

Unix Tutorial

By Joshua Lande

SASS

January 21

This is not a philosophy talk!

- "Doug McIlroy, the inventor of Unix pipes and one of the founders of the Unix tradition, summarized the philosophy as follows:
 - ***This is the Unix philosophy: Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface.***
- (http://en.wikipedia.org/wiki/Unix_philosophy)

The Basics

- All command line programs have 3 main components:
 - **Command line arguments**
 - **Standard Input (stdin)**
 - **Standard Output (stdout)**
- By default, stdin is typed from the terminal and stdout is printed to the terminal
- for help on any command:

```
$ man command
```

A few basic programs

- **echo** – sends the command line arguments to stdout
- **cat** – reads file(s) as command line arguments and sends the lines to stdout. If no files specified, sends stdin to stdout.
- **tac** – Just like cat but backwards
- **tee** – writes the input both to the stdout and to a file specified as a command line argument

Example

```
$ sed 's/lame/awesome/g'  
This example is lame  
This example is awesome  
^D
```

- sed replaces the first word with the second word
- 's/lame/awesome/g' is a command line argument
- First line is the stdin (I typed)
- Second line is the stdout (printed to screen)
- When you are done sending stuff to stdin, type CTRL-D and the program will finish up.

Sorting

```
$ sort -t ":" -n -k2
```

```
Ted:1000
```

```
John:1
```

```
Sally:100
```

```
Bob:10
```

```
John:1
```

```
Bob:10
```

```
Sally:100
```

```
Ted:1000
```

- Sort is a program to sort the lines of standard input
- t specifies the field separator
- n means numeric sort
- k2 means sort the second column

input/output redirection

- Change where stdin comes from and stdout goes.
- End your line with `>` to redirect stdout to a file.

```
$ cat > file.txt
```

```
Some random stuff...
```

```
^D
```

- Use `>>` to append to a file
- Use `<` to read stdin from a file.

```
$ cat < file.txt  
Some random stuff...
```

pipes

- Turn the stdout of one program to the stdin of another using a pipe |

```
$ cat *.txt | sort | uniq > output.txt
```

- In this example, cat outputs all text files, which are sorted. All duplicates are then removed and the output is saved to a file.

```
$ somecommand | tee output.txt
```

- Prints output of a command to stdout and a file!

```
$ somecommand | less
```

- Pipe to less for nice navigation.

awk

- Powerful programming language
- Easy to whip up powerful scripts
- The general syntax is an expression followed by a command.
- loops over stdin
- **Example:** second column if the first column is a number greater than 10

```
$ awk '$1>10{print $2}' file.txt
```

awk (more?)

- Put code you want to run before or after inside BEGIN and END blocks.
- Example:** count number of occurrences in file:

```
$ awk 'BEGIN {print "Analysis:" }  
      /foo/{++foobar }  
      END {print "foo appears  
              " foobar " times." }' file
```

awk (again?)

- Divides each line into columns
- **default** separator is spaces
- Specify the separator between each column:

```
BEGIN {FS=":"}
```

- Set output column separator as semicolons:

```
BEGIN {OFS=";"}
```

awk

```
$ ls -l
```

```
drwxr-xr-x 3 lande gl 2048 Dec 12 19:21 bin  
drwx----- 2 lande gl 4096 Nov 20 15:59 mail  
...
```

- Print only files from Dec 12

```
$ ls -l | awk '$6=="Dec"&&$7=="12" {print $0}'
```

- Sum total memory

```
$ ls -l | awk '{s+=$5} END{print s}'
```

(last awk script)

- Replace all columns with their absolute value:

```
$ awk '{ for (i = 1; i <= NF; i++)  
        if ($i < 0) $i = -$i; print $0}'
```

- So many one liners
- <http://www.catonmat.net/blog/awk-one-liners-explained-part-one/>
- <http://www.catonmat.net/blog/awk-one-liners-explained-part-two/>
- <http://www.catonmat.net/blog/awk-one-liners-explained-part-three/>

Job Control

- Control-z suspends a currently running job
- The jobs command shows you all the jobs running in the terminal

```
$ jobs
```

```
[1]-  Stopped
```

```
yes
```

```
[2]+  Stopped
```

```
yes
```

- Each job given a number. Run the second job in the background or foreground:

```
$ bg 2
```

```
$ fg 2
```

Job Control

- Begin job in the background

```
$ command &
```

- List all jobs running on your machine:

```
$ ps -u lande
```

PID	TTY	TIME	CMD
19231	pts/21	00:00:00	vim
19233	pts/21	00:00:00	find

- Kill any job (by PID or name)

```
$ kill 19231
```

```
$ killall find
```

find (stuff quickly)

- Syntax: find path expression
- Searches recursively through all subfolders

```
$ find /path/ -name "file.txt"
```

- **-iname** for case insensitive search
- **-type f** finds only files and **-type d** only folders
- Example: find files ending with either 'sh' or 'pl':

```
$ find . -type f \( -iname "*.sh" -or \  
                                     -iname "*.pl" \)
```

- Use a \ to continue a long line

grep (is beautiful)

- Search through stdin for things
- Sends to stdout lines matched lines

```
$ grep tacos
this line has tacos
this line has tacos
this line dosen't
more tacos
more tacos
```

- You can do the same in awk with

```
$ awk '/tacos/{print $0}'
```

grep

- `$ grep -B2` -B prints lines before match
- `$ grep -A4` -A prints lines after each match
- `$ grep -C3` -C prints the lines before and after
- `-i` case insensitive search
- `-v` prints lines with no match
- `-c` prints just number of matches
- `--color` highlights matches

grep

- Fancy regular expressions: -E
- **Example:** Match IP range from 172.22.21.1 to 172.22.21.35:

```
$ grep -E '172\.22\.21\.([1-9]|(1[0-9]|2[0-9]|3[0-5]))' hosts.txt
```

- <http://unstableme.blogspot.com/2008/07/match-ip-range-using-egrep-bash.html>

xargs

- Makes stdin as a command line argument
- useful for running a command a bunch of times
- **Example:** Search in all files for a variable name

```
$ find . -name "*.cxx" | xargs grep var
```

- This is equivalent to running grep on all *.cxx files in all subdirectories.

```
$ grep *.cxx
```

- The above would only search files in current directory

xargs (is xtreme)

- Use `-I{}` to replace all occurrences of `{}` in the command with the standard input.
- **Example** (I use all the time): Run all the scripts in all subdirectories

```
$ find . -name "*.sh" | xargs -I{} sh {}
```

- Copy lots of files at once

```
$ find . -name '*.dat' | xargs -I{} cp  
{}/folder/
```

Too many jobs running?

- Kill all jobs running in terminal

```
jobs -p | xargs -i kill -9 {}
```

- jobs -p prints all job IDs.
- kill -9 kills the job with that ID.

xargs (to the rescue)

- **Example:** run cvs update in all subfolders:

```
find . -type d | xargs -i -t sh -c \  
                'cd {};cvs update'
```

- -t prints out the command before executing (for debugging)

<http://en.wikipedia.org/wiki/xargs>

par

- Reformats text
- Not installed by default but easy to build.

```
$ par 30j
```

```
We the people of  
the United States, in order to form a  
more perfect  
union, establish justice...
```

```
We the people of the United  
States, in order to form a  
more perfect union, establish  
justice...
```

par (cont)

- par can fix your code comments

```
$ par 25
# one fish #
# two fish #
# red #
# fish blue fish #

# one fish two fish #
# red fish blue fish #
```

<http://www.nicemice.net/par/>

paste

```
$ cat f1.txt
```

```
a
```

```
b
```

```
c
```

```
$ cat f2.txt
```

```
1
```

```
2
```

```
$ paste f1.txt f2.txt
```

```
a 1
```

```
b 2
```

```
c
```

<http://unstableme.blogspot.com/2009/01/linux-paste-command-good-examples-uses.html>

Various stuff

- Go to previous folder:

```
$ cd -
```

- Get the previous command:

```
$ file.txt
```

```
bash: file.txt: command not found
```

```
$ echo !!
```

- **!\$** is the last part of the last command

Lazy history

\$!comma

- Runs previous command beginning with comma

\$!comma :p

- Prints the command to the screen

Fun stuff

- Creates all subdirectories.

```
$ mkdir -p /home/make/all/of/these/dirs/
```

```
$ cp /path/filename1 /path/filename2
```

- Instead:

```
$ cp /path/filename{1,2}
```

```
$ mkdir foo1 foo2 foo3 bar1 bar2 bar3
```

```
$ mkdir {foo, bar}{1, 2, 3}
```

Guess who?

- Who is on your machine
- send them a message

```
$ who
```

```
lande          pts/16          Jan 21 01:34  
(rescomp-07-119188.stanford.edu)
```

```
...
```

```
$ write lande
```

```
What's up?
```

Questions