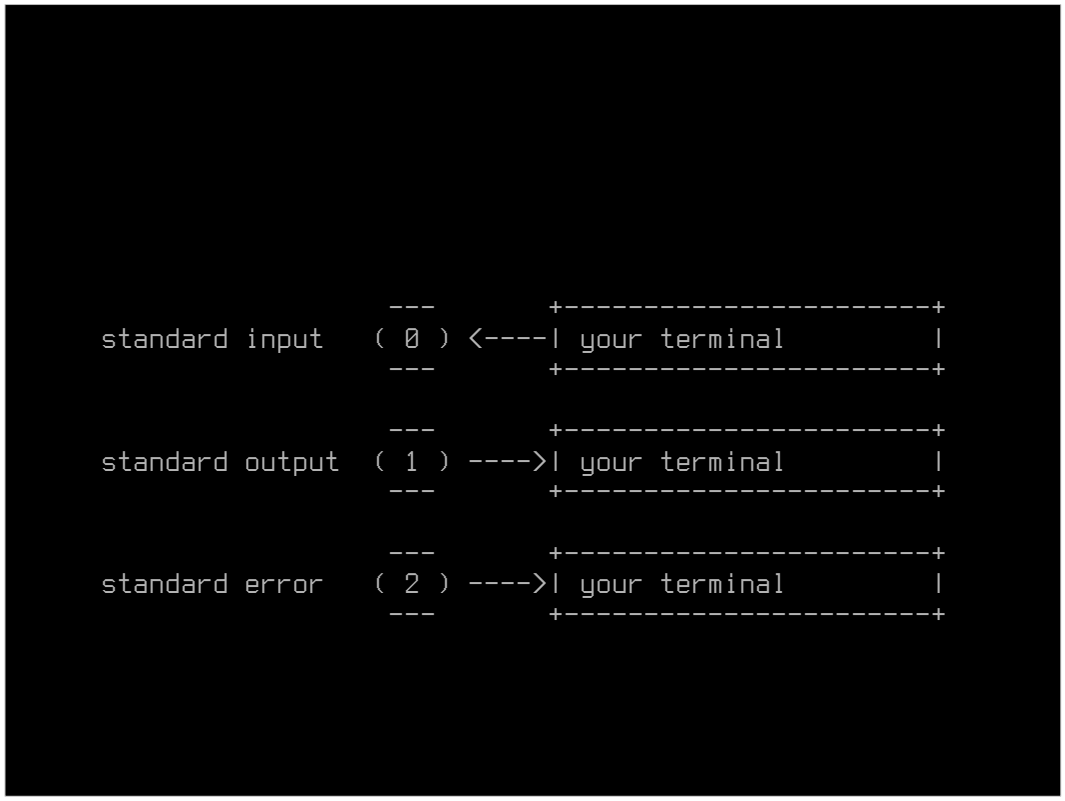
```
(( EXPRESSION ))              <-- Used for arithmetic

[[ EXPRESSION ]]              <-- Used for files and
                                  strings


(( a = 3**2 + 4 ))            <-- Integer math

(( a > 10 ))

[[ $str1 == 'mystring' ]]     <-- String testing

[[ -a filename ]]             <-- File exists?


                              man bash --> CONDITIONAL
                                           EXPRESSIONS
```



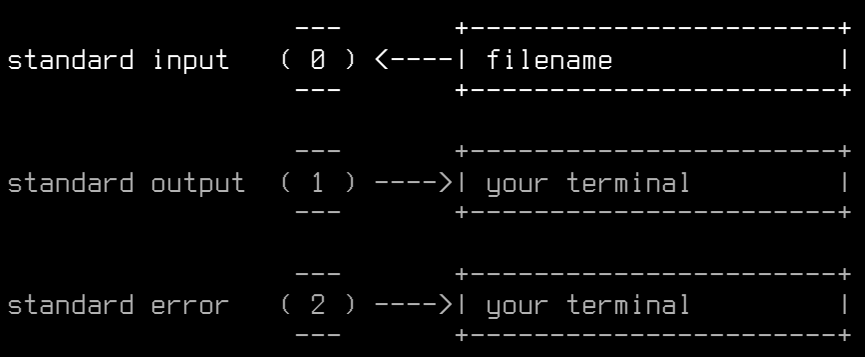
new implementation for Bash Korn Zsh

```
&& <-- and
|| <-- or


if (( 10 > 5 )) && (( 2 > 1 ))
then
  echo 'True'
fi


if [[ $USER == 'evan' ]] || command
then
  echo 'Success'
fi
```

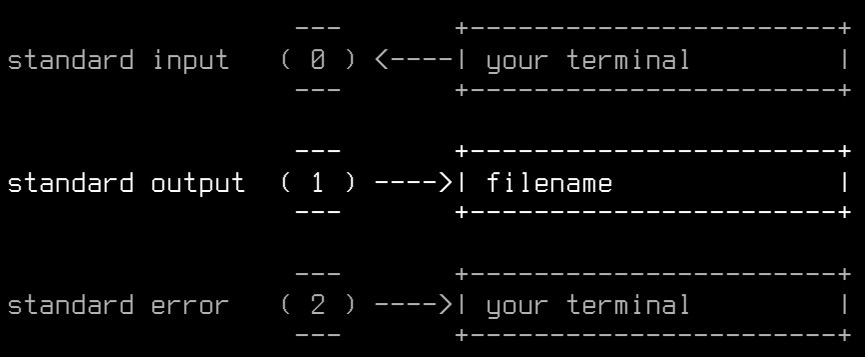
```
command  && echo 'Success'        <-- Direct parallel to C
command  || echo 'Failure'            short circuits

          &&

          ||

if [[ -a filename ]]              <-- long form
  then rm filename
fi


[[ -a filename ]] && rm filename  <-- Alternate form
```


standard input
(0)
your terminal
standard output
(1)
your terminal
standard error
(2)
your terminal

command 0< filename

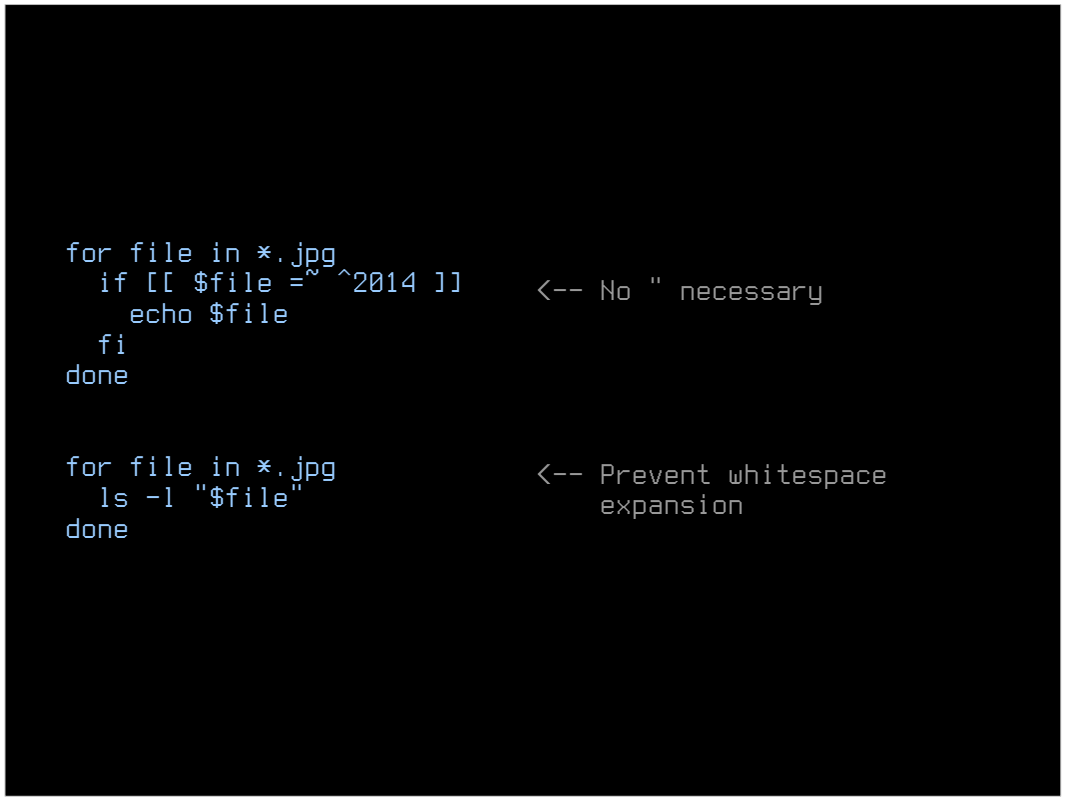


```
echo 'Chili Dogs' 1> filename
```



```
command 1> fileout      <-- these are the same
command > fileout

command 0< filein       <-- these are the same
command < filein


command | command2     <-- stdout to stdin

command >> filename    <-- append to file

command 2> errorlog    <-- redirect stderr to log
```

```
* - match all, many times
? - match all, once


ls ?2.txt
12.txt a2.txt d5.txt 22.txt

ls hello.*
hello.jpg hello.o hello.c
```



examples/loops

examples/loopspace

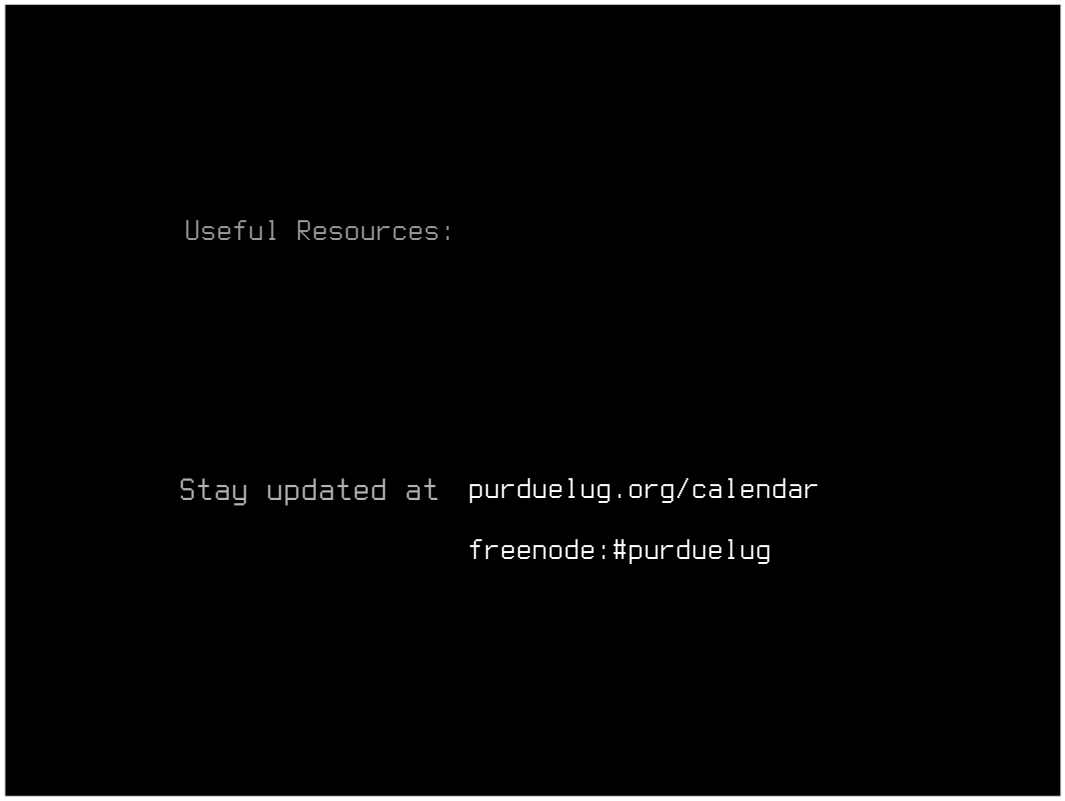
for file in *;do ls -l $file;done

where bash should look for commands
echo $PATH
export allows to be seen by others

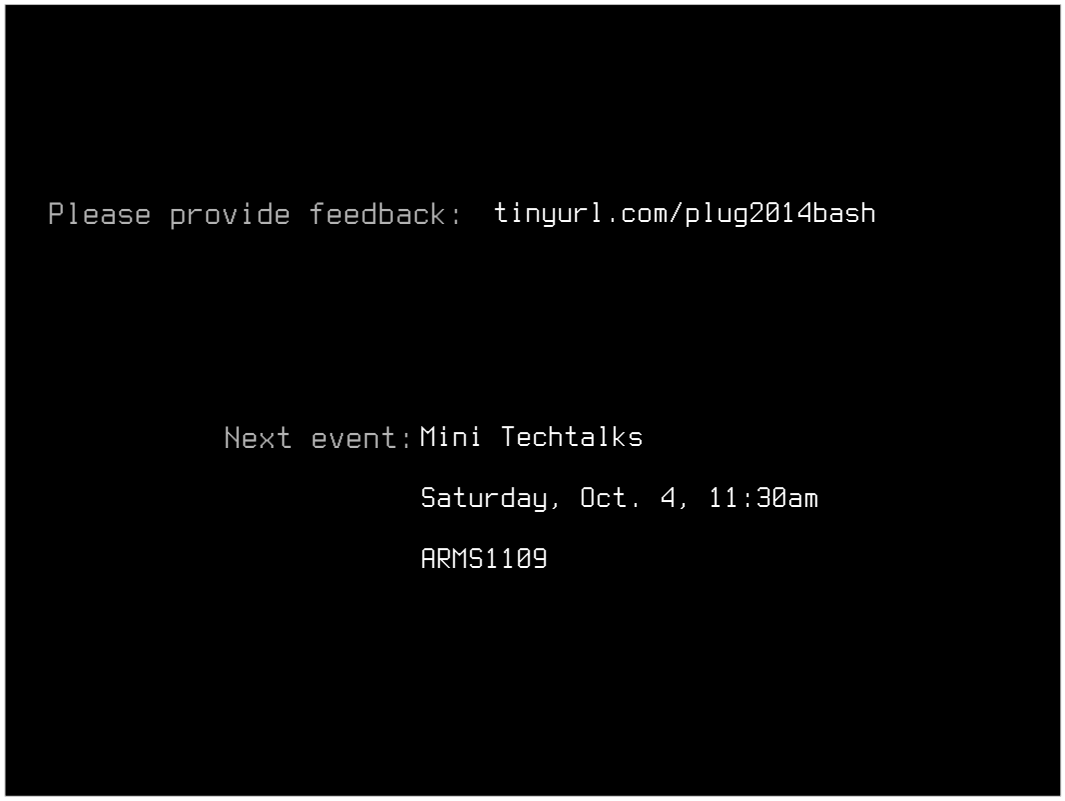
PS - prompt statement
PS1 - default
PS2 - >

where bash should look for commands
echo $PATH
export allows to be seen by others

PS - prompt statement
PS1 - default
PS2 - >



http://blog.openhatch.org/2013/teaching-open-source-at-purdue-university/



http://blog.openhatch.org/2013/teaching-open-source-at-purdue-university/