Linux command line

UNIX shell is an interactive domain specific language used to control and monitor the UNIX operating system, which includes processes, devices, ram, cpus, disks etc. It is also a programming language, though we'll use it mostly to do scripting: lists of commands. If you have to use a Windows machine, the shell is useless so try to install a UNIX shell.

You need to get comfortable on the UNIX command line because many companies use Linux on their servers, which in my opinion, is best used from the command line for ultimate control over the server or cluster. For example, in the next lab we will launch Hadoop jobs from the shell. Facility with the show marks you as a more sophisticated programmer.

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.

The basics

UNIX disk structure: http://www.thegeekstuff.com/2010/09/linux-file-system-structure/

`~parrt` is my home directory, `/home/parrt`, as is ~.

```
$ls /
Applications
Library
Network
System
Users
Volumes
bin
```

```
cores
dev
etc
home
mach_kernel
net
opt
private
sbin
tmp
usr
var
```

Like when were typing in the Python shell, each command is terminated by newline. The first thing we type is the command followed by parameters (separated by whitespace):

```
$ foo arg1 arg2 ... argN
```

That is why whitespace in filenames sucks:

```
$ ls house\ of\ pancakes
```

But we can use this:

```
$ ls 'house of pancakes'
```

The commands can be built into the shell or they can be programs that we write and invoke. For example, here's how you ask which program is being executed when you type a command:

```
$which ls python
/bin/ls
/usr/local/bin/python
```

The Python interpreter is a program installed on our disk and when we say `python` at the shell, it finds the program using an ordered list of directories in `PATH` environment variable and executes it.

Next, we pass information around using streams and we can shunt that data into a file or pull data from a file using special operators. You can pretend these are like operators in a programming language like addition and multiplication. Each program has standard input, standard output, and standard error; three streams.

We can set the standard input of a process using > character:

```
$ls / > /tmp/foo
```

Here is how to type something directly into a text file:

```
cat > /tmp/foo
the first line of the file
the second line of the file
^D
$
```

The `^D` means control-D, which means end of file. `cat` is reading from standard input and writing to the file. The way it knows we are done is when we signal in the file with control-D.

We can set the standard input of a process to the contents of a file and redirect the output of a process to a file.

```
$wc < /tmp/foo
      19      19     118
```

or

```
$wc /tmp/foo
      19      19     118 /tmp/foo
```

We can connect to the output of one program to the input of another using pipes: `|`.

```
$ls / | wc # count files are in the root directory
      19      19     118
```

Here is a simple pipe (show first 5 lines of the text we stored in foo):

```
$cat /tmp/foo | head -5
Applications
Library
Network
System
Users
```

So, some programs take filenames on the command line and some expect standard input. For example, the `tr` translation command expects standard input and writes to standard output

```
$ls / | tr -d e # delete all 'e' char from output
Applications
Library
Ntwork
Systm
Usrs
Volums
bin
cors
dv
tc
hom
mach_krnl
nt
opt
privat
sbin
tmp
usr
var
```

Misc

man, help, apropos

ls, cd, pushd, popd

cp, scp

cat, more

head, tail

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
```

wc

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 GETs 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.

Basics of file processing

cut, paste

```
$cat ../data/coffee
3 parrt
2 jcoker
8 tombu
```

cut grabs one or more fields according to a delimiter like strip in Python. It's also like SQL `select f1, f2 from file`.

```
$cut -d ' ' -f 1 ../data/coffee > /tmp/1
cut -d ' ' -f 2 ../data/coffee > /tmp/2
```

```
$cat /tmp/1
3
2
8
```

```
$cat /tmp/2
```

```
parrt
jcoker
tombu
```

paste combines files as if they were columns:

```
$paste /tmp/1 /tmp/2
3       parrt
2       jcoker
8       tombu

$paste -d ', ' /tmp/1 /tmp/2
3,parrt
2,jcoker
8,tombu
```

Get first and third column from names file

```
cut -d ' ' -f 1,3 names
```

`join` is like an INNER JOIN in SQL. (zip() in python) first column must be sorted.

```
$cat ../data/phones
parrt 5707
tombu 5001
jcoker 5099

$cat ../data/salary
parrt 50$
tombu  51$
jcoker 99$

$join ../data/phones ../data/salary
parrt 5707 50$
tombu 5001 51$
jcoker 5099 99$
```

Here is how I email around all the coupons for Amazon Web services without having to do it manually:

```
$ paste students aws-coupons
jim@usfca.edu X
kay@usfca.edu Y
sriram@usfca.edu Z
...
```

and here is a little Python script to process those lines:

```
import os
import sys
for line in sys.stdin.readlines():
    p = line.split('\t')
    p = (p[0].strip(), p[1].strip())
    print "echo '' | mail -s 'AWS coupon "+p[1]+"' "+p[0]
    os.system("echo '' | mail -s 'AWS coupon "+p[1]+"' "+p[0])
```

and here's how you run it:

```
$ paste students aws-coupons | python email_coupons.py
```

Processing log files

```
cut -d ' ' -f 1 access.log | sort | uniq -c | sort -r -n|head
```

get unique list of IPs. find out who is hitting your site by getting histogram. how many hits to the images directory? how many total hits to the website? histogram of URLs.

Python programs

```
$python resources/printargs.py hi mom
args: hi mom
```

That Python code:

```
import sys
print "args:", sys.argv[1], sys.argv[2]
```

We can use those arguments as filenames to open or we can read from standard input:

```
import sys
print sys.stdin.readlines()
```

Print coffee data out

```
$python resources/mycat.py < ../data/coffee
['3 parrt\n', '2 jcoker\n', '8 tombu\n']
```