COMP2101
Summer 2022

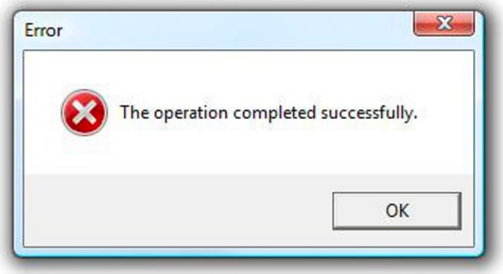
Miscellaneous Topics

https://assets.catawiki.nl/assets/2016/10/12/0/3/0/030dfe3a-90a7-11e6-8a7f-68d7b00d2775.jpg

Miscellaneous Topics

- error output
- functions
- signals and traps
- command line arguments
- data typing, sort of

Logging Error Output



- Commands may generate unwanted or irrelevant error messages
- That output can be saved as a log, sent to whoever should see it, or discarded
- Logs are usually kept in /var/log for programs we care about managing
- Redirecting error output with 2> or 2>> allows you to capture that output
- Consider using the logger command to send messages to the system logging daemon - example uses process substitution to send output to a command in a sub-shell

```
commandthatmakeserrorswedontcareabout 2> /dev/null                # throws away errors
command 2>>/var/log/myerrors.log                                   # adds errors to the end of a logfile
somecommand 2> >(logger -t $(basename "$0") -i -p user.warning) # sends errors to a process like piping but for stderr
```

Functions

- A function is a named script block, it creates a command you can use elsewhere in your script and must be defined before it can be used
- Inside a function, the arguments variables ($1, $2, $3, etc.) contain whatever was on the command line that ran the function
- Functions end with the status code of the last command to run in the function, data results can be passed back on stdout, or using an intermediary means such as storing data in a file or variable
- Variables are by default shared between functions and the rest of the script unless they are declared as local inside the function script block
- A function may be ended immediately with the return command
- Function definitions can be viewed with the type command, and deleted with the unset command

```
function myfunctionname {
   listofcommands
}
# can now use myfunctionname as a command later in the bash process
```

Signals

- Signal are a way of notifying a process you want it to do something, usually terminate
- Signals can be sent using the kill command TERM is default, INT(^C), QUIT(^\), HUP are also common ways to request a process to exit
- Processes can catch and process or ignore most signals
- KILL, STOP(^Z), CONT cannot be caught or ignored, processes do not know these happen
  - Signals STOP and CONT are used to pause/resume processes(jobs)
  - Signal KILL is used to forcibly immediately terminate a process

```
$ kill -SIGNAL pid
$ pkill -SIGNAL processname
```

Bash Jobs

- When bash executes a command line, it is said to be running a job
- Jobs usually run in the foreground until completion and then bash displays a new prompt to let the user know it is ready for the next job
- A job can be run in the background by appending & to the command line
- The current background job list can be viewed with the jobs command
- Background jobs can be brought to the foreground using fg %jobnumber
- ^Z tells bash to pause a foreground job (called stopping a job) by sending it the STOP signal
- bg %jobnumber can be used to continue running a stopped job in the background
- ^C tells bash to ask the foreground job to terminate itself, by sending it the INT signal
- Exiting bash will cause it to send a hangup signal to any background jobs that shell still has running, which may cause them to exit

Trap

- In a shell script, catching signals is done with the trap command
- trap can run a command when a signal is caught, functions are often useful for this

```
function cleanup {
    rm /tmp/mytemporaryfiles
    logger -t `basename "$0"` -i -p user.info -s “Cleaning up and aborting”
    exit 1
}

trap cleanup SIGHUP
trap cleanup SIGTERM
trap cleanup SIGINT
```

Miscellaneous Topics

- Dialog UI
- Data type declarations

https://optimisingmylife645241833.files.wordpress.com/2021/02/rtfm_750.jpg

Dialog boxes

- For more complex user interactions such as choosing files, selecting items from a list, or presenting graphics on text-only terminals, there is the dialog command
- dialog can ask for input/decisions or display information
- dialog is useful when you are working on a terminal and want to present interactions in a more user-friendly way than just displaying text
- e.g.

```
userpicked=$(dialog --menu "choose one" 0 0 0 a 1 b 2 c 3 d 4 e 5 --output-fd 1)
(for ((i=0;i<=100;i+=10)) do echo $i;sleep 1;done)|dialog --gauge "progress" 7 60;clear
foo=$(dialog --rangebox "Pick a value" 8 80 1 9 5 --output-fd 1);clear;echo "You chose $foo"
```

- dialog's command line can be inscrutable

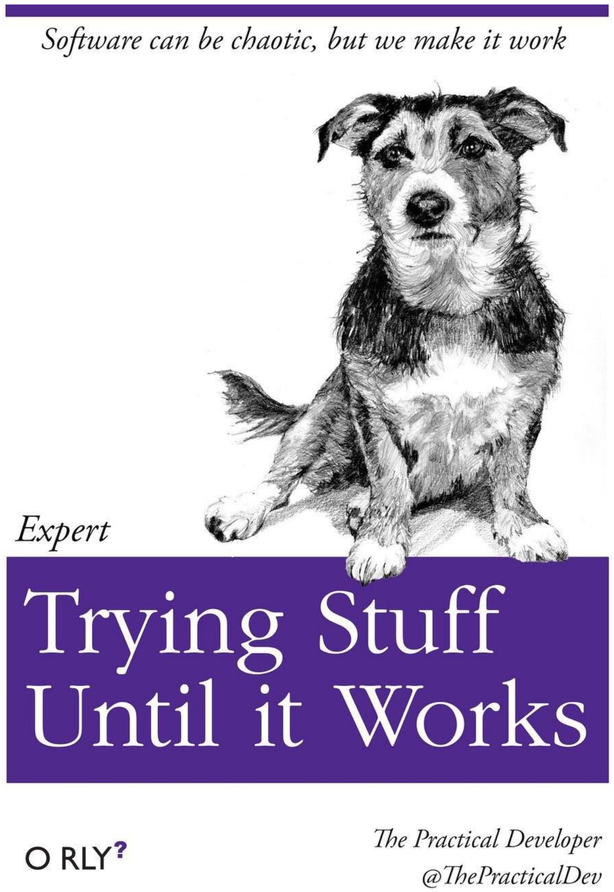
Declare

- The declare command can be used to display things stored in process memory that we can use or to give bash rules for a variable
  - declare -a varname will cause bash to only store arrays in varname
    declare -A varname will cause bash to only store associative arrays in varname
    declare -i varname will cause bash to only store integers in varname
  - declare -x varname will cause bash to put varname in the environment
    declare +x varname will cause bash to take varname out of the environment

```
declare -i myvar
myvar=$((16 * 32))
myvar="red"
```

```
declare -x VARNAME
VARNAME=“Data
declare +x VARNAME
```

Beyond the basics



- error output
- functions
- signals and traps
- command line arguments
- data typing, sort of
- fifth challenge script