## Objectives

- Bash scripting

## Review

- What is a shell script?
  - What is an advantage of shell scripting?
- What is the format of a shell script?
- What can we do in a shell script?
- How do we create and use a variable?
- How do we use command-line arguments?

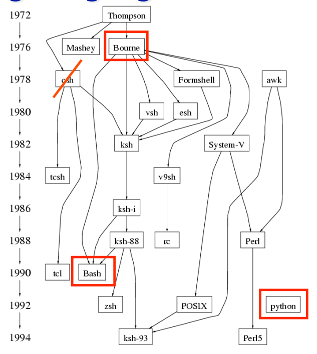
## Follow Up: `zsh`

- Extended Bourne shell
  - Improvements include some of the most useful features of bash, ksh, and tcsh
- 1st version written by Paul Falstad in 1990 when he was a student at Princeton
- Name derives from Yale professor Zhong Shao, then a teaching assistant at Princeton University
  - Paul Falstad thought that Shao's login name, "zsh", was a good name for a shell.

Source: `http://en.wikipedia.org/wiki/Zsh`

## UNIX Scripting Languages

- There are many choices for shells
- Shell features evolved as UNIX grew



## For Review

- Using special parameters $@ and “$@”

for_params.sh

## Case statement

- Like a C/Java *switch* statement for strings:

```
case $var in
        opt1)       command1
               command2
               ;;
        opt2)       command
               ;;
        *)     command
               ;;
esac
```

- ***** is a catch all condition (default)

## Case Example

```
#!/bin/bash

for INPUT in "$@"
do
    case $INPUT in
        hello)
            echo "Hello there."
            ;;
        bye)
            echo "See ya later."
            ;;
        *)
            echo "I'm sorry?"
            ;;
    esac
done
echo "Take care."
```

What does this script do?

How can I exercise all cases, output possibilities?

case.sh

## Case Options

- opt can be a shell pattern or a list of shell patterns delimited by |
- Example:

```
case $name in
    *[0-9]*)
        echo "That doesn't seem like a name."
        ;;
    S*|T*)
        echo "Your name starts with S or T, cool."
        ;;
    *)
        echo "You're not special."
        ;;
esac
```

case2.sh

## Functions

- Functions are similar to scripts and other commands except:
  - They can produce side effects in the callers script.
  - Variables are shared between caller and callee
    - Everything is global
  - The positional parameters are saved and restored when invoking a function.

## Function Syntax

```
function name {
    commands
}
```

or

```
name () {
        commands
}
```

- Local variables: positional parameters
  - $0 is the function's name

## Function Example

- What is the expected output?

```
function function_B {
    echo Function B.
}

function function_A {
    echo $0: $1
    function_C "$1"
}

function function_D {
    echo Function D.
}
```

```
function function_C () {
    echo "---------------"
    echo Function C: $1
    echo GLOBAL = $GLOBAL
    let GLOBAL=$GLOBAL+1
    echo "---------------"
}

GLOBAL=1

# FUNCTION CALLS
# Pass parameter to function A
function_A "Function A."
function_B
function_C "Function C."
function_D
```

functions.sh
functions2.sh

## Script Example

- Emit HTML for directory contents

```
$ bash dir2html.sh day4 > dir.html
```

## Command Search Rules

- When bash encounters some command (without slashes), it needs to figure out what to execute
- In order, bash looks for
  - Functions
  - Built-ins
  - PATH search

## Getting Input: `read`

- Example: getting user input

```
echo -n "Enter a value: "
read var
echo "\"var\" = $var"
```

read.sh

- Reading from a file
  - `bash readFile.sh < filename`

```
while read line
do
  echo "\"line\" = $line"
done
```

readFile.sh

## Command Substitution

- Better syntax with $(*command*)
  - Allows nesting
  - `x=$(cat $(generate_file_list))`
- Backward compatible with ` … ` notation

## Array Variables

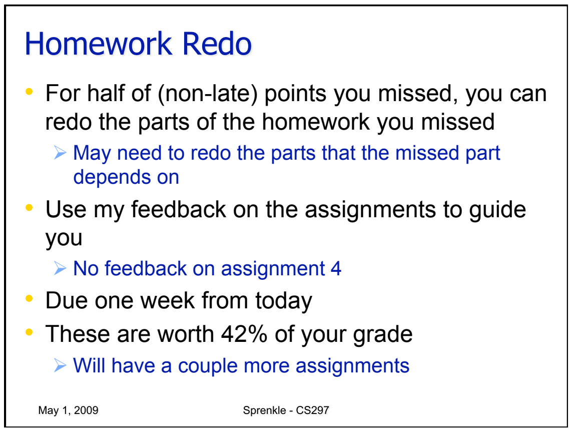
- Variables can be arrays
  - Indexed by number
  - Examples:
    - `foo[3]=test`
    - `echo ${foo[3]}`
- `${#arr}` is length of the array
- I found some information about Bash arrays which seems to be part of a newer version of Bash than we have

arrays.sh

## Some of My Scripts

## Common Homework Issues

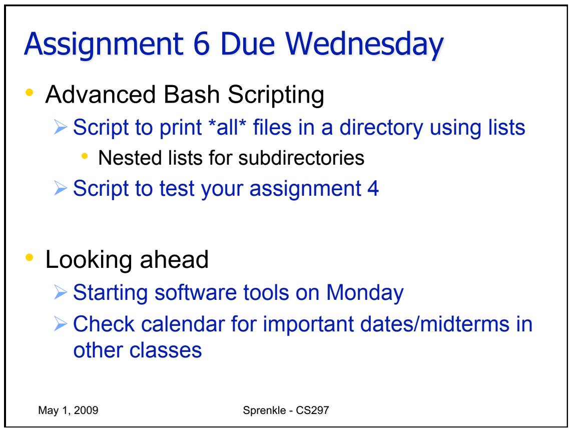
- Not looking at files you're working with
- Not looking at the output at intermediate steps
  - Doing unnecessary commands
- Not using the most appropriate command
- Not reducing output enough
  - Use appropriate options



## Homework Redo

- For half of (non-late) points you missed, you can redo the parts of the homework you missed
  - May need to redo the parts that the missed part depends on
- Use my feedback on the assignments to guide you
  - No feedback on assignment 4
- Due one week from today
- These are worth 42% of your grade
  - Will have a couple more assignments

May 1, 2009 Sprenkle - CS297



## Assignment 6 Due Wednesday

- Advanced Bash Scripting
  - Script to print *all* files in a directory using lists
    - Nested lists for subdirectories
  - Script to test your assignment 4
- Looking ahead
  - Starting software tools on Monday
  - Check calendar for important dates/midterms in other classes

May 1, 2009 Sprenkle - CS297