UNIX 201

Ryan DeRue, Senior Computational Scientist

PURDUE UNIVERSITY
Rosen Center for Advanced Computing

Unix 201

Outline

What to expect from Unix 201

Objectives

- Develop an intuition for Unix processes and the data structures related to them
- Discuss the concepts of subprocesses and subshells and the implications their designs bring to everyday use
- Become comfortable with the concept of Bash variables and the nuances between shell variables and environment variables

Unix 201

Unix Processes

Unix Processes

What is a Unix process?

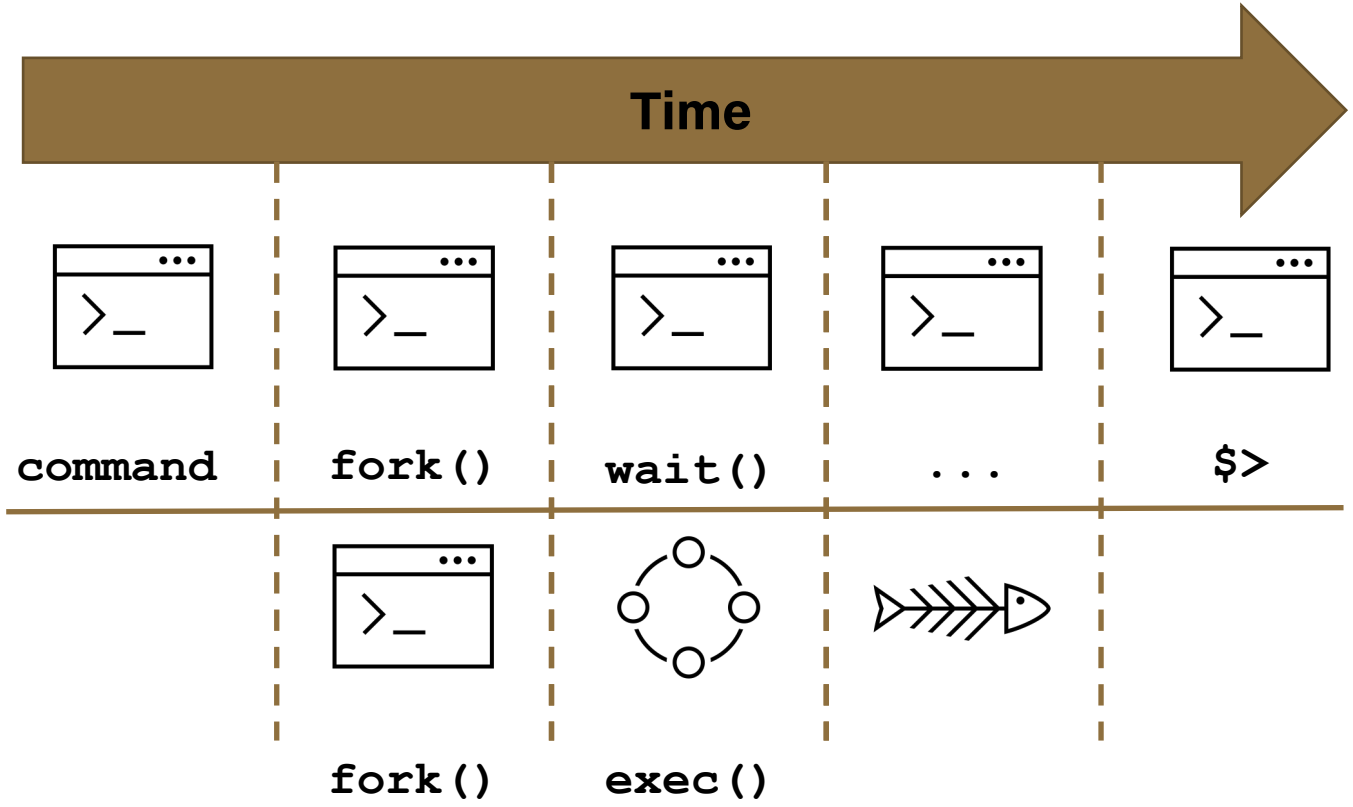
- A single instance of a running program for a given user
  - Program could be a command, a shell script, or an executable
  - Built-in commands do not create a process because they are part of the shell!

- Properties of a Process
  - PID: Process ID
  - PPID: Parent Process ID
  - UID: User's ID
  - TTY: Teletype Writer
  - File Descriptor Table

Unix Processes

How is a process created?

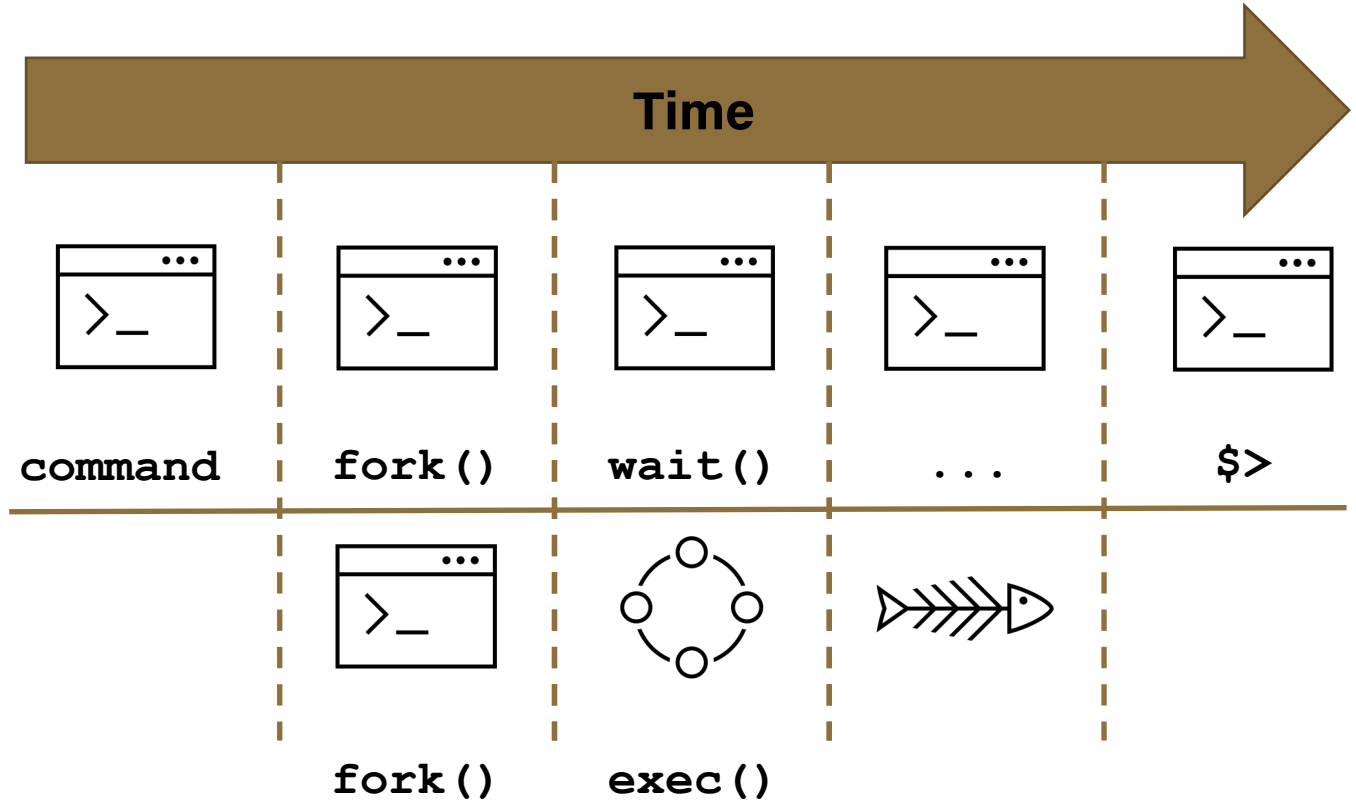
- All commands start out as a shell
- The `fork` and `exec` paradigm
  - `fork()`: Create a copy of me
  - `exec()`: Become something else
- What spawned the shell?
  - The *init* process
    - A special process created upon operating system start-up to spawn the first shell
- Can I spawn a shell from another shell?
  - Subshells



Unix Processes

What information can we find about processes?

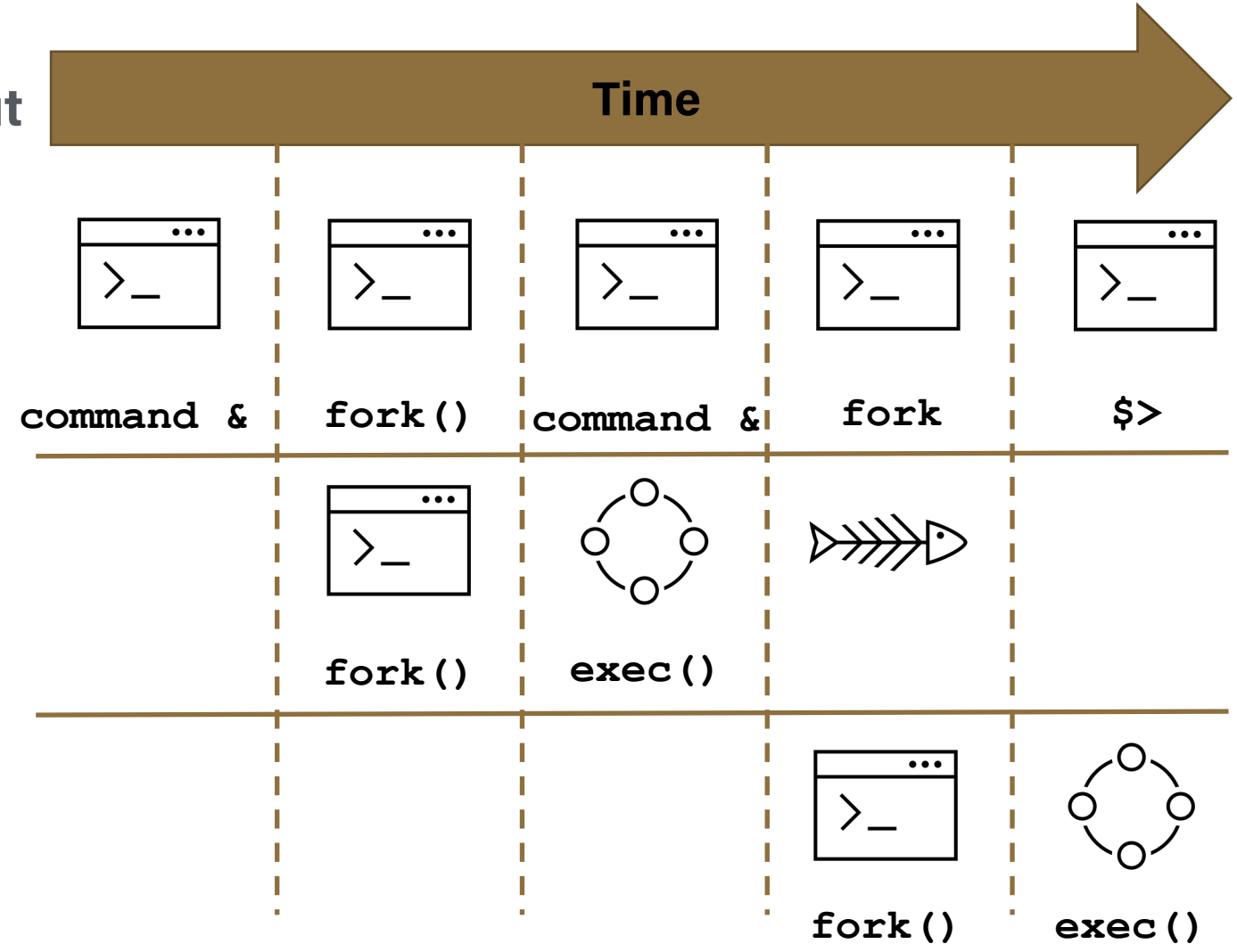
- A command for viewing processes: `ps`
  - Usage: `ps [-options]`
  - Lists all the requested information about running processes
- A command for analyzing processes: `top`
  - Usage: `top [-options]`
  - Lists resource usage of running processes
- Exit codes of processes
  - Zero exit codes report successful execution
  - Non-zero exit codes report non-standard behavior



Unix Processes

Can we execute commands without waiting for them to complete?

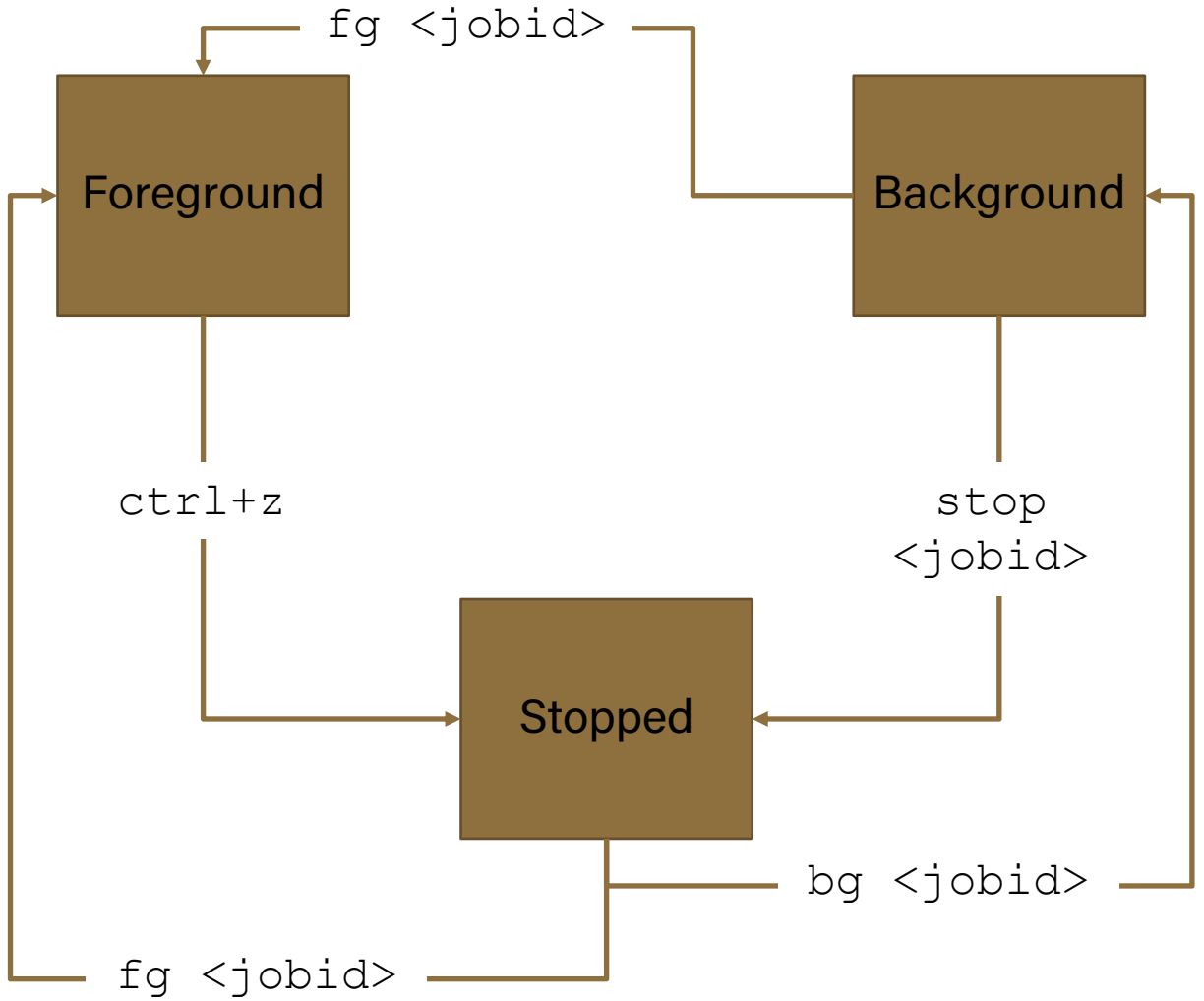
- Foreground processes
  - Interactive commands connected to your keyboard for input
  - Must wait for them to finish
- Background processes
  - Background processes, also called jobs, run while you continue to work
  - Run a command as a background process by appending an ampersand to the end of the line (`&`)
  - Returns a job ID and a process ID for the background job
  - New Built-in: `jobs`
    - list all current jobs



Unix Processes

Can we send a foreground process to the background?

- Sending a process to the background
  - Execution can be paused using ctrl+z
  - Pausing a process gives it a job ID
- Built-ins for interacting with jobs
  - `bg <jobid>`: Resume the job with the specified jobID in the background
  - `fg <jobid>`: Resume the job with the specified jobID as a foreground process
  - `stop <jobid>`: Put a backgrounded job into the stopped state



Why did Ctrl-Z pause the process?

- Signals are Unix defined interrupts which indicate a specific event has occurred
  - Signals can be raised for errors or to indicate user intervention
- A command for sending signals: `kill`
  - Usage: `kill [-options] <pid>`
  - Use the `-s` option to send a specific signal
- Common signals
  - `2) SIGINT & 3) SIGQUIT`
    - Sent by the user as an interrupt
  - `9) SIGKILL & 15) SIGTERM`
    - Sent by the kill command
  - `4)SIGILL & 11) SIGSEGV`
    - Sent by the kernel upon error

```
rderue@gilbreth-fe02:~ $ kill -l
 1) SIGHUP   2) SIGINT     3) SIGQUIT  4) SIGILL
 5) SIGTRAP  6) SIGABRT    7) SIGBUS   8) SIGFPE
 9) SIGKILL 10) SIGUSR1   11) SIGSEGV 12) SIGUSR2
13) SIGPIPE 14) SIGALRM   15) SIGTERM 16) SIGSTKFLT
17) SIGCHLD 18) SIGCONT   19) SIGSTOP 20) SIGTSTP
21) SIGTTIN 22) SIGTTOU   23) SIGURG  24) SIGXCPU
25) SIGXFSZ 26) SIGVTALRM 27) SIGPROF 28) SIGWINCH
29) SIGIO   30) SIGPWR    31) SIGSYS  34) SIGRTMIN
...
```

Unix 201

Variables in BASH

Variables in BASH

Introduction to Shell variables

- A variable is a string that refers to some data
  - We say that we *initialize* a variable when we assign it a value
  - We say that we *reference* a variable when we use its value
- Variable initialization:
  - `VARIABLE=value`
- Variable reference:
  - `$VARIABLE`
- Variables are transient and only exist for the life of the shell

Environment Variables in BASH

- Environment variables are a specific type of variable
  - They are inherited by child shells
  - This implies that programs executed from a shell will have access to the environment variables
  - Often time these are critical to the operation of the shell or programs being executed by the shell
- Commands for interacting with environment variables
  - A command for viewing environment variables: `printenv`
    - Usage: `printenv [-options] [variable]`
  - A built-in for creating an environment variable: `export`
    - Usage: `export [variable[=value]]`

Variables in BASH

Important Environment Variables to Know

- Variables related to shell state
  - `$USER`: The owner of the shell
  - `$HOME`: The home directory of the shell
  - `$PWD`: The present working directory of the shell
  - `$SHELL`: The program your shell is running
  - `$PS1`: A variable which controls the prompt for your shell
- Variables related to search paths
  - Often these variables are colon delimited lists of directories which are checked in order
  - `$PATH`: A list of directories which will be checked for executable files

Unix 201

What Comes Next?

What Comes Next?

Upcoming Seminars

- Unix 202: February 10th

THANK YOU

Feel free to reach out to rderue@purdue.edu with questions.

Slides are posted at:
https://www.rcac.purdue.edu/training/unix201