

# 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](http://bit.ly/csgsa_unix_f2013).

# Introduction

# This time

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

# Processes

# Concepts

- process ID (PID) — associated with each process
- `init` — first process that starts when you boot (PID: 1)
- `ps tree` — 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

## 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



## 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)

## 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)

## 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)

## 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

- 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

# Signals

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

# 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/>

# Signals

- you can send signals using `kill`



# Signals

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

# Signals

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

# Signals

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

# Signals

- you can send signals using `kill`
- more types of signals
- SIGTERM, SIGKILL (SIGKILL can not be caught by process)
- numbers associated (`kill -1`)

# Signals

- you can send signals using `kill`
- more types of signals
- SIGTERM, SIGKILL (SIGKILL can not be caught by process)
- numbers associated (`kill -1`)
- SIGSEGV
- ```
int main() { char* a = 0; printf(*a); return 0; }
```

## Signals using kill

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

## Signals using kill

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

## Signals using kill

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



## Exit status

- what happens when I kill?

## Exit status

- what happens when I kill?
- echo \$?

## Exit status

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

## Exit status

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

## Exit status

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

## 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)

## 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

## 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)



## Multiple commands

- `cat file1 ; cat file2`

## Multiple commands

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

## Multiple commands

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

## Multiple commands

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

## 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!"`

## 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

# Sleep

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

# Sleep

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



## Where are all these binaries anyway?

- `echo $PATH`
- another environment variable

## Where are all these binaries anyway?

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

## A taste of scripting!

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

## A taste of scripting!

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

## 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

## 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

## I/O redirection

## File redirection

- which ls && echo "ls is available"



# File redirection

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

## File redirection

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

## File redirection

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

## Back to cat

- Remember `cat file`

## Back to cat

- Remember `cat file`
- `cat`
- read from `stdin`
- ...

## Back to cat

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

## 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`

## Make every program a filter

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



## Make every program a filter

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

## 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?

## 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)

# 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`

# 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
```

# Pipes

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

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

# 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

# Pipes

- Which ones are there the most of?

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



# Screen

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

- 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)

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

- 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

- 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](https://en.wikipedia.org/wiki/Software_flow_control)

# Scripting

- Script files begin with the line (shebang)
- `#!/bin/sh`
- `chmod u+x myscript.sh` # make executable
- `./myscript.sh` # run it



- 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)

## Command substitution

- `ls -l `$(which ls)``

`ls -l 'which ls'` # backticks (same key as tilde`

- find the location of `ls` and give

## Writing a more complex script

- `watch ls #` repeatedly run command

## Writing a more complex script

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

## Writing a more complex script

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

## Writing a more complex script

```
#!/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
```

## Managing lots of files

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

## 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
```



## Managing lots of files

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

## Managing lots of files

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

## Managing lots of files

```
# the most robust way
# -print0 : null separated
find -type f -iname '*.tex' -print0 \
  | xargs -l{} -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

- 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
```

- 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

- 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



- 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