

# Advanced Unix Minicourse

# ptools Overview

- ptools allow you to examine and control the state of a process
- they use the `procfs`, an amazingly cool system for accessing and screwing with the state of a process.
- most of the ptools live in `/usr/bin/`

## pgrep

- The most useful ptool is `pgrep`
- Use `pgrep` to search through all the processes on the system to find those you want to mess with

```
> pgrep -u kit xdvi  
17224  
15714
```

- there are a lot of options to `pgrep` have a look at the man page (yes, now)
- `pkill` is like `pgrep`, but can be used to signal (kill) processes; this will force all your `xdvi`'s to reload by sending `SIGUSR1` to all my running `xdvi` processes

```
> pkill -USR1 -u kit xdvi
```

## screen **Overview**

- screen(1) lets you have multiple virtual terminal windows active at the same time
- Useful for remote login — you can run several tasks in different virtual windows
- Copy and paste between terms without using the mouse
- Detach from programs that you want to leave running for a long time

# Running screen

- To run screen just type screen in a shell
- Remember that all commands are prefixed with the control character, ^a
- To make a new window, use ^a c
- To move to the next and previous windows, use ^a n and ^a p respectively
- To view a list of windows, use ^a w
- To go directly to a particular window, use ^a *window-number* (e.g. ^a 0)

## Intermediate screen

- Copying and pasting within screen is *really* easy and useful
  - just type `^a [` to enter copy/scrollback mode. To select a region, use the spacebar to begin, vi keys to navigate, and the space bar to end.
  - To paste, just type `^a ]`
- To change the title of a window, type `^a a` and then type the name you want at the prompt
- To lock your screen session (just like `xlock`) type `^a x`
- For a complete list of commands, type `^a ?`

## Detaching and reattaching

- Let's say that you want to log out, but leave the programs in your screen session running. Type `^a d`, and you will be returned to the prompt where you ran screen from. Now you can log out and your screen will still be running.
- To reattach to the session, run `screen -r` from another shell on the same machine. You can also force a screen to detach remotely with `screen -d`.
- It's also possible to use multiple display mode to attach to the same screen session from more than one location. Use `screen -x`.
- Try starting some xterms like this:  
`xterm -e screen -x`  
After you create some new windows, you should now be able to copy and paste between xterms without using the mouse!

# The CS Network

- All computers in the department are equal, but some are more equal than others
- All the computers in the SunLab have a 1.5GHz Athlon processor with 512 Mb of RAM, a nice 18-inch flat panel monitor (don't put fingers on them!), an NVIDIA Geforce3 Ti200 with 128 Mb RAM, and a nice chair.
- But they're nothing without the servers. And yet the servers don't have monitors (just vt320 terminals) or nice chairs. Why this injustice?!?

# File servers

- There are three servers on which you read your files:
  - `fullabull`: holds `/var/mail` — your mail
  - `maytag`: primary file server — stores just about everything, 1.6TB!!!
  - `godzilla`: (aka “god”) — some random other stuff. God is also the Gateway. It’s currently being phased out, eventually, it will be replaced by an array of servers. . . .
- We use NFS (Network File System) to access remote files.
- Under NFS, if at first you don’t succeed, Linux prints a message and tries again. And again, and again. . .
  - NFS server `maytag38-1` not responding  
still trying

- If bull goes down, you'll soon get errors showing up in your console window—at least, if you're reading your mail.

## Other Servers

There are other servers in the department; none of them as important as godzilla or maytag:

- mothra: A Solaris compute server with 4 processors. If you have anything you want to compile in Solaris use this baby, but don't use it to check your mail.
- skey2: The zephyrhost server. So you can send/receive zwrites.
- wilma: The web server. Wilma has essentially a copy of /pro/web/web. Use webupdate to update files, or wait until the next automatic update (done overnight).

## Shared Object (\*.so)

- Most programs use functions defined in the C and C++ standards (malloc, printf, . . . )
- The functions all have the same code, regardless of which program they're used in.
- *Share* the code, using **shared objects**.
- Also eases updating the code.
- Take cs167/9 for a more in-depth view into ld.so and other similar schemes.

## Things to do

- At run-time, the executable needs to find its shared objects.
- Usually, it'll be compiled with a run-path (see `ld -R`)
- But if you set `LD_LIBRARY_PATH`, it'll use that too.
- So when would I want to use this? Let's say you downloaded the latest release of quake 3, but the CS department's computers didn't have the latest libraries that quake 3 required. Well, you could download the latest libraries to your directory and specify where they are using `LD_LIBRARY_PATH`. Isn't that cool?

## Hacking some more

- Another thing ld.so does is check LD\_PRELOAD. If set, it loads the code in there *before* any libraries.
- For example, say we want to fool a program into thinking it's Jan. 1, 1970. Under Unix, most programs use time(2) to find the current time (in seconds since 1970). Compile this:

```
/pro/consult/Minicourses/adv_unix/examples/time.c:  
#include <sys/types.h>  
#include <time.h>  
time_t time(time_t *tloc) {  
    if(tloc) *tloc = 0;  
    return 0;  
}  
$ gcc -c time.c  
$ gcc -o time.so -shared time.o
```

- Next, set LD\_PRELOAD to include time.so.
- Finally, run a program that checks the time:  
0a /u/kit -22-> date  
Wed Dec 31 19:00:00 EST 1969

## Using Symbolic Links for Navigation

Let's say you're working on your Sceneview project for CS 123 and you find you keep needing to access files from `/course/cs123/lib/I/`, but you're working in your own directory `~/course/cs123/sceneview`. One solution is to make a symbolic link to the `lib/I` support directory in the one you're working in: Type `ln -s /course/cs123/lib/I/ I` in the current directory. Now you can enter that directory just by typing `cd I`.

You might notice that `/course/cs123/lib/I` is listed as `I@` in `/course/cs123/lib/`. The `@` means that it is a symbolic link itself. To find out what it is linked to, you can run `ls -l`. You find that `I -> I.Linux.2.0/`, meaning `I` is a symbolic link to `I.Linux.2.0/`.

## cd -, pushd, popd

- `cd -` is a very useful shorthand for changing to the directory that you were just in. You can use it to switch back and forth between two directories quickly. It's also useful in combination with the previous tip about using symbolic links.
- But `tcsh` can do more than just remember the previous directory: you can keep an arbitrary stack of directories.
- `pushd directory` does almost the same thing as `cd`, but it keeps the current directory below the new one on the directory stack.
- `popd` pops the current directory off the directory stack and changes into the previous one.
- `dirs` lets you examine the current directory stack.

# PATH

What really happens when you type a command into the shell, you ask? Well, your shell uses a magic environment variable called `PATH` to determine where the command to execute actually is. `PATH` is an ordered colon-separated list of directories that the shell should prepend to the command you type in order to try to execute it. For example, if `PATH=/bin:/usr/bin`, when you type `foo` into a shell, your shell secretly tries to run `/bin/foo`, then `/usr/bin/foo`, only giving up if it can't find a program to run.

## cdpath

- You might know how your shell locates programs to run using the environment variable `PATH`. `tcsh` has an analogous variable called `cdpath`.
- You can set your `cdpath` like this:

```
set cdpath = ( ~ /course/cs017 /pro/consult )
```

- When you try to `cd` into a directory that doesn't exist in the current directory, `tcsh` will try each of the directories in the `cdpath` in order.
- Now I can `cd course` from anywhere to go to my course directory quickly.

## using find

- `find` is an incredibly useful command: it searches recursively through directories for files that satisfy certain criteria.
- The simplest option is `-name`:

```
find ~ -name '*.class'
```

finds all the Java class files in your home directory. (The single quotes are necessary to prevent your shell from expanding the wildcard.)

- You can insert the output of any command in the command line of the current one with backquotes:

```
rm -f `find ~ -name '*.class'`
```

deletes all the Java class files in your home directory.

- `find` is also useful for checking permissions:

```
find /course/cs017 -perm -o+w
```

finds all world-writable files or directories contained in the starting directory.

## more backquotes

Command substitution (the fancy name for backquotes) can be very useful for many other things too. Take renaming files. Let's say you want to rename every \*.tex file to \*.tex.old. (Note: this example will use Bourne shell syntax. You'll need to run it in sh, ksh, bash, or zsh.)

```
for f in *.tex
do mv $f `echo $f | sed 's/\.tex$/\.tex\.old/'`
done
```