Shells:

* Bourne shell(bsh)
* C shell (csh)
* Korn shell (ksh)
* Bourne Again Shell (bash)
* Tcsh shell (tcsh)
* Almquist Shell (ash)
* Etc.

Shell Scripting

* You can store a sequence of commands in a regular text file, which you can then invoke as “program” (shell script)

Create shell script

* shell scripts stored in plain text file, typically with one command per line
* if you want more than one command per line, separate with semicolon
* by convention, use .sh

Comments

* comments: # I am a comment
* #! Is often called “shebang”

Variables

* variable\_name=variable
* no=10
* \*\* don’t use spaces, it may put in spaces in the name or value.
* Alphanumeric and underscores only
* Accessing variable values:
  + General: $variable\_name
  + Or : ${variable\_name}

Exit code

* 0 means passed, non-0 means failure
* in c, exit code is value returned from int main
* execute program then immediately check special variable $?
* Programname && echo “program failed to run successfully.”

Shell arithmetic

* Legacy way requires program execution:
  + expr op1 math-operator op2
* ex: var = ‘expr 1 + 3
* var = $(expr 2 – 1)
* modern BASH supports inline math:
* var=$((1+3))
* var=$((2 – 1))

Arguments

* arguments to the shell script are done with positional variables:
* $n
* where n is the number of the argument. Note that the arguments will be the same as argv in a c program- that is $0 should be the name of the program being executed.

Read:

* man read:
  + read a line from standard input
* read variable 1, variable2, … variable
* ex:
* echo “What is your name?”
* read tehname
* echo “your name is ${tehname}!”

Shorthand

* \*: all files
* Z\*: all files starting with Z
* \*.z : all files with extension .z
* Xy\*.a : all files starting with Xy and with extension .a
* Ab?: all files starting with Ab and with one more character thereafter
* {a..g} : all characters in the range a through g

Test Statement

* test condition
* exit code of test is 0 if condition is true (success), 1 if false (failure to be true)
* read value
* test 5 == $value || echo “not equal”
* read value
* [ 5 == $value ] || echo “not equal”

If statement

* if condition
* then
  + #if cond is true
  + #exit status 0
* else
  + #if cond is false
  + #(exit status non 0)
* fi

Inline logic

* read choice
* if [$choice > 0] ; then
  + echo “num is pos”
* else
  + echo “num is neg”
* fi

Loops

* variable used in loop condition must be initialized, then execution of loop begins
* a test condition is made at the beginning of each iteration
* the body of loop ends with a statement that modifies the value of test condition variable.