

# UNIX Power Tools

Intermediate Systems Administration DeCal

Lecture #5

George Wu

Slides prepared by Joshua Kwan

# Today

- Learn to use tools cut, sed, sort, tr, and grep to do amazing text manipulation
- Learn how to use regular expressions
- Learn how to use xargs to get over the limitations of command substitution
- Learn to properly use find

- # Sort(1)
- Easy stuff. Takes input, or file(s), and sorts it, ascending alphanumerically by default
  - Can sort by different criteria (see man page) or by columns or backwards

```
sort -k2 a.txt b.txt
```

```
ls | sort -r
```

- Often used in conjunction with `uniq(1)`:  

```
sort classes-taken.txt | uniq
```

because `uniq` needs a sorted input.  
(Shows unique lines in `classes-taken.txt`. `uniq -u` for non-unique

# tr(1)

- Used to **TR**anslate characters or classes of characters in an input stream, or delete them. Does not work with strings!!

```
tr 'a-z' 'A-Z' names.txt
```

- Try commands below at your own risk.

```
echo "shiftclock" | tr -d fl
```

```
echo "go bears" | tr a e
```

# cut(1)

- Splits lines into fields with the delimiter of your choice

```
echo "a,b,c" | cut -d, -f1
```

(returns 1)

```
echo "Jack eats pie" | cut -d' ' -f3
```

(returns pie)

```
echo "Jack eats pie" | cut -d, -f1
```

(returns Jack eats pie, since there are no commas)

# Joke time

Q: How do Unix sysadmins have sex?

A: *unzip ; strip ; touch ; grep ; finger ; mount ; fsck ;  
more ; yes ; yes ; yes ; umount ; sleep*

# sed(1)

- Stands for **S**tream **E**Ditor; takes input and spits it back out with certain modifications

```
sed 's/D/A+/g' < grades.txt
```

(Changes all “D” to “A+” in grades.txt on all lines and spits it to stdout.)

```
sed 's/John/Jeff/' < roster.txt
```

(Changes “John” to “Jeff” once per line in roster.txt.)

```
sed 's/\([^ ]+\) your \([^ ]+\)  
+\/\2\1er/g' < insults.txt
```

(Changes e.g. “fail your test” to “testfailer” in insults.txt.)

# Regular Expressions

- Regular expressions can be used with grep and sed (next slide!)
- A superset of the wildcard system you learned before (?/\*)
- It's best to teach by example, so...



# Regular Expressions

- Find all lines that contain  
“what/What”  
*[wW]hat*
- Find all lines that start with “x” and  
end with a number or a lowercase  
letter followed by any character  
*^x.\*[0-9a-z].\**
- Find all lines that have no  
whitespace:  
*^\S+\$*

# Regular

# Expressions

- You can use these expressions in sed(1) for substitution:  
s/regex1/regex2/
- You can use these expressions in egrep(1) for matching: egrep  
“regex1” < file
- This has just been a really brief overview, but they’re **super powerful**.
- See <http://www.ternent.com/tech/regex/>

# xargs(1)

- Trying “rm \*” in a huge directory or “rm \$(<deleteme.txt)” with a huge file will give “command list too long”!
- Instead: `xargs rm < deleteme.txt` or `find . | xargs rm -f` (achtung!)
- If your files have names with spaces?  
`find . -print0 | xargs -0 rm -f`

# find(1) power user!

- The find command can do way more than just find all the files in a directory. It has predicates!

*find -iname "TeSt.TxT" -and -type f*

Finds **files** called test.txt with case insensitivity

*find -not -name "meh" -or -type d*

Finds directories... or anything not named meh (case sensitive.)

- Consult the manpage for more predicate goodness.