



Basic shell scripting

CS 2204

Class meeting 5



Shell script/program

- A series of shell commands placed in an ASCII text file
- Commands include
 - Anything you can type on the command line
 - Shell variables
 - Control statements (if, while, for)



Resources

- Review UIAN chapter 4
- Online
 - HTML notes with examples
 - Example scripts with comments



Script execution

- Provide script as an argument to the shell program (e.g. `ksh my_script`)
- Or specify which shell to use within the script
 - First line of script is `#!/bin/ksh`
 - Make the script executable using `chmod`
 - Run directly from the command line
- No compilation is necessary!



Simple example script

```
#!/bin/ksh
```

```
print "hi world"
```

```
cd ~
```

```
pwd
```

Output:

```
hi world
```

```
/home/faculty/bowman
```



Shell variables

- Numeric
 - Strings
 - Arrays
 - Command line arguments
-
- `var` refers to the name, `$var` to the value
 - Remove a variable with `unset var`
 - Names begin with alpha characters and include alpha, numeric, or underscore



Numeric variables

- Integer variables are the only pure numeric variables that can be used in ksh
- Declaration and setting value: `integer var=100`
- Expressions in the style of C:
 - `((expression))`
 - e.g. `((var+=1))`
- `+, -, *, /, %, &, |, ~, <, >, <=, >=, ==, !=, &&, ||`



String variables

- If you don't use the `integer` keyword when using a variable for the first time, it will be a string
- `var=100` makes `var` the string '100'.



Array variables

- Array is a list of values
- Reference a value by `${name [index]}`
 - `${a [3]}`
 - `$a` (same as `${a [0]}`)
- Use the `set -A` command to initialize an array
 - `set -A sports football basketball`
 - `set -A moresports ${sports[*]} tennis`



Command line arguments

- If arguments are passed to a script, they can be referenced by `$1`, `$2`, `$3`, ...
- `$0` refers to the name of the script
- `$@` - array filled with arguments
- `$#` - number of arguments



Exporting variables

- The `export` command, when used with a variable name, allows child processes of the shell to access the variable



Quoting

- If you want to use special characters in a variable's value or string, you must use quoting
- `"` - shell only interprets `$` and `\'`
- `'` - shell doesn't interpret any special characters
- `\char` - shell doesn't interpret char



Output

- We've already seen `echo`
- More common in shell scripts is `print`
- Both commands will print a newline character after the string unless you use the `-n` option



Return values

- Scripts can return an integer value
- Use `return N`
- The variable `$?` will contain the return value of the last command run



Conditions

- If using integers: `((condition))`
- If using strings: `[[condition]]`
- Examples:
 - `((a == 10))`
 - `((b >= 3))`
 - `[[$1 = -n]]`
 - `[[($v != fun) && ($v != games)]]`
- Special conditions for file existence, file permissions, ownership, file type, etc.



If statements

- Syntax:

```
if condition  
then
```

```
    statements
```

```
elif condition  
then
```

```
    statements
```

```
else
```

```
    statements
```

```
fi
```

} optional



While loops

- **Syntax:**

```
while condition
```

```
do
```

```
    statements
```

```
done
```

- **The keywords `break`, `continue`, and `return` have the same meaning as in C**



For loops

- **Syntax:**

```
for var [in list]
```

```
do
```

```
    statements
```

```
done
```

- **If *list* is omitted, $\$*$ is assumed**



Case statements

- Syntax:

```
case expression in
  pattern1)
  statements ;;
  pattern2)
  statements ;;
  ...
*)
  statements ;;
esac
```



Command substitution

- Use the output of a command in a variable, condition, etc. by:
 - ``command``
 - `$(command)`



Examples

- Arguments printed in a for loop
- Reformatting the wc command
- Performing a depth-first search