



Basic shell scripting

CS 2204

Class meeting 7

*Notes by Doug Bowman and other members of the
CS faculty at Virginia Tech. Copyright 2001-2003.



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
 - Advanced bash-scripting guide
<http://www.tldp.org/LDP/abs/html/index.html>
 - Bash Reference Manual
<http://www.gnu.org/manual/bash-2.05a/bashref.html>
 - ksh Reference Manual
<http://www.bolthole.com/solaris/ksh.html>



Script execution

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



Simple example script

```
#!/bin/bash
```

```
echo "Hello world!"
```

```
cd ~
```

```
pwd
```

Output:

```
Hello world!
```

```
/home/grads/sgifford
```



Quoting

- Quoting is necessary to use special characters in a variable's value or string
- `""` - shell only interprets `$` and ``` and `\`
 - `$` - variable substitution
 - ``` - Command substitution
 - `\"` - Literal double quote
- `\` is used to escape characters
 - `echo `date +%D`` will print: 10/06/03
- `'` - shell doesn't interpret special characters
 - `echo `date +%D`` will print: `date +%D`



Shell variables

- Numeric
- Strings
- Arrays
- Command line arguments
- Functions
- Read only
- `var` refers to the name, `$var` to the value
 - `t = 100` #Sets var `t` to value 100
 - `echo "\$t = $t"` #will print: `$t = 100`
- 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 bash
- Declaration and setting value:

```
declare -i var=100
```
- Expressions in the style of C:
 - `((expression))`
 - e.g. `((var+=1))`
- `+, -, *, /, %, &, |, ~, <, >, <=, >=, ==, !=, &&, ||`



String variables

- If you do not use the `declare` keyword with option `-i` when using a variable for the first time, it will be a string
- `var=100` makes `var` the string '100'.
 - However, `((var=100))` will treat `var` as an integer even though it is a string



Array variables

- Array is a list of values
 - Don't have to declare size
- Reference a value by `${name[index]}`
 - `${a[3]}`
 - `$a` (same as `${a[0]}`)
- Use the `declare -a` command to declare an array
 - `declare -a sports`
 - `sports=(ball frisbee puck)`
 - `sports[3]=bat`



Arrays

- **Array initialization**
 - `sports=(football basketball)`
 - `moresports=(${sports[*]} tennis)`
- `${array[@]}` **or** `${array[*]}`
refers to the entire array contents
- `echo ${moresports[*]}` **produces**
`football basketball 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 excluding `$0`
- `$#` - number of arguments



Exporting variables

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



Output

- We have already seen `echo`
- More common in other shells including `ksh` is `print` (does not exist in `bash`)
- `echo -n` does not print newline after output



Return values

- Scripts can return an integer value
- Use `return N`
- The variable `$?` will contain the return value of the last command run
- Can be used to test conditions



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.



Conditions (continued...)

- `[[-e $file]]` – File exists?
- `[[-f $file]]` – Regular file?
- `[[-d $file]]` – Directory?
- `[[-L $file]]` – Symbolic link?
- `[[-r $file]]` – File has read permission?
- `[[-w $file]]` – File has write permission?
- `[[-x $file]]` – File has execute perm?
- `[[-p $file]]` – File is a pipe?



If statements

- **Syntax:**

```
if condition
```

```
then
```

```
statements
```

```
elif condition
```

```
then
```

```
statements
```

```
else
```

```
statements
```

```
fi
```

} optional



If statement

- Example

```
if [[ -r $fname ]]
then
    echo '$fname is readable'
elif [[ -w $fname && -x $fname ]]
then
    echo '$fname is writeable and
    executable'
fi
```



For loops

- **Syntax:**

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

```
do
```

```
    statements
```

```
done
```

- **If *list* is omitted, `$@` is assumed**

- **Otherwise `${list[*]}`**

- where *list* is an array variable.



For loop example

```
for colors in Red Blue Green  
    Yellow Orange Black Gray White  
do  
    echo $colors  
done  
echo
```



While loops

- **Syntax:**

```
while condition
```

```
do
```

```
    statements
```

```
done
```

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



Case statements

- Syntax:

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



Case statement example

```
case $1 in
  -a)
    statements related to option a ;;
  -b)
    statements related to option b ;;
  *)
    all other options ;;
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