# Lecture 8:
# Bourne shell scripting (I)

Have you started HW3 yet?

# Exit status

- Every command run returns an *exit status*
  - 0 = success
  - Anything else = failed, somehow
  - `$?` = exit status of last command
- `grep -q` – doesn't print anything
  - Only useful for exit status
  - What if `grep` didn't have `-q`?
- Argument to
  - Java's `System.exit`
  - C's `return` from `main()`

# Signals

- What `kill` really does:
  - Send a “signal” to a process or job
  - Default = SIGTERM (TERMinate; please quit)
  - `9` = SIGKILL (KILL; extreme prejudice)
  - CTRL-C = SIGINT (INTerrupt)
- `yes > /dev/null` (CTRL-C)
  - $? = 130 = 128 + 2; 2 = SIGINT
- `yes > /dev/null; kill -9`
  - $? = 137 = 128 + 9; 9 = SIGKILL

## Using exit status; if/else

- **`if`** `grep -q purple colors`
- **`then`** `echo found purple`
- **`else`** `echo did not find it`
- **`fi`**

- Newlines are important!

## If/else in general

- **`if`** *`command1`*
- **`then`** *`command2`*
- **`elif`** *`command3`*
- **`then`** *`command4`*
- `...`
- **`else`** *`command5`*
- **`fi`**

# Semicolons

- Multiple commands on the same line – separate with semicolon
- Semicolon can substitute for a newline (but only for Bourne shell)

-
  ```
  if grep -q purple colors; then echo Yes; else echo No; fi
  ```

# Other conditions: test

- `test -f /etc/password`
  - true if /etc/password exists and is a normal file – so true
- `test 25 -gr 7`
  - True if 25 > 7 – so true
- `test Hello = World`
  - True if Hello = World – so false
- Many other conditions

- Can be called `[` instead of `test` (need `]`)

# Arguments to shell scripts

- `./myscript.sh 25 “Hello, World”`
- $0 = name of the shell script
  - $0 = ./myscript.sh
- $1 = first argument, $2 = second, etc.
  - $1 = 25
  - $2 = “Hello, World”
- “$*” = “25 Hello, World”
- “$@” = “25” “Hello, World”

## equal.sh

- `#!/bin/sh`
- `**if** [ $1 = $2 ]; **then** echo Equal; **else** echo Nope; **fi**`

- `./equal.sh Red Red`
- `./equal.sh Red Blue`

## For loops

```
for ii in 1 2
do
  echo $ii
done
```

- Prints
  - 1
  - 2
  - 3
- [illegible]

## A script with for

```
#!/bin/sh
for ii in “$@”; do
  echo $ii
done
```

## Using for on the command line (sh/bash/ksh)

- `for ff in *.doc; do cp $ff $ff.bak; done`

- `for ff in *.jpg; do mv $ff `echo $ff | sed -E 's/([0-9]+)-([0-9]+)-([0-9]+)/\3-\2-\1/'`; done`