Little Languages

Lecture 26:

Exit Statuses

VM day today!

Pull 590-material from upstream.

What happens when a process ends?

- Generally, the operating system reclaims the process' resources
  - Allocated memory
  - Open file handles
  - Open network sockets
  - Any other resource mediated by the OS

- The dying process *also* gives an **Exit Status** back to the operating system
  - Also commonly called an **Exit Code** or **Result Code**
  - The operating system signals this exit code back to its dying process' parent

- The exit status of the last process run in a shell is stored in the variable **$?**
  - When you run a process in the Bash shell, the shell is its parent
  - When the process ends, the shell is notified of its exit status and stores it in $?

Hands-on: Finding Exit Statuses

- Navigate to lecture/26-exit-status/exit_statuses
- This is a simple Rust project. Check its main.rs file and run it.
- Investigate its *exit status* after running the program by echoing the special variable: **`echo $?`**
- Respond on PollEv.com/compunc with the exit status, then:

1. Change your program to **panic!** and check the exit status
2. Change your program to exit with **std::process::exit(590);**

- Respond on PollEv with those exit statuses, as well.

An Exit Status lets the parent process know if its child successfully completed its task or not

- Processes normally exit with a status of 0
- When a process exits abnormally, a non-zero status should be used

- Why does 0 mean "OK"?
  - Programs can typically fail in lots of ways so other status codes can be used by the programs themselves to convey *why* they may have failed to a parent

- What do codes > 0 mean
  - Exit statuses are program specific – check their manual page
  - 1 is commonly used for general errors and sometimes expected errors
    - For example, grep's exit status is 1 if no matches were found
  - >=126 should be considered reserved by the system
    - For example, exiting a process with Ctrl+C exits with status 130

Aside: **true** and **false** commands

- In most *nix systems, there are two command-line applications for playing with exit status handling:

- `true` – exits with status `0`

- `false` – exits with status `1`

Aside: Conditional Command Sequencing

- In Bash, if you separate commands with:

  ; - all of the commands run sequentially

  && - the right-hand side command only runs if left-hand side exit status is 0

  || - the right-hand side command only runs if left-hand side exit status is != 0

A common idiom for simple "if success this, else that" is to combine && and ||:

```
command && echo "success case" || echo "failure case"
```

When do exit statuses matter?

- When you write a program or script that depends on *other* processes or scripts completing successfully

- Just like *composing* pipelines of processes is powerful, so too is building systems of coordinated processes to automate bigger jobs

- Scenarios where you want to understand exit statuses:
  - Managing child processes in general purposed languages - "Shelling out"
  - Shell scripting – "duct-taping" a number of CLI applications together

Shelling out in Rust

- Rust's `std::process` package is for working with child processes

- The `Command` struct enables you to build a child process by:
  - Specifying the programs name (to be found in $PATH)
  - Adding command-line arguments to program
  - Optionally controlling stdin/stdout/stderr
  - Spawning, waiting, and capturing Exit Status and (optionally) stdout/stderrerr

- For complete documentation see:
  - https://doc.rust-lang.org/std/process/index.html

```rust
use std::str;
use std::process::{Command, Stdio};

fn main() {

    let child = Command::new("ls")
        .arg("-l")
        .arg("-h")
        .stdout(Stdio::piped())  // Capture stdout rather than print to terminal
        .spawn() // Begin the process
        .expect("Failed to start process"); // If failed, panic!

    if let Ok(output) = child.wait_with_output() { // Wait for proc to complete
        println!("===");
        println!("Status: {}", output.status);
        println!("===");
        println!("Stdout: {}", str::from_utf8(&output.stdout).unwrap());
    } else {
        eprintln!("Failed to wait on child.");
    }
}
```

Developer Operations - DevOps

- It used to be software engineering and systems operations were separate and distinct roles

- Modern service-oriented architectures depend heavily on many systems
  - Web servers and databases are CLI-operated processes that run in the background
  - Build tools (like cargo and npm) are CLI-driven tools that coordinate other CLI tools
  - Continuous integration systems orchestrate sequences of tasks to test, build, and package your project per commit
  - Deployment systems transfer your builds to production servers and initiate sequences of steps to take the old version out of commission and roll-over to the new version

- Most modern software engineering projects heavily rely upon many CLI tools

- In the last 10 years a new type of role emerged. **Developer Operations** combines:
  1. The know-how of a computer scientist and software engineer
  2. The know-how of a systems architect and implementor

Bash's `if-then-else` grammar

```
if-statement   ::= "if" "[" expr "]" "then" statements* else-statement? "fi"
else-statement ::= "else" statements*
```

- Example:

```
if [ $SOME_NUM_VAR -eq 0 ]
then
  echo "Equals 0"
else
  echo "Not equal to 0
fi
```

There's Sun on the Horizon

- Weather.gov provides a textual weather forecast online:
  - https://forecast.weather.gov/MapClick.php?lat=35.9082&lon=-79.0459&unit=0&lg=english&FcstType=text&TextType=1

- How about we script the following:
  1. Grab its contents
  2. Convert it to markdown to simplify textual analysis
  3. Use egrep to search for "sun"
  4. If exit code status is:
     - 0 - match found! Echo a special message about happy days and print matches.
     - 1 - no matches found. Echo an uplifting message.

```
#!/bin/bash

URL="https://forecast.weather.gov/MapClick.php?lat=35.9082&lon=-
79.0459&unit=0&lg=english&FcstType=text&TextType=1"

FORECAST=$(curl --silent $URL | pandoc --wrap=none -f html -t markdown | egrep --ignore-case 'sun[^d]')
SUN_STATUS=$?

echo "======="
if [ $SUN_STATUS -eq 0 ]
then
    echo "Sunshine is on the horizon!"
    echo "======="
    echo "$FORECAST"
else
    echo "There's no sun this week but you're radiant so ¯\_(ツ)_/¯"
fi
echo "======="
```