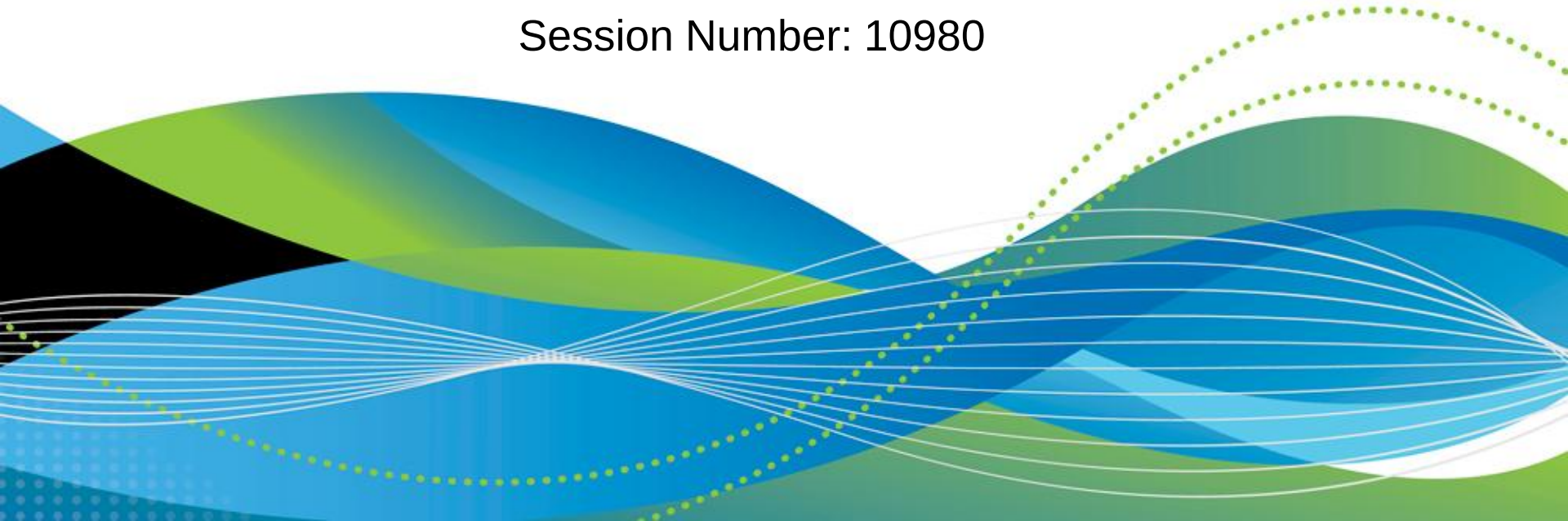
#SHAREorg

SHARE
Technology · Connections · Results

Killing Zombies, Breaking Pipes, and Other UNIX Shenanigans

Brandon Tweed
CA Technologies

3/13/2012
Session Number: 10980



Jurassic Park (1993)

Some Background

- Jurassic Park appeared in theaters in 1993.
- That was 19 years ago.
- The girl in the picture is Ariana Richards.
- She will be 33 this year.
- (I used IMDB. I'm not stalking Ariana Richards)
- Mark Zuckerberg, creator of facebook is 27.

Where Is This Going?

- Source: http://www.filmdetail.com/2011/05/13/from-jurassic-park-to-the-social-network-joseph-mazzello-unix-facebook/
- Facebook runs on Linux (UNIX-like OS)
- Knowing UNIX helped a bit?

The Point

- UNIX awareness was increasing in 1993.
- A lot has happened since 1993
  - Google's Android OS (Linux-based)
  - Apple's iOS and OS-X, both derived from UNIX, are also ubiquitous.
  - Java appeared in 1995.
  - A generation of developers have grown up with UNIX, Linux, and Windows available.
- UNIX and Linux are not a fad.
- (Unless you believe a 43-year-old OS counts as a fad)

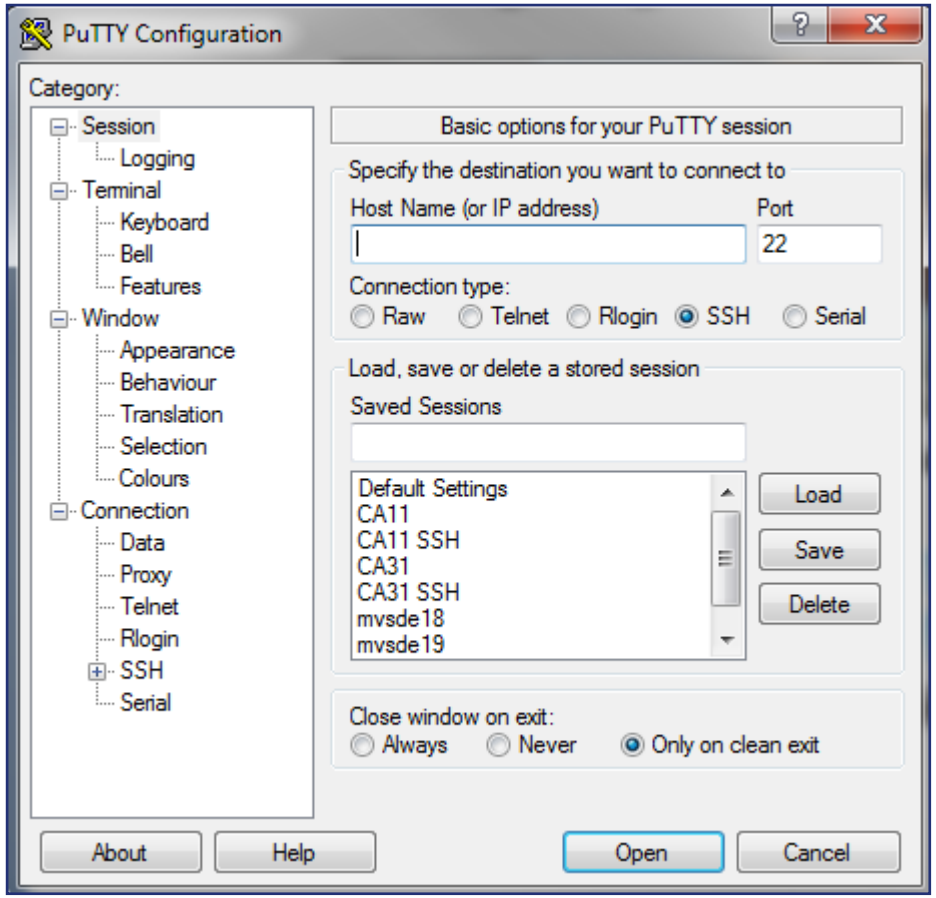
Today's Presentation

- Cover some intermediate-level UNIX topics, become more acquainted with UNIX.
- Topics:
  - Process Management
  - Zombie Processes
  - Job Management
  - Pipes
  - Redirection
  - Shenanigans

Intermediate Level, You're Expected to Know …

- Basic UNIX commands for listing directories and navigation
- UNIX file system and UNIX permissions
- How to access UNIX (via OMVS or preferably puTTY)
- References to keyboard shortcuts assume a puTTY-like environment, not OMVS.
- The same things can be done from OMVS, but they require some terminal set-up.

puTTY Config Crash Course



1. Provide the hostname
2. Select the connection type (typically Telnet or SSH)
3. Ensure that the TCP/IP Port value is correct
4. Click Open
5. Provide userid
6. Provide password
7. UNIX Prompt! Done.

Getting Started - Useful UNIX Commands

- more – displays a text file page-by-page
- grep – search a text file line-by-line for matches to a regex
- wc – word count (can also be used for getting a line count)
- ps – display status information about processes
- find – search the file system for a file
- diff – compare two files line-by-line
- nohup – Run a program that won't terminate if your terminal is disconnected.
- who – see who's logged on
- df – report free space for a file system

The “man” Command

- man – access the UNIX manual pages for any given command.
- Example: man ls
- Gives the manual pages for the ls command
- Yes, “man man” works.
- Note that “man woman” does not.
- Also available: IBM manuals, Google, Bing, StackExchange, forums, and many other resources on the web.

Topic: Killing Zombies

UNIX Processes

- Each program running under UNIX is a process
- Every process has a process id (PID)
- Every process has a parent process id (PPID)

Your Processes

```
CA11:/u/users/framewrk/twebr01> ps
        PID TTY          TIME CMD
        524 ttyp0001     0:00 /bin/sh
   50332181 ttyp0001     0:00 /bin/ps
```

- You will have at least one process running under your userid if you login to USS.
- The process is for your shell (/bin/sh)
- The list also shows the process for ps as it runs.

Terminating Processes

- The ‘kill’ command can be used to end a running process.
- As always, use with caution.
- Sends the signal SIGTERM to the process.
- This is the polite way of asking the process to end.
- Kill requires the PID of the process you want to terminate.

How to Kill

1) Use ‘ps’ to get the PID of the process you wish to kill.

```
       PID TTY        TIME CMD
       524 ttyp0001   0:00 /bin/sh
  16777763 ttyp0001   0:00 /bin/ps
  67109413 ttyp0001   0:01 /bin/find
```

2) Assertively issue the ‘kill’ command.

```
CA11:/u/users/framewrk/twebr01> kill 67109413
CEE5205S The signal SIGTERM was received.
```

Terminating with Extreme Prejudice

- In rare situations, a process might ignore the SIGTERM signal.
- This is usually unintentional behavior.
- Use the –K switch of the kill command to send a “superkill” signal to the process.
- This should be used as a last resort.
- From the man pages, “The process is ended with a non-retryable ABEND”

What's a Zombie Process?

It all starts with forking …

Fork

- UNIX provides a system call named fork()
- fork() makes an identical copy of the current process
- the original process is called the parent
- the new process is called the child
- the child's PPID is the PID of the parent
- The practice is known as "forking"
- In conversations you'll hear about "forking a child" or "forking children". Try not to be disturbed by this.

How Zombies Are Made (Forking Without Care)

- Zombies are created through poor programming, usually during development.
- When a programmer issues a fork() in her program, they must issue a wait().
- If no wait() is issued, the child process lingers when it completes execution.
- The process remains in the process table but really isn't doing anything.

Example Zombie Creation

- I’ve written a program that creates zombies.
- Written in C, compiled in UNIX.
- zombies.c provided as an example.
- Link to source:
- http://dl.dropbox.com/u/39171790/zombies.c

Process Table Full of Zombies

```
CA11:/u/users/framewrk/twebr01> ps -ef
     UID        PID        PPID  C    STIME TTY       TIME CMD
 TWEBR01        524    33554957  - 14:54:58 ttyp0001  0:00 sh -L
 TWEBR01   33554957    50332171  - 14:54:58 ?         0:00 sshd -i
 TWEBR01   33555000    83886843  - 15:58:36 ?         0:00
 TWEBR01   33555002    83886843  - 15:58:36 ?         0:00
 TWEBR01   33555003    83886843  - 15:58:36 ?         0:00
 TWEBR01   33555004    83886843  - 15:58:36 ?         0:00
 TWEBR01   33555005    83886843  - 15:58:36 ?         0:00
 TWEBR01   33555006    83886843  - 15:58:36 ?         0:00
 TWEBR01   33555007    83886843  - 15:58:36 ?         0:00
 TWEBR01   33555008    83886843  - 15:58:36 ?         0:00
 TWEBR01   33555010    83886843  - 15:58:36 ?         0:00
 TWEBR01   33555011    83886843  - 15:58:36 ?         0:00
 TWEBR01   33555012    83886843  - 15:58:36 ?         0:00
 TWEBR01   33555013    83886843  - 15:58:36 ?         0:00
 TWEBR01   33555014    83886843  - 15:58:36 ?         0:00
 TWEBR01   33555015    83886843  - 15:58:36 ?         0:00
 TWEBR01   33555016    83886843  - 15:58:36 ?         0:00
 TWEBR01   33555017    83886843  - 15:58:36 ?         0:00
 TWEBR01   33555019    83886843  - 15:58:36 ?         0:00
 TWEBR01   83886843         524  - 15:58:36 ttyp0001  0:00 a.out
 TWEBR01   67109631         524  - 15:59:12 ttyp0001  0:00 ps -ef
 TWEBR01   16777984    83886843  - 15:58:36 ?         0:00
 TWEBR01   33555201    83886843  - 15:58:36 ?         0:00
 TWEBR01   33555202    83886843  - 15:58:36 ?         0:00
```

The Big Reveal: You Can't Kill Zombies

But Everything is Fine!

- Zombies can’t be killed?!
- The secret: KILL THE PARENT.
- When the parent dies the init process inherits any “orphaned” child processes.
- The init process periodically issues a wait() in order to clean up any zombies.

Zombie Clean-Up Procedures

1. Use ‘ps’ to get the list of processes
2. Locate the zombie and determine it’s PPID
3. Issue ‘kill’ to kill the parent
4. Let the init process take care of the rest

Prevention: Setting User Process Limit

- Typically done with ulimit command on some UNIX variants
- For z/OS, this done via the OMVS segment in a user's security profile.
- PROCUSERMAX limits the max number of processes for the UID.
- More info here: http://publib.boulder.ibm.com/infocenter/zos/v1r13/topic/com.ibm.zos.r13.bpxb200/ussoel.htm#ussoel

Topic: UNIX Job Management

- In UNIX a job is a process associated with your terminal session.
- Jobs can run in the foreground or background.

Foreground Processing

- Good for commands/processes that run and complete quickly
- “hold on” to your command prompt while they are executing.
- Sort of like online processing with TSO.

Background Processing

- Appropriate for things that take a long time to finish.
- Otherwise you must wait for the command prompt to appear once the command finishes.
- Similar to batch processing. You run the process in the “background” and do other things while you’re waiting for it to finish.
- Alternative: Have multiple sessions open

Running a Command in the Background

- Simple: Append an ampersand (&) to the end of the command. It will run in the background.
- Example: find / -name profile -print &

Example Background Command

find / -name profile -print &

- searches for a file with the name “profile” in it
- starts at the root and prints the results to stdout if a match is found.
- This will interrupt any interactive commands you are running to display output.
- Background processes should usually have their output (stdout and stderr) redirected to a file to avoid this.

Displaying Jobs (Processes)

- The jobs command will give you a list of processes associated with your session.
- Each process listed will have an associated “jobid”
- The jobid can be used with commands for managing jobs (such as kill, fg, or bg).

Suspending a Foreground Process

- A long-running process in the foreground can be “suspended” by pressing Ctrl+Z.
- This sends a signal to the process that will pause it. The process can be resumed at a later time.
- The signal is SIGSTP

Foreground Process Suspension Example

- Try this from a puTTY session, press Ctrl+Z while the command is running.

```
[1] + Stopped (SIGTSTP) find / -name profile
CA11:/u/users/framewrk/twebr01>
```

- Issuing the “jobs” command will show that the command is stopped.

```
CA11:/u/users/framewrk/twebr01> jobs
[1] + Stopped (SIGTSTP) find / -name profile
```

Resuming a Background Job

- To bring a background job back to foreground processing, you can issue the “fg” command.
- If you do not specify a job name, it will bring the most recently suspended job to the foreground.

Running a Job In The Background (Another Way)

- Use the ‘bg’ command
- bg *jobid*
- If no jobid is specified, uses the most recently stopped job.
- You could run it in the foreground, suspend it, and then issue bg.
- But there’s already a better way.

And Now For Something Completely Different….

"It is a mistake to think you can solve any major problems just with potatoes."

Douglas Adams,

Life, the Universe, and Everything

Breaking Pipes

- If you experiment with UNIX long enough, you'll encounter this situation although you may not understand why.

```
CA11:/u/users/framewrk/twebr01> ps -ef | head
     UID          PID    PPID  C    STIME TTY          TIME CMD
  STCSYS            1       0  -   Mar 03 ?        2:18 BPXPINPR
  STCSYS            2       1  -   Mar 03 ?        0:00 EZBREINI
  STCSYS            3       1  -   Mar 03 ?        0:00 EZBREUPS
  STCSYS            4       1  -   Mar 03 ?        0:00
OMVSDFLT     16777221       1  -   Mar 03 ?        0:00 CEAPSRVR
  STCSYS            7       1  -   Mar 03 ?        0:10 CAS9ATGS
  STCSYS            9       1  -   Mar 03 ?        0:01 CAS9PDPM
  STCSYS           10       1  -   Mar 03 ?        3h08 ISTMGCEH
  STCSYS           11       1  -   Mar 03 ?        0:00 CAS9PDGM
CEE5213S The signal SIGPIPE was received.
```

Everything Is A File

All UNIX processes typically open three files by default:

1. Standard Input (stdin) – Typically direct input from the terminal, but can come from other places.
2. Standard Output (stdout) – Typically output written to the terminal.
3. Standard Error (stderr) – Also typically written to the terminal unless redirected elsewhere.

Related Topic - Redirection

- Earlier we mentioned that each process works with stdout, stderr, and stdin.
- Each of these is a file that a process opens automatically when the process begins executing.
- The output and input of files can be “redirected” so that it comes from or goes to another place.

Topic: Redirection

- Commands refer to stdin, stdout, and stderr by file descriptor numbers:
- 0 – stdin
- 1 – stdout
- 2 – stderr

Redirecting Standard Output

- Save output from a program to a file
- Example: ls –l > listing.txt
- Saves the output of the command to a text file, listing.txt.
- If this file exists, it will be overwritten
- If it does not exist, it will be created
- (as long as the permissions allow this)

Redirecting Standard Input

- Some commands read from standard input.
- You may want to provide a file as input instead of typing data.
- Better example in UNIX Shenanigans.

Redirecting Standard Input (Example 1)

- cat < file.txt
- This feeds the file “file.txt” as input to the ‘cat’ command.
- Example is silly because you can provide a file name as an argument to the cat command.

Redirecting Standard Input (Example 2)

Example: sh < script.sh

- create a new shell instance with the sh command
- feed it script.sh as the input.
- running a shell script but in a non-conventional way

Remembering How to Redirect

- For simple redirection, pretty easy to remember
- The “<“ or “>” points to where the data is going.
- who > file.txt
- cat < file.txt

Redirecting ALL The Output

Example: command > file.txt 2>&1

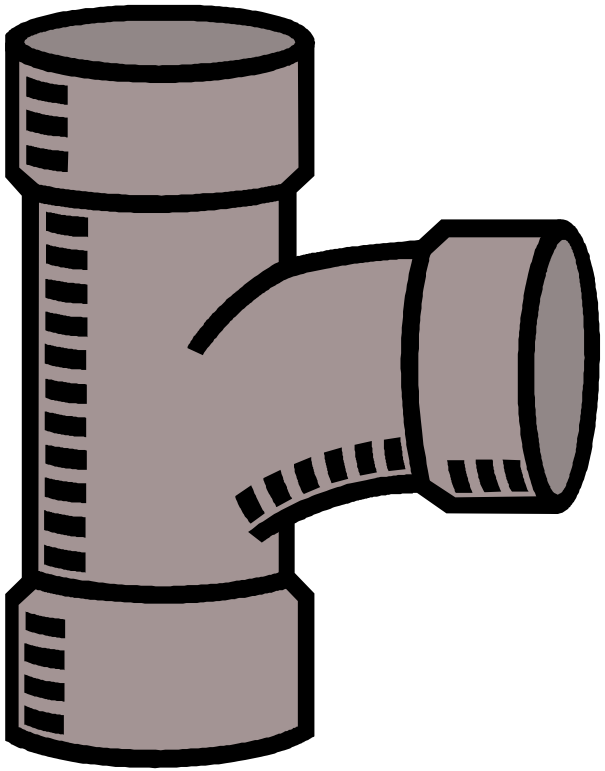
- Redirects stdout to file.txt
- Redirects stderr to stdout
- Result: Both stdout and stderr written to same file.
- ‘command’ can be any UNIX command

Next: Pipes

"A common mistake that people make when trying to design something completely foolproof is to underestimate the ingenuity of complete fools."

- Douglas Adams, The Hitchiker's Guide to the Galaxy

Pipes



- Transfer data between UNIX processes. The output of one process becomes the input to another.
- Alternative to pipes: Create a file as output from each command, then feed the file as input to a subsequent command.

Using Pipes

String together commands using the '|' character (X'4F' or 0x4F).

Example:
who | grep *userid*

Output of the 'who' command is given as input to the grep command. Grep will search it for a match for <userid>.

A Side Note On Grep

- Grep is an incredibly useful tool.
- Learning regular expressions makes it even more powerful.
- Regular expressions aren't the purpose of this session.
- There are many great books and online resources that explain regular expressions very well.
- Regular expressions are also useful for other tools, commands, and editors (sed, awk, vi, et cetera)

Looking Closer

Example:
who | grep *userid*

1. The who command writes its output to stdout (normally goes directly to the terminal).
2. The pipe forces the output of the who command to become the input of the next command, grep. (The stdout of the ‘who’ command becomes the stdin of the grep command.)
3. grep reads the input and uses it for its search

More Pipe Examples

ls -l | more

Display the output of a long listing page-by-page.

who | wc -l

Count the number of UNIX sessions.

who | sort

Sort the output of who (displays lines in alphabetical order)

More Pipe Examples (2)

ls -l | grep ‘^d’

Obtain a listing that includes only directories.

find . -name *.c | grep hello

Search recursively from the current directory downward for files with names that end in “.c”then filter that list to include only filenames with “hello” in them. (Modify the command to make it useful)

More Pipe Examples (3)

ps -a -o pid= | wc -l

Get the list of all processes (that you're allowed to know about) and give the count of the list produced. End result: count of processes running that you're allowed to see by security.

env | grep '^a'

Search all UNIX environment variable definitions to find names that begin with 'A' or 'a'.

What's a broken pipe?

- One of the commands in a string of piped commands has its input (stdio) or output (stdout) closed or EOF
- Example scenario:
  - a command pipes its output (stdout) into another command
  - the receiving command closes stdin before all the input is received
  - the pipe is "broken", the sender has nowhere to write the data

Example C Program

```
#include<stdio.h>
int main(void) {
fclose(stdin);
return 0;
}
```

- The program closes standard input (stdin) immediately and then ends with an RC = 0.
- A program feeding the right amount of input to this through a pipe will result in SIGPIPE (broken pipe)

Example Run

```
CA11:/u/users/framewrk/twebr01> ls -la | a.out
CEE5213S The signal SIGPIPE was received.
```

- The compiled program (a.out) immediately closes stdin when it starts.
- The ls command tries to continue writing to stdout, which has been closed, and receives the SIGPIPE signal and terminates processing.

Are Broken Pipes A Problem?

- Yes, if you're developing a program that doesn't handle stdin/stdout/stderr correctly.
- When working with standard UNIX commands, not really a big issue.
- Correct the command, try again, and move on with life.
- Now you know more about what SIGPIPE actually means.

Other UNIX Shenanigans

UNIX Shenanigans

- Experimenting with UNIX commands is often the best way to learn about them.
- This can yield some unexpected results.
- I don’t endorse doing things that are actually malicious/harmful to your system.
- However, I do take great pleasure from causing harmless mischief.
- Don’t try this at home unless you claim full personal responsibility for it.

UNIX Shenanigans - wall

wall - Send a broadcast message to all logged-in UNIX users.

- Requires superuser or permission to write to the terminals of others
- Takes input from stdin. Use a pipe to feed whatever file you like to the wall command.
- Maybe try sending War and Peace to all the users?

UNIX Shenanigans – logger

logger - send a message to the console log

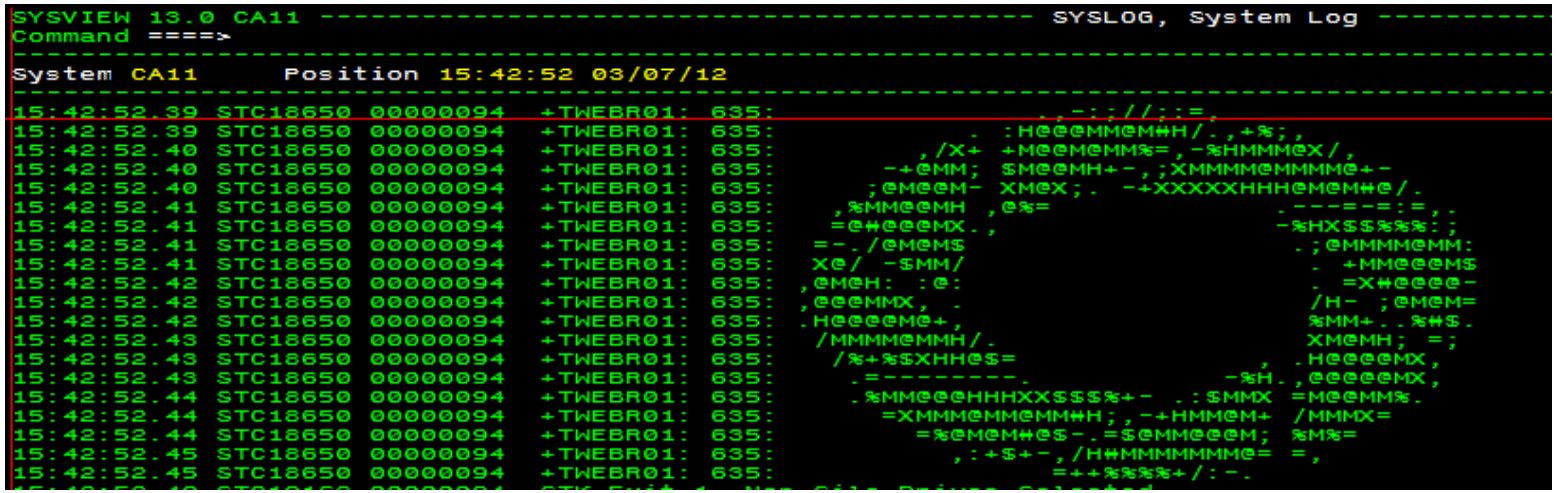
```
CA11:/u/users/framewrk/twebr01> logger
danger will robinson!! DANGER!
```

```
SYSVIEW 13.0 CA11 ---------- SYSLOG, System Log ----------
Command ====>
------------------------------------------------------------------------------ Lv1
System CA11     Position 15:31:37 03/07/12
------------------------------------------------------------------------------
15:31:37.14 STC18649 00000094  +TWEBR01: 229: danger will robinson!! DANGER!
15:31:37.15 JOB18877 00000094  +GSVX014I (API.TOBSH02) TOBSH02 entered command TYPE CONTINUE HOME
                               NEWLINE 4 EREOF DATA '1'
15:31:37.31 JOB18905 00000094  TSS7053I Default ACID <BATCHDEF> Assigned
```

UNIX Shenanigans - logger

Send an entire file to the console.

```
CA11:/u/users/framewrk/twebr01> logger -f funwithscience.txt
```



UNIX Shenanigans – kill

Not fond of someone? Kick them off the system!

- Typically requires superuser
- Traceable

1. Use ps to find the PID of their shell (grep the output if you have to)
2. Issue the kill command with the PID of the shell

Example: kill -HUP 33554862

UNIX Shenanigans – talk

talk *userid*

- Establishes a two-way chat session.
- Requires acceptance from whoever you're trying to chat with.
- Disruptive to work and agitating if done repeatedly.

UNIX Shenanigans – write

write *userid terminal*

- More assertive than talk
- Input comes from stdin
- Written to receiver's terminal without asking
- Very disruptive! (Much like e-mail and instant messenger)

Example:

cat file.txt | write twebr01 ttyp0020

Other Mean Tricks

Add ‘umask 777’ to another user’s .profile.

- Sets the permissions for newly created files or directories to 000.
- Won’t be able to read, write, or execute files or directories created.
- Directories created this way can’t be listed by the user .
- Change won’t be noticed unless user checks his/her profile.
- Even better, add this to /etc/.profile

Other Mean Tricks

Make a user think all her files are gone.

1) Create an empty zfs (size doesn't matter), use ISHELL for conveniece
2) Mount it on top of their home directory, use ISHELL if you're not sure how to use the mount command
3) User's home directory will appear to be empty. Files are still there but inaccessible until the zfs is unmounted.

Other Mean Tricks

Prevent a user from logging in – add “exit” statement to their .profile.

- Requires write access to .profile or superuser
- When the session starts, it will immediately end.
- Very annoying.
- Use ISPF option 3.17 to open the file outside of USS and remove the exit statement if you try this with your own .profile.

Why Would You Do This??

- At some point, someone will knowingly or accidentally do something to cause trouble in UNIX.
- Real Example:
  - Statement added to /etc/.profile that causes an error (unintentional)
  - Error prevents user's .profile loading for all users
  - Broke a number of UNIX build processes.
- When something like this happens, you'll be equipped to diagnose/fix problems.
- By causing trouble, you also learn how to fix problems

Conclusion

- Common UNIX commands
- puTTY set-up
- Processes
- Zombies
- Pipes
- Job Management
- I/O Redirection
- Causing Trouble

Mischief Managed (Q&A)