How To Look Like A UNIX Guru

Terence Parr

Last updated: September 2, 2003

UNIX is an extremely popular platform for deploying server software partly because of its security and stability, but also because it has a rich set of command line and scripting tools. Programmers use these tools for manipulating the file system, processing log files, and generally automating as much as possible.

If you want to be a serious server developer, you will need to have a certain facility with a number of UNIX tools; about 15. You will start to see similarities among them, particularly regular expressions, and soon you will feel very comfortable. Combining the simple commands, you can build very powerful tools very quickly--much faster than you could build the equivalent functionality in C or Java, for example.

This lecture takes you through the basic commands and then shows you how to combine them in simple patterns or idioms to provide sophisticated functionality like histogramming. This lecture assumes you know what a shell is and that you have some basic familiarity with UNIX.

Everything is a stream

The first thing you need to know is that UNIX is based upon the idea of a *stream*. Everything is a *stream*, or appears to be. Device drivers look like streams, terminals look like streams, processes communicate via streams, etc... The input and output of a program are streams that you can redirect into a device, a file, or another program.

Here is an example device, the null device, that lets you throw output away. For example, you might want to run a program but ignore the output.

```
$ ls > /dev/null # ignore output of ls
```

where "`# ignore output of ls`" is a comment.

Most of the commands covered in this lecture process `stdin` and send results to `stdout`. In this manner, you can incrementally process a data stream by hooking the output of one tool to the input of another via a *pipe*. For example, the following piped sequence prints the number of files in the current directory modified in August.

```
$ ls -l | grep Aug | wc -l
```

Imagine how long it would take you to write the equivalent C or Java program. You can become an extremely productive UNIX programmer if you learn to combine the simple command-line tools. Even when programming on a PC, I use MKS's UNIX shell and command library to make it look like a UNIX box. Worth the cash.

Getting help

If you need to know about a command, ask for the "man" page. For example, to find out about the `ls` command, type

```
$ man ls
LS(1)                   System General Commands Manual
LS(1)

NAME
     ls - list directory contents

SYNOPSIS
     ls [-ACFLRSTWacdfgiklnoqrstux1] [file ...]

DESCRIPTION
     For each operand that names a file of a type other than directory,
ls
...
```

You will get a summary of the command and any arguments.

If you cannot remember the command's name, try using `apropos` which finds commands and library routines related to that word. For example, to find out how to do checksums, type

```
$ apropos checksum
cksum(1), sum(1)        - display file checksums and block counts
md5(1)                  - calculate a message-digest fingerprint
(checksum) for a file
```

The basics

There are 4 useful ways to display the contents or portions of a file. The first is the very commonly used command `cat`. For example, to display your bash initialization file, type:

```
$ cat ~parrt/.bash_profile
```

where `~parrt` is user parrt's home directory. If a file is really big, you will probably want to use `more`, which spits the file out in screen-size chunks.

```
$ more /var/log/mail.log
```

If you only want to see the first few lines of a file or the last few lines use `head` and `tail`.

```
$ head /var/log/mail.log
$ tail /var/log/mail.log
```

You can specify a number as an argument to get a specific number of lines:

```
$ head -30 /var/log/mail.log
```

The most useful incantation of `tail` prints the last few lines of a file and then waits, printing new lines as they are appended to the file. This is great for watching a log file:

```
$ tail -f /var/log/mail.log
```

If you need to know how many characters, words, or lines are in a file, use `wc`:

```
$ wc /var/log/mail.log
     164    2916   37896 /var/log/mail.log
```

Where the numbers are, in order, lines, words, then characters. For clarity, you can use `wc -l` to print just the number of lines.

Tarballs

Note: *The name comes from a similar word, hairball (stuff that cats throw up), I'm pretty sure.*

To collect a bunch of files and directories together, use `tar`. For example, to tar up your entire home directory and put the tarball into `/tmp`, do this

```
$ cd ~parrt
$ cd .. # go one dir above dir you want to tar
$ tar cvf /tmp/parrt.backup.tar parrt
```

By convention, use `.tar` as the extension. To untar this file use

```
$ cd /tmp
$ tar xvf parrt.backup.tar
```

`tar` untars things in the **current** directory!

After running the untar, you will find a new directory, `/tmp/parrt`, that is a copy of your home directory. Note that the way you tar things up dictates the directory structure when untarred. The fact that I mentioned `parrt` in the tar creation means that I'll have that dir when untarred. In contrast, the following will also make a copy of my home directory, but without having a `parrt` root dir:

```
$ cd ~parrt
$ tar cvf /tmp/parrt.backup.tar *
```

It is a good idea to tar things up with a root directory so that when you untar you don't generate a million files in the current directly. To see what's in a tarball, use

```
$ tar tvf /tmp/parrt.backup.tar
```

Most of the time you can save space by using the `z` argument. The tarball will then be `gzip`'d and you should use file extension `.tar.gz`:

```
$ cd ~parrt
$ cd .. # go one dir above dir you want to tar
$ tar cvfz /tmp/parrt.backup.tar.gz parrt
```

Unzipping requires the `z` argument also:

```
$ cd /tmp
$ tar xvfz parrt.backup.tar.gz
```

If you have a big file to compress, use `gzip`:

```
$ gzip bigfile
```

After execution, your file will have been renamed `bigfile.gz`. To uncompress, use

```
$ gzip -d bigfile.gz
```

To display a text file that is currently `gzip`'d, use `zcat`:

```
$ zcat bigfile.gz
```

Searching streams

One of the most useful tools available on UNIX and the one you may use the most is `grep`. This tool matches regular expressions (which includes simple words) and prints matching lines to `stdout`.

The simplest incantation looks for a particular character sequence in a set of files. Here is an example that looks for any reference to `System` in the java files in the current directory.

```
grep System *.java
```

You may find the dot '.' regular expression useful. It matches any single character but is typically combined with the star, which matches zero or more of the preceding item. Be careful to enclose the expression in single quotes so the command-line expansion doesn't modify the argument. The following example, looks for references to any a forum page in a server log file:

```
$ grep '/forum/.*' /home/public/cs601/unix/access.log
```

or equivalently:

```
$ cat /home/public/cs601/unix/access.log | grep '/forum/.*'
```

The second form is useful when you want to process a collection of files as a single stream as in:

```
cat /home/public/cs601/unix/access*.log | grep '/forum/.*'
```

If you need to look for a string at the beginning of a line, use caret '^':

```
$ grep '^195.77.105.200' /home/public/cs601/unix/access*.log
```

This finds all lines in all access logs that begin with IP address 195.77.105.200.

If you would like to invert the pattern matching to find lines that do not match a pattern, use `-v`. Here is an example that finds references to non image `GET`s in a log file:

```
$ cat /home/public/cs601/unix/access.log | grep -v '/images'
```

Now imagine that you have an http log file and you would like to filter out page requests made by nonhuman spiders. If you have a file called `spider.IPs`, you can find all nonspider page views via:

```
$ cat /home/public/cs601/unix/access.log | grep -v -f /tmp/spider.IPs
```

Finally, to ignore the case of the input stream, use `-i`.

Translating streams

Morphing a text stream is a fundamental UNIX operation. PERL is a good tool for this, but since I don't like PERL I stick with three tools: `tr`, `sed`, and `awk`. PERL and these tools are line-by-line tools in that they operate well only on patterns fully contained within a single line. If you need to process more complicated patterns like XML or you need to parse a programming language, use a context-free grammar tool like **ANTLR**.

tr

For manipulating whitespace, you will find `tr` very useful.

If you have columns of data separated by spaces and you would like the columns to collapse so there is a single column of data, tell `tr` to replace space with newline `tr ' ' '\n'`. Consider input file `/home/public/cs601/unix/names`:

```
jim scott mike
```

```
bill randy tom
```

To get all those names in a column, use

```
$ cat /home/public/cs601/unix/names | tr ' ' '\n'
```

If you would like to collapse all sequences of spaces into one single space, use `tr -s ' '`.

To convert a PC file to UNIX, you have to get rid of the '\r' characters. Use `tr -d '\r'`.

sed

If dropping or translating single characters is not enough, you can use `sed` (stream editor) to replace or delete text chunks matched by regular expressions. For example, to delete all references to word `scott` in the names file from above, use

```
$ cat /home/public/cs601/unix/names | sed 's/scott//'
```

which substitutes `scott` for nothing. If there are multiple references to `scott` on a single line, use the `g` suffix to indicate "global" on that line otherwise only the first occurrence will be removed:

```
$ ... | sed 's/scott//g'
```

If you would like to replace references to `view.jsp` with `index.jsp`, use

```
$ ... | sed 's/view.jsp/index.jsp/'
```

If you want any `.asp` file converted to `.jsp`, you must match the file name with a regular expression and refer to it via `\1`:

```
$ ... | sed 's/\(.*\).asp/\1.jsp/'
```

The `\(...\)` grouping collects text that you can refer to with `\1`.

If you want to kill everything from the ',' character to end of line, use the end-of-line marker `$`:

```
$ ... | sed 's/,.*$//' # kill from comma to end of line
```

awk

When you need to work with columns of data or execute a little bit of code for each line matching a pattern, use `awk`. `awk` programs are pattern-action pairs. While some `awk` programs are complicated enough to require a separate file containing the program, you can do some amazing things using an argument on the command-line.

`awk` thinks input lines are broken up into fields (i.e., columns) separate by whitespace. Fields are referenced in an action via `$1`, `$2`, ... while `$0` refers to the entire input line.

A pattern-action pair looks like:

```
pattern {action}
```

If you omit the pattern, the action is executed for each input line. Omitting the action means print the line. You can separate the pairs by newline or semicolon.

Consider input

```
aasghar Asghar, Ali
wchen   Chen, Wei
zchen   Chen, Zhen-Jian
```

If you want a list of login names, ask `awk` to print the first column:

```
$ cat /home/public/cs601/unix/emails.txt | awk '{print $1;}'
```

If you want to convert the login names to email addresses, use the `printf` C-lookalike function:

```
$ cat /home/public/cs601/unix/emails.txt | awk
'{printf("%s@cs.usfca.edu,",$1);}'
```

Because of the missing `\n` in the `printf` string, you'll see the output all on one line ready for pasting into a mail program:

```
aasghar@cs.usfca.edu,wchen@cs.usfca.edu,zchen@cs.usfca.edu
```

You might also want to reorder columns of data. To print firstname, lastname, you might try:

```
$ cat /home/public/cs601/unix/emails.txt | awk '{printf("%s %s\n", $3,
$2);}'
```

but you'll notice that the comma is still there as it is part of the column:

```
Ali Asghar,
Wei Chen,
Zhen-Jian Chen,
```

You need to pipe the output thru `tr` (or `sed`) to strip the comma:

```
$ cat /home/public/cs601/unix/emails.txt | \
  awk '{printf("%s %s\n", $3, $2);}' | \
  tr -d ','
```

Then you will see:

```
Ali Asghar
Wei Chen
Zhen-Jian Chen
```

You can also use `awk` to examine the value of content. To sum up the first column of the following data (in file `/home/public/cs601/unix/coffee`):

```
3 parrt
2 jcoker
8 tombu
```

use the following simple command:

```
$ awk '{n+=$1;} ; END {print n;}' < /home/public/cs601/unix/coffee
```

where `END` is a special pattern that means "after processing the stream."

If you want to filter or sum all values less than or equal to, say 3, use an `if` statement:

```
$ awk '{if ($1<=3) n+=$1;} END {print n;}' <
/home/public/cs601/unix/coffee
```

In this case, you will see output `5` (3+2);

Using `awk` to grab a particular column is very common when processing log files. Consider a **http://www.jguru.com** page view log file, `/home/public/cs601/unix/pageview-20021022.log`, that are of the form:

```
date-stamp(thread-name): userID-or-IPaddr URL site-section
```

So, the data looks like this:

```
20021022_00.00.04(tcpConnection-80-3019):       203.6.152.30
/faq/subtopic.jsp?topicID=472&page=2    FAQs
20021022_00.00.07(tcpConnection-80-2981):       995134  /index.jsp
Home
20021022_00.00.08(tcpConnection-80-2901):       66.67.34.44
/faq/subtopic.jsp?topicID=364   FAQs
20021022_00.00.12(tcpConnection-80-3003):       217.65.96.13
/faq/view.jsp?EID=736437        FAQs
20021022_00.00.13(tcpConnection-80-3019):       203.124.210.98
/faq/topicindex.jsp?topic=JSP   FAQs/JSP
20021022_00.00.15(tcpConnection-80-2988):       202.56.231.154
/faq/index.jsp FAQs
20021022_00.00.19(tcpConnection-80-2976):       66.67.34.44
/faq/view.jsp?EID=225150        FAQs
220021022_00.00.21(tcpConnection-80-2974):       143.89.192.5
/forums/most_active.jsp?topic=EJB       Forums/EJB
```

```
20021022_00.00.21(tcpConnection-80-2996):       193.108.239.34
/guru/edit_account.jsp  Guru
20021022_00.00.21(tcpConnection-80-2996):       193.108.239.34
/misc/login.jsp Misc
...
```

When a user is logged in, the log file has their user ID rather than their IP address.

Here is how you get a list of URLs that people view on say October 22, 2002:

```
$ awk '{print $3;}' < /home/public/cs601/unix/pageview-20021022.log
/faq/subtopic.jsp?topicID=472&page=2
/index.jsp
/faq/subtopic.jsp?topicID=364
/faq/view.jsp?EID=736437
/faq/topicindex.jsp?topic=JSP
/faq/index.jsp
/faq/view.jsp?EID=225150
/forums/most_active.jsp?topic=EJB
/guru/edit_account.jsp
/misc/login.jsp
...
```

If you want to count how many page views there were that day that were not processing pages (my processing pages are all of the form `process_`*xxx*), pipe the results through `grep` and `wc`:

```
$ awk '{print $3;}' < /home/public/cs601/unix/pageview-20021022.log | \
  grep -v process | \
  wc -l
67850
```

If you want a unique list of URLs, you can sort the output and then use `uniq`:

```
$ awk '{print $3;}' < /home/public/cs601/unix/pageview-20021022.log | \
  sort | \
  uniq
```

`uniq` just collapses all repeated lines into a single line--that is why you must sort the output first. You'll get output like:

```
/article/index.jsp
/article/index.jsp?page=1
/article/index.jsp?page=10
/article/index.jsp?page=2
...
```

Moving files between machines

rsync

When you need to have a directory on one machine mirrored on another machine, use `rsync`. It compares all the files in a directory subtree and copies over any that have changed to the mirrored directory on the other machine. For example, here is how you could "pull" all logs files from `livebox.jguru.com` to the box from which you execute the `rsync` command:

```
$ hostname
jazz.jguru.com
$ rsync -rabz -e ssh -v 'parrt@livebox.jguru.com:/var/log/jguru/*' \
  /backup/web/logs
```

`rsync` will delete or truncate files to ensure the files stay the same. This is bad if you erase a file by mistake--it will wipe out your backup file. Add an argument called `--suffix` to tell `rsync` to make a copy of any existing file before it overwrites it:

```
$ hostname
jazz.jguru.com
$ rsync -rabz -e ssh -v --suffix .rsync_`date '+%Y%m%d'` \
 'parrt@livebox.jguru.com:/var/log/jguru/*' /backup/web/logs
```

where ``` `date '+%Y%m%d'` ``` (in reverse single quotes) means "execute this `date` command".

To exclude certain patterns from the sync, use `--exclude`:

```
$ rsync -rabz --exclude=entitymanager/ --suffix .rsync_`date '+%Y%m%d'`
\
  -e ssh -v 'parrt@livebox.jguru.com:/var/log/jguru/*' /backup/web/logs
```

scp

To copy a file or directory manually, use `scp`:

```
$ scp lecture.html parrt@nexus.cs.usfca.edu:~parrt/lectures
```

Just like `cp`, use `-r` to copy a directory recursively.

Miscellaneous

find

Most GUIs for Linux or PCs have a search facility, but from the command-line you can use `find`. To find all files named `.p4` starting in directory `~/antlr/depot/projects`, use:

```
$ find  ~/antlr/depot/projects -name '.p4'
```

The default "action" is to `-print`.

You can specify a regular expression to match. For example, to look under your home directory for any xml files, use:

```
$ find ~ -name '*.xml' -print
```

Note the use of the single quotes to prevent command-line expansion--you want the '*' to go to the `find` command.

You can execute a command for every file or directory found that matches a name. For example, do delete all xml files, do this:

```
$ find ~ -name '*.xml' -exec rm {} \;
```

where "{}" stands for "current file that matches". The end of the command must be terminated with ';' but because of the command-line expansion, you'll need to escape the ';'.

You can also specify time information in your query. Here is a shell script that uses `find` to delete all files older than 14 days.

```
#!/bin/sh

BACKUP_DIR=/var/data/backup

# number of days to keep backups
AGE=14 # days
AGE_MINS=$[ $AGE * 60 * 24 ]

# delete dirs/files
find $BACKUP_DIR/* -cmin +$AGE_MINS -type d -exec rm -rf {} \;
```

fuser

If you want to know who is using a port such as HTTP (80), use `fuser`. You must be root to use this:

```
$ sudo /sbin/fuser -n tcp 80
80/tcp:              13476 13477 13478 13479 13480
13481 13482 13483 13484 13486 13487 13489 13490 13491
13492 13493 13495 13496 13497 13498 13499 13500 13501 13608
```

The output indicates the list of processes associated with that port.

whereis

Sometimes you want to use a command but it's not in your `PATH` and you can't remember where it is. Use `whereis` to look in standard unix locations for the command.

```
$ whereis fuser
fuser: /sbin/fuser /usr/man/man1/fuser.1 /usr/man/man1/fuser.1.gz
$ whereis ls
ls: /bin/ls /usr/man/man1/ls.1 /usr/man/man1/ls.1.gz
```

`whereis` also shows `man` pages.

which

Sometimes you might be executing the wrong version of a command and you want to know which version of the command your `PATH` indicates should be run. Use `which` to ask:

```
$ which ls
alias ls='ls --color=tty'
        /bin/ls
$ which java
/usr/local/java/bin/java
```

If nothing is found in your path, you'll see:

```
$ which fuser
/usr/bin/which: no fuser in
(/usr/local/bin:/usr/local/java/bin:/usr/local/bin:/bin:/usr/bin:/usr/X
11R6/bin:/usr/X11R6/bin:/home/parrt/bin)
```

kill

To send a signal to a process, use `kill`. Typically you'll want to just say `kill pid` where `pid` can be found from `ps` or `top` (see below).

Use `kill -9 pid` when you can't get the process to die; this means kill it with "extreme prejudice".

traceroute

If you are having trouble getting to a site, use `traceroute` to watch the sequence of hops used to get to a site:

```
$ /usr/sbin/traceroute www.cnn.com
 1  65.219.20.145 (65.219.20.145)  2.348 ms  1.87 ms  1.814 ms
 2  loopback0.gw5.sfo4.alter.net (137.39.11.23)  3.667 ms  3.741 ms
3.695 ms
 3  160.atm3-0.xr1.sfo4.alter.net (152.63.51.190)  3.855 ms  3.825 ms
3.993 ms
...
```

What is my IP address?

```
$ /sbin/ifconfig
```

Under the `eth0` interface, you'll see the `inet addr`:

```
eth0      Link encap:Ethernet  HWaddr 00:10:DC:58:B1:F0
          inet addr:138.202.170.4  Bcast:138.202.170.255
Mask:255.255.255.0
          ...
```

pushd, popd

Instead of `cd` you can use `pushd` to save the current dir and then automatically `cd` to the specified directory. For example,

```
$ pwd
/Users/parrt
$ pushd /tmp
/tmp ~
$ pwd
/tmp
$ popd
~
$ pwd
/Users/parrt
```

top

To watch a dynamic display of the processes on your box in action, use `top`.

ps

To print out (wide display) all processes running on a box, use `ps auxwww`.

Useful combinations

How to kill a set of processes

If you want to kill all `java` processes running for `parrt`, you can either run `killall java` if you are `parrt` or generate a "kill" script via:

```
$ ps auxwww|grep java|grep parrt|awk '{print "kill -9 ",$2;}' >
/tmp/killparrt
$ bash /tmp/killparrt # run resulting script
```

The `/tmp/killparrt` file would look something like:

```
kill -9 1021
kill -9 1023
kill -9 1024
```

Note: you can also do this common task with:

```
$ killall java
```

How to make a histogram

A histogram is set of count, value pairs indicating how often the value occurs. The basic operation will be to sort, then count how many values occur in a row and then reverse sort so that the value with the highest count is at the top of the report.

```
$ ... | sort |uniq -c|sort -r -n
```

Note that `sort` sorts on the whole line, but the first column is obviously significant just as the first letter in someone's last name significantly positions their name in a sorted list.

`uniq -c` collapses all repeated sequences of values but prints the number of occurrences in front of the value. Recall the previous sorting:

```
$ awk '{print $3;}' < /home/public/cs601/unix/pageview-20021022.log | \
  sort | \
  uniq
/article/index.jsp
/article/index.jsp?page=1
/article/index.jsp?page=10
/article/index.jsp?page=2
...
```

Now add `-c` to `uniq`:

```
$ awk '{print $3;}' < /home/public/cs601/unix/pageview-20021022.log | \
  sort | \
  uniq -c
 623 /article/index.jsp
   6 /article/index.jsp?page=1
  10 /article/index.jsp?page=10
 109 /article/index.jsp?page=2
...
```

Now all you have to do is reverse sort the lines according to the first column numerically.

```
$ awk '{print $3;}' < /home/public/cs601/unix/pageview-20021022.log | \
  sort | \
  uniq -c | \
  sort -r -n
6170 /index.jsp
2916 /search/results.jsp
1397 /faq/index.jsp
1018 /forums/index.jsp
 884 /faq/home.jsp?topic=Tomcat
...
```

In practice, you might want to get a histogram that has been "despidered" and only has faq related views. You can filter out all page view lines associated with spider IPs and filter in only faq lines:

```
$ grep -v -f /tmp/spider.IPs /home/public/cs601/unix/pageview-
20021022.log | \
  awk '{print $3;}'| \
  grep '/faq' | \
  sort | \
  uniq -c | \
  sort -r -n
1397 /faq/index.jsp
 884 /faq/home.jsp?topic=Tomcat
 525 /faq/home.jsp?topic=Struts
 501 /faq/home.jsp?topic=JSP
 423 /faq/home.jsp?topic=EJB
...
```

If you want to only see despidered faq pages that were referenced more than 500 times, add an `awk` command to the end.

```
$ grep -v -f /tmp/spider.IPs /home/public/cs601/unix/pageview-
20021022.log | \
  awk '{print $3;}'| \
  grep '/faq' | \
  sort | \
  uniq -c | \
  sort -r -n | \
  awk '{if ($1>500) print $0;}'
1397 /faq/index.jsp
 884 /faq/home.jsp?topic=Tomcat
 525 /faq/home.jsp?topic=Struts
 501 /faq/home.jsp?topic=JSP
```

Generating scripts and programs

I like to automate as much as possible. Sometimes that means writing a program that generates another program or script.

Processing mail files

I wanted to get a sequence of SQL commands that would update our database whenever someone's email bounced. Processing the mail file is pretty easy since you can look for the error code followed by the email address. A bounced email looks like:

```
From MAILER-DAEMON@localhost.localdomain  Wed Jan  9 17:32:33 2002
Return-Path: <>
Received: from web.jguru.com (web.jguru.com [64.49.216.133])
        by localhost.localdomain (8.9.3/8.9.3) with ESMTP id RAA18767
        for <notifications@jguru.com>; Wed, 9 Jan 2002 17:32:32 -0800
Received: from localhost (localhost)
        by web.jguru.com (8.11.6/8.11.6) id g0A1W2o02285;
        Wed, 9 Jan 2002 17:32:02 -0800
Date: Wed, 9 Jan 2002 17:32:02 -0800
From: Mail Delivery Subsystem <MAILER-DAEMON@web.jguru.com>
Message-Id: <200201100132.g0A1W2o02285@web.jguru.com>
To: <notifications@jguru.com>
MIME-Version: 1.0
Content-Type: multipart/report; report-type=delivery-status;
        boundary="g0A1W2o02285.1010626322/web.jguru.com"
Subject: Returned mail: see transcript for details
Auto-Submitted: auto-generated (failure)

This is a MIME-encapsulated message

--g0A1W2o02285.1010626322/web.jguru.com

The original message was received at Wed, 9 Jan 2002 17:32:02 -0800
from localhost [127.0.0.1]

   ----- The following addresses had permanent fatal errors -----
<pain@intheneck.com>
    (reason: 550 Host unknown)

   ----- Transcript of session follows -----
550 5.1.2 <pain@intheneck.com>... Host unknown (Name server:
intheneck.com: host not found)
...
```

Notice the SMTP 550 error message. Look for that at the start of a line then kill the angle brackets, remove the `...` and use `awk` to print out the SQL:

```
# This script works on one email or a file full of other emails
# since it just looks for the SMTP 550 or 554 results and then
# converts them to SQL commands.
grep -E '^(550|554)' | \
        sed 's/[<>]//g' | \
        sed 's/\.\.\.//' | \
        awk "{printf(\"UPDATE PERSON SET bounce=1 WHERE
email='%s';\n\",\$3);}" >> bounces.sql
```

I have to escape the `$2` because it means something to the surround bash shell script and I want `awk` to see the dollar sign.

Generating getter/setters

```
#!/bin/bash
# From a type and name (plus firstlettercap version),
# generate a Java getter and setter
#
# Example: getter.setter String name Name
#

TYPE=$1
NAME=$2
UPPER_NAME=$3
```

```
echo "public $TYPE get$UPPER_NAME() {"
echo "  return $NAME;"
echo "}"
echo
echo "void set$UPPER_NAME($TYPE $NAME) {"
echo "  this.$NAME = $NAME;"
echo "}"
echo
```