Working on the Chain Gang

Linux prefers small, single-purpose functions and utilities. Chain them together with the "pipe", which sends the output of one command into the next as input.

```
$ grep pattern input.txt | sort | uniq -c
```

Iteratively build a series of commands to create output that definitively addresses your requirements.

Redirect Output: I Don't Want to Hear You

Redirect output to a file instead of the shell itself with the "greater than" character. (Warning: Overwrites any existing contents!)

```
$ grep pat1 input.txt > results.txt
```

Append to existing files with "double greater than".

```
$ grep pat2 input.txt >> results.txt
```

Gimme That (File)!

chown Change file's owner (optionally its group)

```
$ chown username[:groupname] file.txt
```

`-R`	Recursively change ownership of directory contents

chmod Change file permissions

```
$ chmod <permissions> file.txt
```

`-R`	Recursively change permissions of directory contents

Permissions:

`rwx`	`r` = read; `w` = write; `x` = execute (files), traverse (directories)

Assign with octal values (`r` = 4, `w` = 2, `x` = 1) or human-readable-ish user/group/other values:

`755`	rwx for user, rx for group, rx for others
`400`	r for user, none for group or others
`664`	rw for user, rw for group, r for others
`u=rwx`	Assign read, write, and execute perms to file owner
`g+rx`	Add read and execute perms for owning group
`a-w`	Remove write perms from everyone

Slicing and Dicing

cut Remove sections from each input line

```
$ cut -d ',' -f 2,5,7 input.txt
```

`-d`	Specify an alternate delimiter (default is TAB)
`-f`	Field numbers to display

Bring Out the Big Guns

sed Stream editor to filter and transform text. **Seriously powerful stuff™!**

```
$ sed 's/<regex>/<replacement>/' input.txt
```

awk Pattern scanning and processing language **Even more seriously powerful stuff™!**

```
$ awk -F ',' '{ print $1,$6,$3 }' in.txt
```

`-F`	Specify input field separator (default is space)

Input and output field separators can be specified in the awk script itself with the `FS` and `OFS` variables:

```
$ awk '{ FS = ","; OFS = "\t"; \
  print $2,$4 }' in.txt
```

To Learn More About These and Other Fine Linux Commands...

Use the built-in reference manual:

man Interface to the on-line reference manuals

```
$ man find
```

`-k`	Perform keyword search through all man pages

Use inline command help where available – many commands provide brief usage statements with the "`--help`" or "`-h`" options

```
$ tcpdump --help
```

Learn Command Line Kung-Fu, with Hal Pomeranz, Tim Medin, Ed Skoudis, and Paul Asadoorian:

`http://blog.commandlinekungfu.com/`

The Grymoire - home for UNIX wizards:

`http://www.grymoire.com/Unix/`

SANS DFIR

DIGITAL FORENSICS & INCIDENT RESPONSE

Linux Shell Survival Guide v1.1

POCKET REFERENCE GUIDE

SANS Institute
http://computer-forensics.sans.org

by Phil Hagen
http:/lewestech.com

Purpose

This guide is a supplement to SANS FOR572: Advanced Network Forensics and Analysis. It covers some of what we consider the more useful Linux shell primitives and core utilities. These can be exceedingly helpful when automating analysis processes, generating output that can be copied and pasted into a report or spreadsheet document, or supporting quick-turn responses when a full tool kit is not available.

Remember: *If you can make it happen in a shell over SSH on bad dialup, there is a better chance of being the lethal forensicator when it really matters!*

How To Use This Document

Linux has been around since 1991, and its *NIX parents since 1969. This handout cannot begin to scratch the surface of the great and powerful things you can do with nothing more than a shell prompt and some moxie. Use this document as a "memory jog" for some of the capabilities of the more commonly used tools in this course and in the forensic workflow in general.

Dig into the details of each tool's features through its manual pages (aka "man pages") and other online and offline references. We think you will find the shell to be as powerful as the GUI, and in some cases a far superior alternative – especially for scalability and automation.

Use the Force

BASH provides many functions to improve your accuracy, speed, and efficiency – know and use them!

Tab Completion

Hit the <TAB> key to expand the first few characters of a command, directory name, filename, or variable name. If there is more than one possible option, it will complete as far as possible. Press <TAB> again to see the possible completion options.

Standard Variables

~	*An alias for the current user's home directory (also `$HOME`)*
`$PATH`	*The command search path*
`$?`	*The exit value of the previous command*
`$PWD`	*The current working directory*

Command History

Cycle through previous commands by pressing the up and down arrows. Modify previous commands with the left and right arrows. Use the `history` command to see a list of the command history buffer. (BASH writes this buffer is `~/.bash_history` upon exiting, overwriting any existing contents.) Press Ctrl-R to search through history for commands that match a search string.

Searching: Where For Art Thou?

`grep` Print lines matching a pattern

```
$ grep pattern input.txt
```

`-i`	Case-insensitive pattern matching
`-v`	Print lines that do not match
`-c`	Count matching lines instead of printing them
`-l`	Print filenames containing matching lines
`-h`	Do not include filenames when searching multiple input files (e.g. `output*.txt`)

Order in the Court

`sort` Sort lines alphabetically or numerically

```
$ sort input.txt
```

`-n`	Sort numerically (5 before 10)
`-r`	Reverse sort order
`-k`	Specify an alternate sort field
`-t`	Specify a field delimiter for `-k` (default

De-Duplication and De-Duplication

`uniq` Only print consecutive matching lines once

```
$ grep pattern input.txt | uniq
```

`-c`	Print the number of consecutive lines

Remember: Only finds consecutive matching lines! Most useful with input piped from the `sort` command.

```
$ grep pattern input.txt | sort | uniq
```

The Replacement Characters

`tr` Translate (replace) or delete characters

```
$ grep foo input.txt | tr '\t' ','
```

`-d`	Delete specified character, rather than replace it with another (only takes one argument)

Go With The (Net)Flow

`nfdump` Process NetFlow data from files on disk

```
$ nfdump -R ./ -b -O tstart -o extended
```

`-R`	Recursively read data from the specified directory
`-b`	Aggregate records bidirectionally
`-a`	Aggregate by protocol, src/dst IPs, src/dst ports
`-A`	Specify custom aggregation
`-t`	Time window, in "YYYY/MM/DD.hh:mm:ss" format (See man page for additional details)
`-s`	Generate "TopN" statistics
`-O`	Specify output ordering
`-o`	Specify output format

No Packets, No Party

`tcpdump` Dump network traffic

```
$ sudo tcpdump -i eth0 '<BPF filter>'
```

`-D`	Enumerate network interfaces (useful on Windows)
`-w`	Write packet data to a file
`-s`	Number of bytes per packet to capture
`-i`	Specify the network interface on which to capture
`-r`	Read from existing pcap file instead of the network
`-C`	Number of bytes to save in a capture file before starting a new file
`-G`	Number of seconds to save in a capture file before starting a new file
`-W`	Used with the `-C` or `-G` options, limit the number of rotated files (i.e. create a "ring buffer")
`-n`	Prevent DNS lookups on IP addresses. Use twice to also prevent port-to-service lookups
`-x`	Display packet contents in hex

Note that tcpdump requires root privileges to capture network traffic promiscuously. User-level permissions are sufficient for manipulating existing capture files.

See the `pcap-filter` man page for information on building BPFs to control captured traffic.

GUI-less Packet Spelunking

`tshark` Dump and analyse network traffic (aka "Wireshark in the shell")

```
$ tshark -n -r in.pcap -Y '<disp filter>'
```

`-n`	Prevent DNS and port lookups
`-r`	Read from a pcap file instead of the network
`-w`	Write output to a pcap file instead of the terminal
`-T`	Output format (text, fields, etc)
`-e`	With "`-T fields`", add a field to the output
`-Y`	Protocol-aware display filter to apply
`-z`	Statistical output modes – see man page for many different options!

See the `wireshark-filter` man page for information on building protocol-aware display filters.