LINUX

Data manipulation/formatting:

```bash
1: ls -al | head -n 5 2: ls -al | tail -n 5 3: ls -al | awk '{print
$8, $9}' 4: ls -al | awk '{print $9, $8}' 5: ls -al | awk '{printf
"%-20.20s %s\n",$9, $8}' 6: ls -al | grep bash 7: ls -al > ls.out 8:
cat ls.out 9: cat ls.out | sed 's/bash/I replace this!!!/g' 10: ls -
al | tr -s ' ' | cut -d ' ' -f 8,9 11: wc filename # prints the
number of lines, words and bytes in the given input. # -l, -w, -c
tail -n 5 file1 tail -f /var/log/messages # follow file head -n 5
file1
```

Bash

while `cut` understands only single symbol (space in our case) as a way to tell field apart (field separator), `awk` treats any number of spaces and tabs as filed separator, so there is no need to use `tr` which removes unnecessary spaces.

`sed` is one powerful stream editor

- for any debugging: `-v -vvv`

join:

```bash
#Join command combines lines from two files based on a common field.
# like DB join join file1.txt file2.txt
```

Bash

translate:

```
# change to uppercase tr cmd tr a-z A-Z < employee.txt
```
Bash

xargs:

```
#xargs sort a| uniq | xargs -I{} sh -c 'grep {} a | wc -l; echo {}'
k get pods |grep proctor| grep Err | awk '{print $1}'| xargs -I{}
kubectl delete po {} docker ps | grep app | awk '{ print $1 }' |
xargs -I{} docker kill {}
```
Bash

sort:

```
#sort sort file1 sort -r file1 #desc sort -t: -k 2 file1 #file
delimited by `:`, sort by col 2 sort -t: -u -k 2 file1 #file
delimited by `:`, sort by col 2, no duplicates # sort by ip addr sort
-t . -k 1,1n -k 2,2n -k 3,3n -k 4,4n /etc/hosts #127.0.0.1
localhost.localdomain localhost ls -al | sort +4n #sort by file size
ls -al | sort +4nr
```
Bash

uniq:

```
#uniq #used in combination with sort command, #as uniq removes
duplicates only from a sorted file sort file1 | uniq #or sort -u
file1 sort namesd.txt | uniq -c #display uniq row cnt #2 Alex
Jason:200:Sales sort namesd.txt | uniq -cd #display only duplicate
row
```
Bash

cut:

```
cut -d: -f 1,3 file1 #file delimited by `:`, display field 1 and 3
cut -c 1-8 file1 #display first 8 chars of every line #Displays the
total memory available on the system. free | tr -s ' ' | sed
'/^Mem/!d' | cut -d" " -f2
```

Bash

grep:

```
grep [options] pattern [files] grep John /etc/passwd # display all
matching lines grep -v John /etc/passwd # display all unmatching
lines -c == count lines -i == ignore case -r == recursive, search in
dir -l == show only file name
```

Bash

find:

```
find [pathnames] [conditions] find /etc -name "*mail*" #find files
containing a specific word in its name find / -type f -size +100M
#find files greater than certain size find . -mtime +60 #find files
not modified in the last x number of days find . -mtime -2 #find
files modified in the last x number of days #delete archive files
with extension *.tar.gz and greater than 100MB find / -type f -name
*.tar.gz -size +100M -exec ls -l {} \; #list find / -type f -name
*.tar.gz -size +100M -exec rm -f {} \; #rm #archive all the files
that are not modified in the last x number of days find /home/jsmith
-type f -mtime +60 | xargs tar -cvf /tmp/`date
'+%d%m%Y'_archive.tar` -name == file name -mtime == modified time
```

Bash

```
1: echo Hello, $LOGNAME! 2: echo 'echo Hello, $LOGNAME!' >> .profile
3: cp -v .bash_history{,1} 4: ls .*
```

Bash

local & global env vars for terminals:

```
1: foo='Hello World!' #local to terminal 3: set | grep foo 4: env |
grep foo 5: export foo #global 6: env | grep foo
```

Bash

random numbers:

```bash
echo $RANDOM
```

Bash

Redirection:

```bash
1: sudo aptitude install pv 2: read foo < /dev/tty 3: Hello World!
4: echo $foo > foo.out 5: cat foo.out 6: echo $foo >> foo.out 7: cat
foo.out 8: echo > foo.out 9: cat foo.out 10: ls -al | grep foo 11:
ls -al | tee ls.out 12: dd if=/dev/zero of=~/test.img bs=1M count=10
13: pv ~/test.img | dd if=/dev/stdin of=/dev/null bs=1 14: crontab -
l | tee crontab-backup.txt | sed 's/old/new/' | crontab - 15: ls |
tee file1 file2 file3 16: ls | tee -a file # append to file
```

Bash

pv(pipe viewer) shows you a progress of reading the file

tee:

Tee command is used to store and view (both at the same time) the output of any other command.

Tee command writes to the STDOUT, and to a file at a time

```bash
$ ls | tee file
```

Bash

fg & bg jobs:

```bash
1: less -S .profile 2: <CTRL+z> 3: less -S .bashrc 4: <CTRL+z> 5:
jobs 6: fg 7: q 8: fg 9: q 10: jobs # list background jobs
```

Bash

exit code:

```bash
1: ls 2: echo $?
```

Bash

searching documentation:

```
1: man -k . 2: man -k [search string] 3: man -wK [search string]
```

Bash

arch linux

Runlevels:

```
S: This executed then system is powered on. 0: Halt, this defines
which actions are executed when system is shutting down. 1: Single-
User mode, this is a special mode for troubleshooting. In this mode
most daemons are not automatically started. 2-5: Full Multi-User,
configured daemon start in this mode. 6: Reboot. Like halt, but
instead of powering down system reboots.
```

Markdown

rcconf package which allows you to easily manage runlevels:

```
1: sudo aptitude install rcconf 2: ls -al /etc/rc2.d 3: sudo rcconf
--list 4: sudo update-rc.d exim4 disable 5: ls -al /etc/rc2.d 6: sudo
rcconf --list 7: sudo update-rc.d exim4 enable 8: ls -al /etc/rc2.d
9: sudo rcconf --list
```

Bash

Daemon — a program which runs in background all the time. This means that it does not care if you are logged into the system, and usually you do not need to start it manually, because daemons are started automatically when computer boots up.

Signals:

```
0: Exit HUP: Hangup INT: Interrupt QUIT: Quit ILL: Illegal TRAP:
Trace/breakpoint ABRT: Abort FPE: Floating KILL: Non-catchable TERM:
Terminate (can be ignored by a process, or process can save its state
before it stops) SEGV: Invalid mem PIPE: Broken ALRM: Timer
```

Markdown

Process:

```bash
#process status ps auxe --forest #USER PID %CPU %MEM VSZ RSS TTY STAT
START TIME COMMAND ps -ef | sort # a == all users # x == include
processes which do not have a controlling terminal # u == udisplay
the processes belonging to the specified usernames. ps U <user>
#processes owned by a user ps U $USER #processes owned by current
user pmap -x 3401 # displays the memory map of a process # Address
Kbytes RSS Anon Locked Mode Mapping 1: dd if=/dev/zero of=~/test.img
bs=1 count=$((1024*1024*1024)) & 2: kill -s USR1 $! # Queries dd for
status. 3: <ENTER> #status 4: kill -s TERM $! 5: <ENTER>
```
Bash

Cron:

```markdown
# min hr day-of-month month-of-year day-of-week # 0-59 0-23 1-31 1-12
0-6(0==sunday) `*` == all `/` eg: `30-40/3` means run program on 30-
th minute and every 3 minutes thereafter until 40-th minute. `,` ==
list of values eg: `30,40` in minutes field means 30th and 40th
minute `-` == range of values eg: `30-40` in minutes field means
every minute between 30th and 40th minute. `L` == last value
```
Markdown

```bash
crontab -l # print out current crontab. crontab -e # edit crontab
for current user. crontab -r # remove crontab for current user. at #
executes commands at a specified time. atq # lists pending jobs. atrm
# removes a job.
```
Bash

logrotate daemon

```bash
tail -f /var/log/auth.log find /var/log -mmin -10 # find any files
which are modified in last 10 minutes. grep -irl rcconf . last #
Prints out information about last logins of users. lastlog # Prints
out information about the most recent logins of all users.
```
Bash

filesystem, mounting, mountpoint:
fdisk to create and modify partitions on disk

fstab - file system table

```
mount # /dev/vda5 on / type ext3 (rw,errors=remount-ro) cat
/etc/fstab # <file system> <mount point> <type> <options> <dump>
<pass> # UUID=128559db-a2e0-4983-91ad-d4f43f27da49 / ext3
errors=remount-ro 0 1 mount -a # mount all filesystem described in
/etc/fstab. mount /dev/sda<N> /<mount point> # mount a partition.
umount /tmp # unmount a filesystem # if device is used by some
process then umount can proceed. fuser -mu /mnt/ # to check which
process & user is currently accessing the filesystem # command used
to identify processes using the files / directories umount -f /tmp #
force unmount umount /tmp # lazy unmount fsck /tmp # check partition
for errors. blkid # print out unique partition identifiers. UUID
fdisk /dev/sda # p d n p w resize2fs /dev/sda1 fdisk -l # list all
disks and their partitions mkfs -t ext4 /dev/sdb # creates a new
filesystem on /dev/sdb mkdir /var/lib/postgresql blkid #(copy UUID)
vim /etc/fstab. #(UUID="b1473cac-a9dc-4adf-af1c-3cb1c21cc94e"
/var/lib/postgresql ext4 defaults 0 2) mount -a df -h . #list
devices/partitions df #disk usage tune2fs -l /dev/sda8 # print out
and change file system parameters.
```

Bash

filesystem stats/status/properties:

```
stat file1 #statistics stat -f / #filesystem status
```

Bash

tar:

```
tar -czvf root.tgz /opt/root/ # c == compress # z == use gzip to
compress the resultant archive # f == read/write archive from/to file
(root.tgz) # v == verbose # j == use bzip2 to compress the resultant
archive tar -tzvf root.tgz # t == list archive contest tar -xzvf
~/root.tgz # x == extract to disk from archive
```

Bash

zip:

```
zip compressed-file-name.zip /var/log/* zip -r compressed-file-
dir.zip /var/log/ zip -p <pwd> compressed-file-name.zip /var/log/*
#password protected zip zip -e compressed-file-name.zip /var/log/*
Enter password: unzip compressed-file-name.zip unzip -l var-log.zip
#list zipped file content
```

Bash

File permission classes: `user`, `group`, `others`

permissions: `r,w,x`

`owner` (r,w,x) `group` (r,w,x) `others` (r,w,x)

```
chown chmod umask chmod 777 file.txt chmod ugo+rwx file.txt chmod
u+x file.txt chgrp <grp> /home/john/sample.txt # change group
ownership of a file. chgrp -R <grp> dir/ chgrp --reference=file2
file1
```

Bash

Networking:

```
1. ifconfig 2. ip, ip route show # addr, Hwaddr(MAC addr), Bcast,
Mask, UP, Running, errors, dropped, # packets, RX and TX bytes 1.
netstat -ap -ntpl -nr 2. ss # Proto, Recv-Q, Send-Q, Local Address,
Foreign Address, State, PID/Program name route -n # destination,
gateway, Iface(Interface to which packets for this route will be
sent.) nc -l 80 #set up nc to listen on port 80 echo 'Hello, world!'
| nc localhost 80
```

Bash

File transfer:

```
netcat -l -p <port> > file.pdf #sender m/c netcat $SENDER_IP_ADDRESS
<port> < file.pdf #receiver m/c
```

Bash

DNS configuration:

```
scutil --dns
```

Bash

Secure shell:

```
ssh # client program which allows you to connect to ssh server.
#Putty is such client program for example. sshd # the server program
which allows you to connect to it with ssh. /etc/ssh/ssh_config #
default client program configuration file. /etc/ssh/sshd_config #
default server program configuration file. scp # file transfer over
ssh. sftp # ftp-like protocol for managing remote files. sshfs #
remote filesystem mounting over ssh. ssh tunnelling # a method to
transfer almost any data over secure connection.
```

Bash

Toggle ssh session:

```
#remotehost ~^Z # escape char ~ + (ctrl+Z) #localhost jobs fg %1
```

Bash

ssh statistics:

```
#remotehost, only works on SSH2 client. ~s `remote host: remotehost
local host: localhost remote version: SSH-1.99-OpenSSH_3.9p1 local
version: SSH-2.0-3.2.9.1 SSH Secure Shell (non-commercial) compressed
bytes in: 1506 uncompressed bytes in: 1622 compressed bytes out: 4997
uncompressed bytes out: 5118 packets in: 15 packets out: 24 rekeys: 0
Algorithms: Chosen key exchange algorithm: diffie-hellman-group1-sha1
Chosen host key algorithm: ssh-dss Common host key algorithms: ssh-
dss,ssh-rsa Algorithms client to server: Cipher: aes128-cbc MAC:
hmac-sha1 Compression: zlib Algorithms server to client: Cipher:
aes128-cbc MAC: hmac-sha1 Compression: zlib`
```

Bash

logging:

```
dmesg # print or control the kernel ring buffer /var/log/dmseg # log
file which contains copy of dmesg messages # during system boot only,
without timestamps. /var/log/kern.log # log file which contains copy
of all dmesg messages, # including timestamps. # Timestamps start
ticking after rsyslog logging daemon is started # Contains
/var/log/dmesg. /var/log/messages # log file which logs all non-debug
```

```
and non-critical messages. # Contains /var/log/dmesg. /var/log/syslog
# log file which logs everything, except auth related messages. #
Contains /var/log/dmesg, /var/log/messages and /var/log/kern.log
/var/log/auth
```

Bash

suppress stdout and stderr:

```
./shell-script.sh > /dev/null # supress err and stdout ./shell-
script.sh 2> /dev/null # supress err 30 1 * * * command > /dev/null
2>&1 # supress err and stdout
```

Bash

performance:

1. uptime
2. free
3. vmstat
4. top
5. iostat
6. iotop
7. perf

```
uptime # how long the system has been running. # 03:13:58 up 4 days,
22:45, 1 user, load average: 0.00, 0.00, 0.00 free # display amount
of free and used memory in the system. # Mem: total used free shared
buffers cached vmstat -S M # information about processes, memory,
paging, block IO, traps, # disks and cpu activity. # procs --------
memory------- ---swap-- -----io---- -system-- ----cpu---- vmstat -d #
disk- ----------reads-------- ---------writes-------- -----IO------
top # display Linux tasks in real-time. # Uptime, Tasks, CPU,
Mem/swap, process iostat # cpu and I/O # avg-cpu: %user %nice %system
%iowait %steal %idle # Device: tps Blk_read/s Blk_wrtn/s Blk_read
Blk_wrtn iotop # I/O # Call-Graph profiling, trace system calls,
trace/count kernel events perf record <slow_cmd> perf record <pid>
perf record <pid> <cmd> perf record -a #profile all processes cat
perf.data perf report # know how much CPU every fn is using perf top
perf list # lists events: # system calls # sending n/w packets #
reading from block device(disk) # context switches, page faults #
perf trace safe to run in production unlike strace which is slow perf
```

```
trace perf stat # CPU counters -> h/w counters # L1 cache hits/misses
# instr/cycle # page faults # CPU cycles # TLB misses # branch
prediction misses
```

Bash

lsof - ls open files:

list all the open files in the system

open files → network connection, devices and directories

```
look <prefix> whereis ls # find out where the binary, source, and
man page files # for a command is located. whatis <cmd> # displays a
single line information about a command finger <user> # used to
lookup information about an user # login, username, home directory,
shell # split file into multiple files split -l 1500 testfile
<prefix> # split file into 1500 lines files split --bytes=50M
logdata <prefix> # split file into 50MB files stat <file> touch
<file> # used to change the timestamps(access, modify and change) of
a file.
```

Bash

Http/Https

```
check_http -H 192.162.1.50 -p 8080 # check status of remote HTTP
server check_http -H 192.164.1.50 -S -p 8443 # check status of remote
HTTPS server check_http -H 192.163.1.50 -u <url> # check specific
url check_http -H 192.165.1.50 -C 365 # check ssl certificate expiry
of the website # -C <days> check_ping check_ftp
```

Bash

file checksum:

```
# cksum computes a cyclic redundancy check (CRC) checksum for files #
and counts the number of bytes of a file. cksum sample.txt
#3704359389 91 sample.txt cksum file1.txt file2.txt # Check whether
two files are identical. #3704359389 91 file1.txt #3704359389 91
file2.txt # md5sum is a 128 bit checksum which will be unique for the
same data provided. md5sum sample.txt
#3b85ec9ab2984b91070128be6aae25eb samplefile.txt
```

Bash

diff b/w files:

```bash
diff file1 file2 diff -w file1 file2 #ignore whitespace
```

Bash

cd:

```bash
export CDPATH=.:~:/etc:/var
```

Bash

users' connect time:

```bash
ac -d #per day ac -p #per user ac -d user1 #for specific user per
day
```

Bash

Systemcalls:

http://asm.sourceforge.net/syscall.html

kernel module:

https://www.thegeekstuff.com/2013/07/write-linux-kernel-module/

```bash
lsmod # lists loaded kernel modules
```

Bash

resolving dns name:

```bash
dig dns_name/IP host dns_name/IP dig +trace dns_name/IP
```

Bash

/etc/nsswitch.conf → contains order in which lookups are performed (files and

dns)
/etc/host

/etc/services → service-port mapping

processes listening on specific port:

```bash
lsof -i :<port>
```

Bash

Test availability of remote service:

ping

netcat nc/ncat:

```bash
nc -l <port> #starts a server listening at port [listener mode] nc
<ip> <port> #opens a connection with remote server #nc to open
connection b/w two servers #Making any process a server nc -l -p 1234
-e /bin/sh
```

Bash

nmap:

```bash
nmap -n <network> # scan network - available nodes, IPs, listening
ports, nmap -n 192.12.4.0/24 # mac address nmap <host> nmap
192.13.22.1 # u == udp
```

Bash

memory leak → program keeps on claiming memory w/o returning it

```bash
top -o <key> #sort by key
```

Bash

/proc/meminfo

tcpdump:

```bash
tcpdump port 8997 -w service.pcap # packet capture # -w -> file
tcpdump -n(numerical addr and port) -A(ascii text o/p) tcpdump -A # print out packets # what dns queries is my m/c sending tcpdump -i any port 53 # server is running on port 1337, are any packets arriving at that port tcpdump -i any port 1337 # what packets are coming into my server from ip 1.2.3.4 tcpdump port 1337 and host 1.2.3.4 # -i -> -i any -> n/w interface # -w -> -w my_packets.pcap -> write to file # -c -> count of packets to capture (limit) # -A -> prints out the packet's content, only works for http not https -> `ngrep` # -e -> shows ethernet info. like MAC addr # -p -> packets that are to or from your m/c
```
Bash

tcpdump o/p:

1. src+dest ip addr and port
2. timestamp
3. which tcp flags → for spotting beginning of a tcp connection
    1. `.` → ack
    2. `S` → syn
    3. `R` → RST
4. dns query → for dns packet

tcpdump uses BPF language to filter packets

BPF filters:

1. port, host → src
2. and, or, not

Performance:

1. RAM/memory
    1. top
    2. free -m
    3. /proc/meminfo
    4. swap, free mem, cache
    5. memory leak → top
2. I/O → fs, disk
    1. fs
        1. block size, fragmentation journaling

        2. tune2fs
    2. iotop
3. networking
4. cpu
    1. top

Systems troubleshooting:

1. glances
2. htop/top
3. ps aux | grep mysqld
4. lsof -a -p 28290 (open files)
5. pidof mysqld
6. Free
7. logs
8. journalctl
9. iostat
10. df -h -i
11. vmstat
12. dstat → prints how much n/w and disk your m/c used
13. lsblk
14. uptime
15. systemctl status
16. strace (to find which system calls are executed) -e -f -p
    1. -e → strace -e <system_call>
    2. -f → follow subprocesses
    3. -p → strace -p <pid>
    4. strace -e -f execve ./script.rb
    5. strace o/p → pid system_call(args)=fd
    6. -y → shows filename with fd
    7. -o → send o/p to file

8. spy on ssh → strace -f -o ssh_output.txt ssh <host>
9. which files are being opened → strace -e open -p <pid>

17. perf trace
18. perf
19. ltrace → trace library calls

Network troubleshooting:

1. iptraf → IP traffic
2. ping -c 3 <host> → are the computers connected
3. traceroute / tracepath -n <dns_name> / mtr
4. netstat -t(tcp) -u(udp) -l(listening sockets) -p(pid and program) -r(routetable) -i(n/w interface) -n(numerical addr and port)
5. ss -tuna (like netstat) → netstat/ss - am I using that port
6. lsof → what ports are being used
7. tcpdump -n(numerical addr and port) -A(ascii text o/p)
8. telnet
9. ssh
10. ngrep / tcpdump
11. nmap → is your n/w scanning your ports
12. dig / host, nslookup → does that domain exist
13. whois → lookup a domain
14. ifconfig
15. ip (successor to ifconfig) → configures interfaces, routes and more
16. arp → see your arp table
17. nc → make tcp connections manually, netcat/socat
18. curl
19. nftables/iptables → setup firewalls and NAT
20. sysctl → configure socket buffer sizes and more
21. ethtool → ethernet connections

Resources:

http://dbp-consulting.com/tutorials/debugging/linuxProgramStartup.html

https://linux.101hacks.com/toc/

learn linux the hard way

https://jvns.ca/

https://linux.101hacks.com/unix/nohup-command/

https://spin.atomicobject.com/2013/08/19/debug-ruby-processes/