Introduction
Processes
I/O redirection
Screen
Scripting
Network

Unix workshop: Let's Go Scripting

Zaki Mughal

Computational Biomedicine Lab
University of Houston

2013 Nov 20

Slides are at http://bit.ly/csgsa_unix_f2013.

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Introduction

Introduction
Processes
I/O redirection
Screen
Scripting
Network

This time

- processes
- more on I/O redirection
- screen (terminal multiplexer)
- advanced scripting
- network

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Processes

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Concepts

- process ID (PID) — associated with each process
- init — first process that starts when you boot (PID: 1)
- pstree — see tree of processes (forked off)

Job control

- when you run a process it is in *foreground*
- Ctrl+C — will interrupt the process
- Ctrl+Z — will suspend the process

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Job control

- when you run a process it is in *foreground*
- Ctrl+C — will interrupt the process
- Ctrl+Z — will suspend the process
- these are shell functions

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Job control

- when you run a process it is in *foreground*
- Ctrl+C — will interrupt the process
- Ctrl+Z — will suspend the process
- these are shell functions
- bg (shell builtin to background process)

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Job control

- when you run a process it is in *foreground*
- Ctrl+C — will interrupt the process
- Ctrl+Z — will suspend the process
- these are shell functions
- bg (shell builtin to background process)
- fg (and foreground)

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Job control

- when you run a process it is in *foreground*
- Ctrl+C — will interrupt the process
- Ctrl+Z — will suspend the process
- these are shell functions
- bg (shell builtin to background process)
- fg (and foreground)
- jobs (and see the processes under job control) (fg %1, bg %2)

Introduction
Processes
I/O redirection
Screen
Scripting
Network

pgrep

- you can find the process ID using:
- pstree -p
- ps (my processes in this terminal)
- ps -u (my processes)
- ps -elf (everyone's processes)

Introduction
Processes
I/O redirection
Screen
Scripting
Network

pgrep

- you can find the process ID using:
- `pstree -p`
- `ps` (my processes in this terminal)
- `ps -u` (my processes)
- `ps -elf` (everyone's processes)
- `pgrep`
- `pgrep -lf`
- the name will make sense more sense in a bit

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Signals

- internally, they send signals to the process
- Ctrl+C — SIGINT
- Ctrl+Z — SIGTSTP

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Signals

- internally, they send signals to the process
- Ctrl+C — SIGINT
- Ctrl+Z — SIGTSTP
- `man 7 signal # to find out more`
- also read `http://linusakesson.net/programming/tty/`

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Signals

- you can send signals using `kill`

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Signals

- you can send signals using `kill`
- more types of signals

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Signals

- you can send signals using `kill`
- more types of signals
- SIGTERM, SIGKILL

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Signals

- you can send signals using `kill`
- more types of signals
- SIGTERM, SIGKILL (SIGKILL can not be caught by process)

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Signals

- you can send signals using `kill`
- more types of signals
- SIGTERM, SIGKILL (SIGKILL can not be caught by process)
- numbers associated (`kill -l`)

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Signals

- you can send signals using `kill`
- more types of signals
- SIGTERM, SIGKILL (SIGKILL can not be caught by process)
- numbers associated (`kill -l`)
- SIGSEGV
- `int main() { char* a = 0; printf(*a); return 0; }`

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Signals using kill

- kill [pid] # default to send SIGTERM (15)

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Signals using kill

- kill [pid] # default to send SIGTERM (15)
- kill -TERM [pid] # same as default

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Signals using kill

- kill [pid] # default to send SIGTERM (15)
- kill -TERM [pid] # same as default
- `kill -9 [pid]` # send SIGKILL (DANGER)

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Exit status

- what happens when I kill?

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Exit status

- what happens when I kill?
- echo $?

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Exit status

- what happens when I kill?
- `echo $?`
- 0 indicates success

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Exit status

- what happens when I kill?
- echo $?
- 0 indicates success (int main { ... return 0; } ?)

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Exit status

- what happens when I kill?

- `echo $?`

- 0 indicates success (int main { ... return 0; } ?) (`return EXIT_SUCCESS`, `return EXIT_FAILURE`)

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Exit status

- what happens when I kill?
- `echo $?`
- 0 indicates success (int main { ... return 0; } ?) (`return EXIT_SUCCESS`, `return EXIT_FAILURE`)
- `$?` is a environment variable (more later)

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Exit status

- what happens when I kill?
- `echo $?`
- 0 indicates success (int main { ... return 0; } ?) (`return EXIT_SUCCESS`, `return EXIT_FAILURE`)
- `$?` is a environment variable (more later)
- after `kill -TERM`, we get 143

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Exit status

- what happens when I kill?
- `echo $?`
- 0 indicates success (int main { . . . return 0; } ?) (`return EXIT_SUCCESS`, `return EXIT_FAILURE`)
- `$?` is a environment variable (more later)
- after `kill -TERM`, we get 143
- 128 + 15 (the number for SIGTERM)

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Multiple commands

- `cat file1 ; cat file2`

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Multiple commands

- `cat file1 ; cat file2`
- these could fail (file1 does not exist)

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Multiple commands

- `cat file1 ; cat file2`
- these could fail (file1 does not exist)
- `cat file1 && cat file2`

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Multiple commands

- `cat file1 ; cat file2`
- these could fail (file1 does not exist)
- `cat file1 && cat file2`
- only run when the first is successful

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Multiple commands

- `cat file1 ; cat file2`
- these could fail (file1 does not exist)
- `cat file1 && cat file2`
- only run when the first is successful
- `cat file1 || echo "could not cat!"`

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Multiple commands

- `cat file1 ; cat file2`
- these could fail (file1 does not exist)
- `cat file1 && cat file2`
- only run when the first is successful
- `cat file1 || echo "could not cat!"`
- only run second when the first is NOT successful

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Sleep

- ```
  date && sleep 5m && echo "Nap time is over!" &&
  date
  ```

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Sleep

- `date && sleep 5m && echo "Nap time is over!" && date`
- BONUS COMMAND: `date`

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Where are all these binaries anyway?

- `echo $PATH`
- another environment variable

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Where are all these binaries anyway?

- `echo $PATH`
- another environment variable
- colon-separated paths
- which ls
- which vi
- which -a matlab

Introduction
Processes
I/O redirection
Screen
Scripting
Network

A taste of scripting!

- `vi ~/.bashrc`
- `alias ..='cd ..'`
- `alias l='ls -CF'`
- `alias c='cd'`
- shortcuts!

Introduction
Processes
I/O redirection
Screen
Scripting
Network

A taste of scripting!

- vi ~/.bashrc
- alias ..='cd ..'
- alias l='ls -CF'
- alias c='cd'
- shortcuts!
- disable with a backslash:
- \ls

Introduction
Processes
I/O redirection
Screen
Scripting
Network

A taste of scripting!

- you can set environment variables here
- set the $PATH and other environment variables
- export PATH= /script:$PATH # prepend
- export EDITOR=vim
- export PAGER=less

Introduction
Processes
I/O redirection
Screen
Scripting
Network

A taste of scripting!

- you can set environment variables here
- set the $PATH and other environment variables
- export PATH= /script:$PATH # prepend
- export EDITOR=vim
- export PAGER=less
- speaking of paths, spaces are bad for scripting — don't use spaces in your filenames

Introduction
Processes
I/O redirection
Screen
Scripting
Network

I/O redirection

Introduction
Processes
I/O redirection
Screen
Scripting
Network

File redirection

- which ls && echo "ls is available"

Introduction
Processes
I/O redirection
Screen
Scripting
Network

File redirection

- which ls && echo "ls is available"
- extra output

Introduction
Processes
I/O redirection
Screen
Scripting
Network

File redirection

- which ls && echo "ls is available"
- extra output
- which ls > /dev/null && echo "ls is available"

Introduction
Processes
I/O redirection
Screen
Scripting
Network

File redirection

- which ls && echo "ls is available"
- extra output
- which ls > /dev/null && echo "ls is available"
- stdout goes to /dev/null

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Back to cat

- Remember `cat file`

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Back to cat

- Remember `cat file`

- `cat`

- read from stdin

- ...

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Back to cat

- Remember `cat file`
- `cat`
- read from stdin
- ...
- hit Ctrl+D (sends EOF, zero-bytes left to read)

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Back to cat

- Remember `cat file`
- `cat`
- read from stdin
- ...
- hit Ctrl+D (sends EOF, zero-bytes left to read)
- then outputs the stdin to stdout

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Make every program a filter

- `cat file` is equivalent to:
- `cat < file`
- many programs follow this principle

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Make every program a filter

- `cat file` is equivalent to:
- `cat < file`
- many programs follow this principle
- `wc`
- word count

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Make every program a filter

- `cat file` is equivalent to:
- `cat < file`
- many programs follow this principle
- `wc`
- word count
- `wc -l < essay.txt`
- how many lines?

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Make every program a filter

- shuf (shuffle the files)
- grep (match using regex)
- less (navigate long output)
- head, tail (see n lines of start/end of input)
- sort
- uniq (unique lines)

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Pipes

- instead of files, send output of one command to the output of another
- let's look at some examples:
- ls | grep −o '\.[a−z][^.]*$' | sort | uniq

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Pipes

- instead of files, send output of one command to the output of another
- let's look at some examples:
- ls | grep −o '\.[a−z][^.]*$' | sort | uniq

  ```
  ls | grep −o '\.[a−z][^.]*$' \
  | sort | uniq \
  | wc −l
  ```

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Pipes

- List all file extensions (starting with a–z)

  ls | grep −o '\.[a−z][^.]*$' | sort | uniq

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Pipes

- List all file extensions (starting with a–z)

  ```
  ls | grep -o '\.[a-z][^.]*$' | sort | uniq
  ```

- How many of them?

  ```
  ls | grep -o '\.[a-z][^.]*$' \
     | sort | uniq \
     | wc -l
  ```

- we're using the backslash for line-continuation

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Pipes

- Which ones are there the most of?

```
ls  |  grep −o  '\.[a−z][^.]*$'  \
    |  sort    |  uniq −c  \
    |  sort −n  |  tail
```

Screen

Introduction
Processes
I/O redirection
Screen
Scripting
Network

- GNU Screen (you may have to install it)
- terminal multiplexer — multiple terminals in one terminal
- persistent session

Introduction
Processes
I/O redirection
Screen
Scripting
Network

- GNU Screen (you may have to install it)
- terminal multiplexer — multiple terminals in one terminal
- persistent session
- `screen -S session_name`
- Ctrl+A (the command prefix)
- Ctrl+A a (actual Ctrl+A)
- Ctrl+A c (create a new window)
- Ctrl+A n (next window)
- Ctrl+A p (previous window)
- Ctrl+A d (detach)
- `screen -d -r session_name # reattach`
- Ctrl+A ? (help)

Introduction
Processes
I/O redirection
Screen
Scripting
Network

- Ctrl+A S (splits into regions [horizontal])
- Ctrl+A Tab (move between regions)
- Ctrl+A | (splits into regions [vertical])

Introduction
Processes
I/O redirection
Screen
Scripting
Network

- Ctrl+A S (splits into regions [horizontal])
- Ctrl+A Tab (move between regions)
- Ctrl+A | (splits into regions [vertical])
- NOT Ctrl+A s !!!!
- that will stop flow control to the terminal
- fix by doing Ctrl+A q

Introduction
Processes
I/O redirection
Screen
Scripting
Network

- Ctrl+A S (splits into regions [horizontal])
- Ctrl+A Tab (move between regions)
- Ctrl+A | (splits into regions [vertical])
- NOT Ctrl+A s !!!!
- that will stop flow control to the terminal
-      fix by doing Ctrl+A q
- similar to Ctrl+S, Ctrl+q in terminal itself
- XOFF/XON, see
  http://unix.stackexchange.com/questions/12107/,
  https:
  //en.wikipedia.org/wiki/Software_flow_control

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Scripting

Introduction
Processes
I/O redirection
Screen
Scripting
Network

- Script files begin with the line (shebang)

- #!/bin/sh

- chmod u+x myscript.sh # make executable
- ./myscript.sh # run it

Introduction
Processes
I/O redirection
Screen
Scripting
Network

- we can put any of the commands we used before
- `count_extensions.sh`
- fc (open last command in editor)
- Ctrl+X Ctrl+C (open current command in editor)

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Command substitution

- ls −l \$(which ls)

  ls −l `which ls` # backticks (same key as tilde

- find the location of ls and give

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Writing a more complex script

- watch ls # repeatedly run command

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Writing a more complex script

- watch ls # repeatedly run command
- write your own watch (well, slightly modified)

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Writing a more complex script

```sh
#!/bin/sh
if ! which sleep clear > /dev/null; then
  # check that sleep and clear are in path
  echo "Require: sleep, clear"
  exit 1
fi
```

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Writing a more complex script

```sh
#!/bin/sh
if ! which sleep clear > /dev/null; then
  # check that sleep and clear are in path
  echo "Require: sleep, clear"
   exit 1
fi
while [ "$?" = 0 ]; do
  # man 1 test # (test for files, dirs, etc.)
  # run the args: "$1", "$2"
  "$@" \
     && sleep 2 \
     && clear # clear the screen
done
```

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Managing lots of files

```
for i in `ls`; do
  echo "$i";
done
```

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Managing lots of files

```
for i in `ls`; do
  if [ -f "$i" ] \
      && echo "$i" | grep -iq ".tex$"; then
    # check if it is a file
    # and that it ends in tex
    echo "Found a TeX file: $i";
  fi
done
```

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Managing lots of files

```
find -type f -iname '*.tex'
# same but does it recursively
```

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Managing lots of files

```
find -type f -iname '*.tex' \
  -exec echo "Found a TeX file: {}" \;
```

Introduction
Processes
I/O redirection
Screen
Scripting
Network

Managing lots of files

```
# the most robust way
# -print0 : null separated
find -type f -iname '*.tex' -print0 \
    | xargs -I{} -0 echo "Found a TeX file: {}"
# xargs takes stdin and passes it as an argument
# to a command
```

More info at the Advanced Bash-Scripting Guide:
http://tldp.org/LDP/abs/html/

Network

Introduction
Processes
I/O redirection
Screen
Scripting
Network

- ssh, sftp — remote terminal and download files

- ```
  diff <(ssh FAR_AWAY ls ~/sw_projects) \
       <(ls ~/sw_projects)
  # find out which files are different
  # in a remote directory
  # uses process substitution <() (mkfifo)
  # + ssh
  # + diff
  ```

Introduction
Processes
I/O redirection
Screen
Scripting
Network

- ssh, sftp — remote terminal and download files

- ```
  diff <(ssh FAR_AWAY ls ~/sw_projects) \
       <(ls ~/sw_projects)
  # find out which files are different
  # in a remote directory
  # uses process substitution <() (mkfifo)
  # + ssh
  # + diff
  ```

- sshfs — mount remote computers as a file system

Introduction
Processes
I/O redirection
Screen
Scripting
Network

- ssh, sftp — remote terminal and download files

- ```
  diff <(ssh FAR_AWAY ls ~/sw_projects) \
       <(ls ~/sw_projects)
  # find out which files are different
  # in a remote directory
  # uses process substitution <() (mkfifo)
  # + ssh
  # + diff
  ```

- sshfs — mount remote computers as a file system
- rsync, unison — unidirectional and bidirectional backups/syncing

Introduction
Processes
I/O redirection
Screen
Scripting
Network

- ssh, sftp — remote terminal and download files

- ```
  diff <(ssh FAR_AWAY ls ~/sw_projects) \
       <(ls ~/sw_projects)
  # find out which files are different
  # in a remote directory
  # uses process substitution <() (mkfifo)
  # + ssh
  # + diff
  ```

- sshfs — mount remote computers as a file system
- rsync, unison — unidirectional and bidirectional backups/syncing
- wget — mirror FTP/HTTP