Linux Reference Collection

A work-in progress, updated as subjects are
edited and formatted sufficiently for inclusion.
(this version released Feb 1 2023)

To add: SELinux(!), FirewallD, FreeIPA(!), Kubernetes/Openshift, Git, Docker, and more.

A few topics will be large enough to require stand-alone documentation (i.e., Ansible, OpenStack).

Tristan Mendoza
Austin, Texas USA
tristanm.tx@gmail.com

01. Managing the Kernel - Loadable Kernel Modules
02. Changing Parameters of Kernel Modules (Devices)
03. Tuning Resources with sysctl
04. More on /proc and /sys
05. /dev - Device Nodes
06. dmesg
07. Important Log File Locations and Using journalctl
08. Integrating and Configuring rsyslogd and journald
09. Basic Hardware Utilities
10. Process Management, Using PS to Inspect Processes
11. The sysstat Suite and sar (vmstat, iostat, etc)
12. Linux Performance Observability Tools as shown by Brendan Gregg
13. SysVinit and systemd Command Equivalents
14. systemd Command Overview
15. SysVinit - Directory Structures and inittab
16. Runlevel Service Management Tools -SysVinit initscript utilities
17. Example systemd Unit Script
18. Contents of the systemd Package
19. ifconfig, netstat, ip - net-tools and iproute2 equivalents
20. NetworkManager Service - Persistent Changes with nmcli
21. Getting Rid of Network Manager
22. systemd-networkd
23. Using iw and wpa_supplicant
24. WiFi Pentesting Tools
25. Netfilter/ iptables
26. Reviewing RAID Types (Prep for soon adding mdadm, NFS/iSCSI/NAS/SAN, etc stuff)

Managing the Kernel - Loadable Kernel Modules

Linux typically modules have the extension .ko ("kernel object") since version 2.6 (previous versions used the .o extension). Other OS's - kernel loadable module (kld) in FreeBSD; kernel extension (kext) in macOS; kernel extension module in AIX; kernel-mode driver in Windows NT. Linux modules are generally in subdirectories of **/lib/modules/** named for kernel version. Numbering format same as Linux versioning- major.minor.patch Odd numbers for minor version are developmental. /lib/modules/*.ko files i.e, bridge.ko for network support

After **/etc/initab** specifies the default runlevel, it kicks off **/etc/rc.sysinit** to load modules
/etc/modprobe.conf -- associates/aliases drivers to devices such as eth0
"alias eth0 natsemi" says use natsemi driver/module for eth0
/etc/modprobe.d/modprobe.conf.dist -- large set of standard autoloaded aliases
/etc/modprobe.d/blacklist and **blacklist-compat** -- aliases that are not loaded

modprobe <modulename> - loads/adds modules AND auto-adds their dependencies
-v verbose; -r remove; -a add; -l list all modules; -t [dir] list modules in directory
insmod - to insert 3rd party drivers/modules; common error: dependency error: "unresolved symbol…." inserts only specified module -not it's dependencies. **modprobe** is preferable
-e make persistent; -f force; -L prevent simultaneous loading of the same module; -o specify optional module name
modinfo <modulename> - param fields have variables and such used in the system calls. -V version, -n name -a author -d description -p parameters
depmod - is run at startup right before **modprobe** to give it dependency info- creates **/lib/modules/modules.dep**
lsmod - lists loaded modules. Indicates size, number of dependencies and what it is used by
modprobe -l - will do sort of the same, with dependencies
cat /proc/modules/ - all modules currently loaded- subdirectories for /pcmcia, /net, /arch, /fs, /drivers
/sys/module also has sub-directories that contains information about each kernel module installed

rmmod remove modules from the kernel but it does not check for dependencies
modprobe -r <modulename> - removes a module from the kernel after checking for dependencies

udev is a device manager that manages the automatic detection and kernel module loading for both coldplug and hotplug devices; in charge of the **/dev** virtual file system to dynamically creates device files as devices are added and removed. When providing new hardware like a USB key, udev wakes up, intitializes the new HW with kernel so kernel can load proper modules automatically.

udevadm monitor
Upon plugging in a usb stick, devices and bus messages (truncated below), see module listed for fat and vfat

```
[root@rhelserver ~]# udevadm monitor
monitor will print the received events for:
UDEV - the event which udev sends out after rule processing
KERNEL - the kernel uevent

KERNEL[69484.521158] add      /devices/pci0000:00/0000:00:11.0/0000:02:03.0/usb1/1-1 (usb)
KERNEL[69484.554385] add      /devices/pci0000:00/0000:00:11.0/0000:02:03.0/usb1/1-1/1-1:1.0 (usb)
UDEV  [69484.564011] add      /devices/pci0000:00/0000:00:11.0/0000:02:03.0/usb1/1-1 (usb)
KERNEL[69484.583125] add      /module/usb_storage (module)
UDEV  [69484.608070] add      /module/usb_storage (module)
KERNEL[69484.619712] add      /devices/pci0000:00/0000:00:11.0/0000:02:03.0/usb1/1-1/1-1:1.0/host3 (scsi)
UDEV  [69484.619737] add      /devices/pci0000:00/0000:00:11.0/0000:02:03.0/usb1/1-1/1-1:1.0 (usb)
KERNEL[69484.619745] add      /devices/pci0000:00/0000:00:11.0/0000:02:03.0/usb1/1-1/1-1:1.0/host3/scsi_host/host3 (scsi_host)
KERNEL[69484.619752] add      /bus/usb/drivers/usb-storage (drivers)
UDEV  [69484.621580] add      /bus/usb/drivers/usb-storage (drivers)
UDEV  [69484.622757] add      /devices/pci0000:00/0000:00:11.0/0000:02:03.0/usb1/1-1/1-1:1.0/host3 (scsi)
UDEV  [69484.623611] add      /devices/pci0000:00/0000:00:11.0/0000:02:03.0/usb1/1-1/1-1:1.0/host3/scsi_host/host3 (scsi_host)
KERNEL[69486.685029] add      /module/fat (module)
KERNEL[69486.685064] add      /kernel/slab/fat_cache (slab)
KERNEL[69486.685076] add      /kernel/slab/fat_inode_cache (slab)
UDEV  [69486.687754] add      /module/fat (module)
UDEV  [69486.687778] add      /kernel/slab/fat_cache (slab)
UDEV  [69486.687785] add      /kernel/slab/fat_inode_cache (slab)
KERNEL[69486.689217] add      /module/vfat (module)
UDEV  [69486.690176] add      /module/vfat (module)
```

Do it manually: unplug USB, then
lsmod | grep fat - shows vfat module still loaded. Don't rely on **udev** to unload what it activated
modprobe -r vfat --removes modules and dependencies no longer needed
modprobe vfat -- loads the module and dependencies (which udev is expected to accomplish automatically)
---
udevadm controls **systemd-udevd**, requests kernel events, manages event queue, and simple debugging.
udevadm [**info** | **trigger** | **settle** | **monitor**] <options> AND **udevadm control** <command>
The **/etc/udev/rules.d/** - directory allows naming devices when they are connected

Changing Parameters of Kernel Modules (Devices)

modinfo cdrom - reports params. These can be changed, but only by unloading the module with **modprobe -r cdrom**, then **modprobe cdrom lockdoor=0** (for example, to turn off the lockdoor boolean param)

```
[root@rhelserver ~]# lsmod | grep cdrom
cdrom                  42556  1 sr_mod
[root@rhelserver ~]# modinfo cdrom
filename:       /lib/modules/3.10.0-121.el7.x86_64/kernel/drivers/cdrom/cdrom.ko
license:        GPL
srcversion:     B5F2D59440347DFFB175E71
depends:
intree:         Y
vermagic:       3.10.0-121.el7.x86_64 SMP mod_unload modversions
signer:         Red Hat Enterprise Linux kernel signing key
sig_key:        42:49:68:9E:EF:C7:7E:95:88:0B:13:DF:E4:67:EB:1B:7A:91:D1:08
sig_hashalgo:   sha256
parm:           debug:bool
parm:           autoclose:bool
parm:           autoeject:bool
parm:           lockdoor:bool
parm:           check_media_type:bool
parm:           mrw_format_restart:bool
```

It used to be modifying modprobe.conf could make changes but in RHEL7 it changed
These are default settings for kernel modules, from the associated rpm packages:
/lib/modprobe.d/dist-alsa.conf and **/lib/modprobe.d/dist-blacklist.conf**
You don't want to edit them.

Instead, edit files in **/etc/modprobe.d/** By default it is empty- it is the place to put custom conf files.
(see **man 5 modprobe.d** - "options" section*). In this directory create/ **vim cdrom.conf** and add:
options cdrom lockdoor=0

Generally you need to restart to see the effects and truly reload- reloading the module isn't enough.
For some modules you can look in **/sys/module/**, find a directory for the module, and see a file called parameters (or something), but it is up to the programmers to provide this kind of file. For cdrom it isn't. Looking in **dmesg | grep <modulename>** might help find something about when the module was initialized.

```
[root@rhelserver module]# cd cdrom/
[root@rhelserver cdrom]# ls
coresize  holders  initsize  initstate  notes  refcnt  sections  srcversion  taint  uevent
[root@rhelserver cdrom]# dmesg | grep cdrom
[    2.738573] cdrom: Uniform CD-ROM driver Revision: 3.20
[root@rhelserver cdrom]# dmesg | grep -A5 cdrom
[    2.738573] cdrom: Uniform CD-ROM driver Revision: 3.20
[    2.739174] sr 1:0:0:0: Attached scsi CD-ROM sr0
[    2.777003] usb 2-2: New USB device found, idVendor=0e0f, idProduct=0002
[    2.777006] usb 2-2: New USB device strings: Mfr=0, Product=1, SerialNumber=0
[    2.777008] usb 2-2: Product: VMware Virtual USB Hub
[    2.782794] hub 2-2:1.0: USB hub found
```

*man page refers to locations **/etc/modprobe.d/*.conf**, **/lib/modprobe.d/*.conf**, and **/run/modprobe.d/*.conf**

Fields in /etc/modprobe.d/modprobe.conf (or **/etc/modprobe.conf**)

alias {wildcard} {module name}	Specify an alternate name for a module with a long name.
include {file name}	Add configuration files to a module.
options {module name} {option}	Options to be added to each module before insertion into the kernel.
install {module name} {command}	Run the command specified without inserting the module into the kernel.

Don't confuse with shared library files! (aren't kernel modules)

The **/usr/lib** and **/lib** directories are the default system library file locations where the system libraries are kept. Contains routines, which are used by various applications; loaded into memory when executable that links to them is loaded. They are then shared with other applications.
When added, new library file details are passed on to **/etc/ld.so.conf** (default system library info)
Running **ldconfig** updates changes in that file and loads the shared libraries from locations specified by **/etc/ld.so.cache**.

ldconfig -f <config-file>
- -C <cache-file> where library updates will be stored
- -v view details of library file, rebuilds cache
- -p - show shared library cache
- -n /<location> update the library file info in the specified location instead of the default

ldd -v <program-name> - List dynamic dependencies of executable files or shared objects.
LD_LIBRARY_PATH environment variable

Tuning Resources with sysctl

/proc/sys/ - has directories for all interfaces the kernel offers: abi, crypto, debug, dev, fs, kernel, net, sunrpc, vm
/proc/sys/kernel/ for core kernel functionality, **/proc/sys/net/** for networking, **/proc/sys/vm/** for virtual memory
kernel files for different parameters like "osrelease" or "hostname"
cat /proc/sys/net/ipv4 spits out 1. Change this to off with **echo 0 > ip_forward** (not persistent)
vm contains "swappiness" holding a value 0-100 (eagerness of kernel to swapping out unneeded memory)
Default is 30. Greater value (60) might help kernel swap out faster. Change it the same way but it wouldn't stick after reboot (not persistent).

/etc/sysctl.conf contains the default system configurations which hold these sort of values. It should be edited if you want to change these values and have them stick after a reboot.

sysctl -a -- to display **/proc/sys/** values
sysctl -a | grep forward ---to show forwarding/ routing files/values

Just like you see in Java, net.ipv6.conf.lo.mc would refer to the file **/proc/net/ipv6/conf/lo/mc**

```
[root@rhelserver ~]# sysctl -a | grep forward
net.ipv4.conf.all.forwarding = 0
net.ipv4.conf.all.mc_forwarding = 0
net.ipv4.conf.default.forwarding = 0
net.ipv4.conf.default.mc_forwarding = 0
net.ipv4.conf.ens33.forwarding = 0
net.ipv4.conf.ens33.mc_forwarding = 0
net.ipv4.conf.lo.forwarding = 0
net.ipv4.conf.lo.mc_forwarding = 0
net.ipv4.ip_forward = 0
net.ipv6.conf.all.forwarding = 0
net.ipv6.conf.all.mc_forwarding = 0
net.ipv6.conf.default.forwarding = 0
net.ipv6.conf.default.mc_forwarding = 0
net.ipv6.conf.ens33.forwarding = 0
net.ipv6.conf.ens33.mc_forwarding = 0
net.ipv6.conf.lo.forwarding = 0
net.ipv6.conf.lo.mc_forwarding = 0
```

On boot the **sysctl** process is started. In the past, it just checked **/etc/sysctl.conf**, which is now empty (as of RHEL7). The file instructs that "default settings live in **/usr/lib/sysctl.d/00-system.conf** To override those settings enter new settings here, or in an **/etc/sysctl.d/<name>.conf** file"
So you can still put your custom stuff in **/etc/sysctl.conf**, but putting them in an **/etc/sysctl.d/<name>.conf** might be considered more organized if many custom changes are applied
Before you add custom scripts to **/etc/sysctl.d/** it will likely only contain **/etc/sysctl.d/00-sysctl.conf** which it turns out is simply a symlink to **/etc/sysctl.conf**

/usr/lib/sysctl.d/ contains: 00-system.conf 50-default.conf libvirtd.conf
The numbers in the filename represent the read-order

So, in **/etc/sysctl.d/** run **vim 50-ipforward.conf** and put in it **net.ipv4.ip_forward = 1**, save and on reboot you should be able to route IP packets.

The **sysctl** command promises to set the value and write to the **sysctl** files. Sander Vugt (whose examples I just used in this section on /proc and kernel modules) recommends against trusting it, instead suggesting to make sure by going directly to the source and echoing the value in **/proc** as demonstrated.

<u>sysctl command options</u>
sysctl to modify kernel parameters at runtime. The parameters available are those listed under /proc/sys/
-w {variable}={value} Write a parameter value/ change the sysctl setting.
{variable}={value} Set a key parameter value.
-n Disable the printing of the key name while displaying the kernel parameters.
-e Ignore errors about unknown keys.
-a Display all the parameter values that are currently available.
-A Display those in a tabular format.

More on /proc and /sys

cat /proc/cpuinfo - to display cpu info
/proc/net/ has files containing protocol information and settings, such as routing tables (used by netstat, ss)
/proc/meminfo - the used and unused memory and the shared memory and buffers used by the kernel.
/proc/version - produces same mostly as uname
/proc/cmdline - command line boot options passed to the kernel by the boot loader at boot time.
/proc/devices -list of device drivers (hardware) configured into the currently running kernel.
/proc/filesystems - list of filesystems that are configured into the kernel for mounting
/proc/partitions - partition info: the major and minor number of each partition, name, and number of blocks.
/proc/dma direct memory access. DMA gives hardware devices direct access to memory independent of the CPU.
/proc/interrupt lists the interrupt request (IRQ) channels in use.
/proc/iomem - mapping of the memory allocated to each device and the input/output port assignments for memory.
/proc/modules - lists the kernel modules that the computer is currently using.
/proc/bus - contains a file or directory for each USB device attached

Original UNIX systems needed to access structured data so user-space applications could find out about process attributes. Early programs like ps found out about the running processes by directly accessing **/dev/mem** or **/dev/kmem**, and interpreting the raw data). That requires root access, so the applications that use them have to be setuid or SGID. Also it's not good to expose system data directly to user-space. One solution was to make these types of info available through system calls, but creating new system calls over and over to export small process data wasn't so good either. The /proc filesystem was the answer- where interfaces and structures (directories and files) could be kept the same, even as the underlying data structures in the kernel changed.

On kernel startup virtual filesystems are made to reference hardware and resources:
- **udev** creates the virtual filesystem **/dev** to put hardware definitions (like addresses) into files representing them as interfaces so other things can talk to the them
- **sysctl** then helps set up **/proc** and **/sys** virtual filesystem representations of how the kernel modules and drivers to interact with those /dev device files

The **procfs** /proc virtual filesystem was only meant to hold legacy ***process*** information, system attributes from a few main systems. Since it is easily accessible from both kernel and user-space it eventually got the reputation as the convenient place to put other read/write system files to adjust settings, kernel and subsystem operation (cpuinfo, memory statistics, device information). Things then arbitrarily got thrown in created clutter with device data stuck in different spots all over the place.

The **sysfs** /sys virtual filesystem was implemented in kernel 2.5-2.6 (2003 or so) to organize things better, separate device and driver ***system*** information from **/proc** to add structure, provide a uniform way to expose system information and control points (settable system and driver attributes) to user-space from the kernel. For each object found to put in **/dev** on the system, the kernel automatically creates directories in **/sys** when drivers are registered based on the driver type and their values (representing the device hierarchy too)

Many of the legacy system information and control points are still accessible in /proc - entries already added to /proc were allowed to remain for backward compatibility (especially traditional items like CPU and memory). All new device busses and drivers are expected to expose their info and control points via sysfs.
Note: Many traditional UNIX and Unix-like operating systems use /sys as a symlink to the kernel source tree

<u>The /sys Directory Structure</u>
/sys/block - has an entry for each block device (mostly drives use data blocks)
/sys/bus/devices/ - symlinks for each device - point to device's directory under root/
/sys/bus/drivers/ has a directory for each device driver that is loaded
/sys/class/- has files for each class of devices
/sys/devices lists devices discovered, in a directory tree reflecting/representing the device hierarchy
/sys/firmware/
/sys/net/
/sys/fs/ - where each filesystem exporting attributes creates its hierarchy- see ./fuse.txt
/sys/dev/char/ and **/sys/dev/block/** hold symlinks named <major>:<minor> pointing to the appropriate

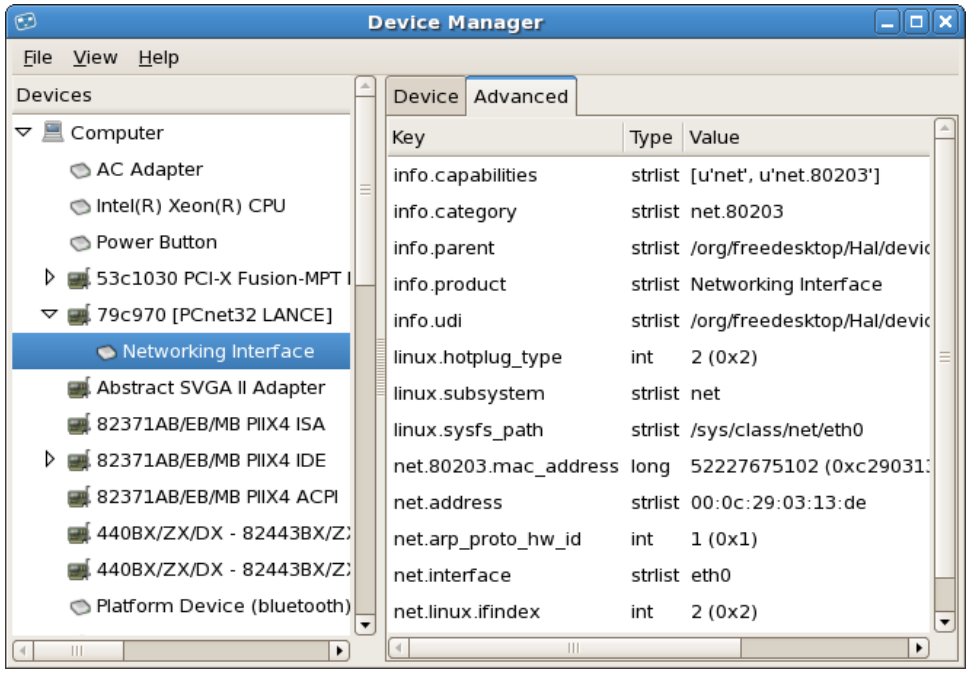
Quick examples:

Setting laptop brightness (not persistent)	**echo N > /sys/class/backlight/acpi_video0/brightness**
Get a network card's MAC address	cat /sys/class/net/enp1s0/address

/dev - Device Nodes

- Device drivers mapping service requests to device access represent HW resources in a device tree
- contains vital information such as the device type, with a minor # to identify device, major # to identify driver
- acts as an interface between the OS and hardware; part of OS or installed on-demand

Device Tree
- "a structure that lists all HW installed on a system and assigns device nodes to them."
- auto generated by the computer's RAM on startup, when a new device is installed, or when a device or system configuration is modified.



Special Devices

/dev/zero Provides unlimited null characters (0 bytes) for writing into any program or file. It is used for generating an empty file of certain size.

/dev/null Does not provide any data to a program or file. It discards all data written to it. It is used as an output file when the output is not required by the user and should be trashed.

/dev/random Functions as a random number generator; gathers random input from device drivers and other sources, saves it as bits in an entropy pool, output as bytes to applications.
When the pool is exhausted, will block the reading application until more random input is collected.

/dev/urandom Like /dev/random, except doesn't block the reading application if the entropy pool is exhausted; also uses pseudorandom input (less secure)

mknod [OPTION] {NAME} {TYPE} [MAJOR MINOR].
Create device files that are not present. Makes use of the major and minor node numbers of a device

dmesg

dmesg shows info about all the hardware controlled by the kernel. CPU, memory, disk drives, network cards, etc. Shows the contents of **/var/log/dmesg**, which holds the kernel ring buffer (ring because it dumps old messages as new ones come in, maintaining the same file size)

It includes actions taken at startup, like configuring hardware devices.
See error messages as they occur with **watch "dmesg | tail -20"**
Setting the level to number 1 (-n 1) only allows panic messages to be seen.

dmesg > kernel_msgs.txt - dump current ring buffer contents into a file, handy for emailing for help troubleshooting
← Pipe output to more, less, head, tail, or grep such as **dmesg | grep -i memory**
← **cat /var/log/dmesg** can generate similar output, but running dmesg provides more control
←
Some options for dmesg:

-C, **--clear**	Clear the ring buffer. you can still view logs stored in '**/var/log/dmesg**' files
-c, **--read-clear**	Clear the ring buffer contents after printing.
-f, **--facility** *list*	Restrict output to defined (comma separated) list of facilities
-k, **--kernel**	Print kernel messages.
-l, **--level** *list*	Restrict output to defined (comma separated) list of levels.
-r, **--raw**	Print the raw message buffer, i.e., don't strip the log level prefixes.
-s, **--buffer-size** *size*	Output **size** to query the buffer; 16392 by default. If you set larger, can view entire buffer.
-u, **--userspace**	Print userspace messages.
-x, **--decode**	Decode facility and level (priority) number to human readable prefixes.
-n, **--console-level** *level*	Set priority level that messages are logged using syslog level number or name (*-n 1* or *-n alert* prevents all messages except emergency) All levels are still written to **/proc/kmsg**, so syslogd can still be used to control exactly where kernel messages appear. When the *-n* option is used, dmesg will not print or clear the kernel ring buffer.

- "dmesg Explained" article from Linux Gazette website explains **dmesg** lines
- *http://www.tldp.org/LDP/LG/issue59/nazario.html*

hald - Hardware Abstraction Layer (HAL) daemon provides all applications with data about current hardware.
dbus - Desktop Bus - Inter-Process Communication (IPC) system
- Allows processes to communicate with each other provides notification system for HAL events
- Can start services for an application's needs- applications register with dbus daemon to participate.
- Can send system wide alerts such as “new hardware detected” and “print queue modified.”

Important Log File Locations and Using journalctl

Commands for viewing, including those with 'relevant command' listed: cat, tail, grep, less, zcat, zgrep, zmore, etc.

Log location	***Relevant command***	***Comments***
/**var/log/messages**		Includes some security related info- where PAM is logged (Ubuntu after Natty instead uses /var/log/syslog)
/var/log/boot.log		Self-explanatory see also boot.log.* for older logs
/var/log/wtmp	**last**	Successful (mnemonic "working") logins, logouts, reboots
(same as above)	**last reboot**	Shows just reboots, same with keyword logins, etc
/var/log/btmp	**lastb**	Shows history of failed login attempts (mnemonic "bad")
/var/log/lastlog	**lastlog**	Shows recent user logins
/var/log/secure		Authpriv messages, more
/var/log/dmesg	**dmesg**	Boot and dbus messages
/root/install.log		Is updated - good to keep copies after fresh installs of things
/var/log/anaconda.log		System installation info on Red Hat systems
/var/log/dpkg.log		dpkg logs for Debian installs/removes (see also /apt/ directory)
/var/log/yum.log		yum command log for Red Hat installs/removes
/var/log/kern.log		Kernel logs
/var/log/daemon.log		Info from various background daemons
/var/log/user.log		Information about all user level logs
/var/log/audit/		Audit daemon (auditd).
/var/log/setroubleshoot/		SELinux's setroubleshootd
/var/log/sssd/		System security services (remote directory access, auth)
/var/log/cron.log		Crond logs
/var/log/sa/		Daily sar files for systat
/var/log/maillog		Mail server logs (sendmail/ dovecot)
/var/log/qmail/		Qmail log directory
/var/log/httpd/		Apache access and error logs directory
/var/log/lighttpd/		Lighttpd access and error logs directory
/var/log/mysqld.log		MySQL database server log file
/var/log/cups		Printer and printing related log messages
/var/log/samba/		Samba info, Windows support

- For authenication, Debian-based use **/var/log/auth.log** while Red Hat-based use **/var/log/secure**
- **/var/log/faillog** - if available can also provide info on failed login attempts
- Similar to **wtmp**, but perhaps not as useful is **/var/log/utmp**

Remember the syslog standard levels/priorities:
0= emerg, 1=alert, 2=crit, 3= err, 4=warning, 5=notice, 6=info, 7=debug
Levels with a higher numerical level give less information (7 least, 0 most)

journald and journalctl options

-f	New log entries as they are added	journalctl -u mysql -f
-k	Kernel messages (example: 5 boots previous	journalctl -k -b -5
-u	Messages for specified systemd service	journalctl -u httpd
-b	Boot msgs; last boot, use -1; two boots ago -2; etc.	(see above -k example)
--list-boots	List system boots	
-r	Show in reverse order; most recent entries first	
-p	Display messages by priority	journalctl -p err
--since, --until	Time range; formats: 09:00; "1 hour ago", 2 days ago	journalctl --since "2017-05-23 23:15:00"
-o	Output options, includes short, verbose export > filename	journalctl -o json-pretty
_PID, _UID	Messages produced by a specific PID, UID, GID	journalctl _UID=100 (remember **id** command)
_COMM, etc.	Name of executable or path, hostname. Similar options	journalctl _HOSTNAME=myhost
	Various attributes supported- see man page for list	_SELINUX_CONTEXT= system_r:policykit_t

The journal is saved in the **/var/log/journal/**

Integrating and Configuring rsyslogd and journald

Rsyslog is still central to logging - journald doesn't have all the mechanisms to do things
journalctl: -b for booting info, --since=yesterday or , -o for verbose, u= service (or PID) for process
journalctl without options just dumps from the binary to screen.
/etc/systemd/journald.conf

Sending journald logs to rsyslog: *In **/etc/syslog.conf** add:*
$modload imuxsock - (input module unix socket)
$OmitLocalLogging off
- and -
*In **/etc/rsyslog.d/listend.conf**, add:* $SystemLogSocketName /run/systemd/journal/syslog

Sending rsyslog to journald *In **/etc/rsyslog.conf** add:*
$modload omjournal *.* :omjournal:
(this tells it, from any facility, and any priority, send to omjournal

Other input modules (Apache into rsyslog example)

```
$ModLoad imfile

$InputFileName /var/log/httpd/error_log
$InputFileTag apache-error:
$InputFileStateFile state-apache-error
$InputRunFileMonitor
```

Exporting to a DB using an output module:

```
$ModLoad ommysql

$ActionOmmysqlServerPort 1234
*.* :ommysql:database-servername,database-name,database-
userid,database-password
```

Enabling remote logging in /etc/rsyslog.conf (these are there for us in the file, just commented out)
Provides UDP syslog reception - classical method - best backward compat but you can lose messages
$ModLoad imudp
$UDPServerRun 514

Provides TCP syslog reception - the better option
$ModLoad imtcp
$TCPServerRun 514

For sending out, look at the forwarding rules and find this:
Replace remote-host with IP addy or servername in hosts files
. @@remote-host:514

Sample conf file lines. Basic syntax is facility.level … target

```
*.info:mail.none:authpriv.none:cron.none    /var/log/messages
        .none is exclusion, * is wildcard.  This line logs everything of level 1 or higher except as noted
 authpriv.*                 /var/log/secure
        This catches all messages from authpriv and puts into /secure
*.emerg                     *
        Sends all emergency messages to all tty's and logs (local- not remote)
uucp.news.crit              /var/log/spooler
        News errors using uucp facility
local7.*                    /var/log/boot.log
        local7 is a boot facility.  See more facilities in the syslog man page
```

Logrotate - /etc/logrotate.conf **-** Specifies to rotate logs, daily, weekly, monthly; how long to keep logs before deleting; a create directive to replace the moved log with a blank empty file to use; dateext directive to use date as a filename extension; compress or not
There is also an include directive pointing to /etc/logrotate.d as a place for specific RPMs to throw logs /etc/logrotate.d/ to hold more granular rule files for syslog, http, yum, up2date, samba, etc., processes).

Basic Hardware Utilities

lspci Display info about all the PCI buses and all the peripheral components connected to a computer.
-k shows the kernel drivers that support the device.**-v** for verbose on most of these ls commands
-t displays a tree diagram that shows connections between all busses, bridges, and devices.
lsusb Display all the USB components connected to a computer. gives more info
-v for verbose, **-s bus_name** to specify a bus
There is also **lshw, lshal**, **lscpu**, **lsblk**, **lspcmcia**
lshw -X - a graphic frontend
lshw -html > hardware.html -- Create a html overview of the hardware

hwinfo --<hw_item> - displays HW info using libhd **- <hw_item>** is one of the following: all, bios, block, bluetooth, braille, bridge, camera, cdrom, chipcard, cpu, disk, dsl, dvb, fingerprint, floppy, framebuffer, gfxcard, hub, ide, isapnp, isdn, joystick, keyboard, memory, modem, monitor, mouse, netcard, network, partition, pci, pcmcia, pcmcia-ctrl, pppoe, printer, scanner, scsi, smp, sound, storage-ctrl, sys, tape, tv, usb, usb-ctrl, vbe, wlan, zip
--short - for brief output
--debug level - set debuglevel
hwinfo --disk show all disks
hwinfo --short --block just an overview
hwinfo --disk --only /dev/sdb - show a particular disk
hwinfo --disk --save-config=all - save disk config state
hwprobe=bios.ddc.ports=4 hwinfo --monitor - try 4 graphics card ports for monitor data (default: 3)

kudzu [options]
- deprecated in RHEL6 - probed and compared with /etc/sysconfig/hwconfig. Configuration options, ran at boot time.

Kernel State Monitoring Utilities
uname OS name, version, license, processor, and hardware details.
uptime Duration the system has been running, number of users currently logged on, and load averages
tload Graphical representation of load average for the past 1 minute, 5 minutes, and 15 minutes.
w is essentially a combination of uptime, who, and ps -a

Memory Monitoring Utilities
free Total memory: free, used, shared, buffered, and cached. -m for megabytes, -k for kb, -s delay
vmstat Virtual memory usage; I/O address info, processor allocation currently running
pmap Display the mapping of processes with memory resources.
iostat Reports on CPU and device utilization. Usage statistics for storage devices and partitions.
memstat -a
systat -m

Disk and Filesystem Utilities
partprobe - tells the kernel to re-read the partition table
-d Cancel any updates; **-s** Display the storage devices and their partitions.
df -h list volumes- memnonic disk free
du -h -d1 file sizes, human readable, depth=1 --max-depth=1; **-c** shows combined total at bottom
cifsiostat - Average reads and writes/sec vs those ops issued per sec, files opened/closed/deleted per sec

Other useful stuff:
id - lists current user's uid, gid, and group memberships
file <filename> Dumps file info including file type
stat <filename.txt> Shows full attributes of file
which <command> -a Shows the pathnames for the command using $path env var
lsof | grep <keyword> List open files, filter by keyword
Remember you can use the watch command (like **watch free**) to have it update on the screen
watch -n 1 -d free Sets a 1 sec interval and -d will highlight values that changed

Make a baseline of system performance on a newly installed Linux system
/proc/meminfo - the used and unused memory and the shared memory and buffers used by the kernel.
/proc/cpuinfo - the CPU information and system architecture dependent items
dmesg displays a snapshot of information about the hardware that is controlled by the kernel, and that output can be redirected to a file for use in system baseline documentation.

Process Management

Moving a running job from foreground to background
- Typing ctrl-z *stops* and places in the background, then type **bg** to *resume* in background
- Type **jobs -l** to see it listed as running (lists all background jobs)
- Executing **fg <job#>** with the number it is listed as will bring it to the foreground
- **dd if=/dev/zero of=/dev/null &** - Copying nothing to nowhere;
- The **&** sends it to the background upon execution without having to stop and resume

[A "shell job" is any command run from the shell. Processes belong to the shell from which they were started]
nohup ./test > test.output - makes process keep running when terminal/tty has been closed
(means to run without attaching process to parent terminal)

Kill Signals
kill -9 <PID> - brute-force terminate process
killall <program-name> - affects all matching processes running

Num.	Name	Default Action	Description
1	SIGHUP	terminate process	Terminal line hangup- restarts the process. After doing so, the process will keep the same PID that it had before. Useful for restarting a service after making changes to a configuration file.
2	SIGINT	terminate process	Interrupt program - sends a ctrl-c key sequence
3	SIGQUIT	create core image	Quit program
6	SIGABRT	create core image	Abort program (formerly SIGIOT)
9	SIGKILL	terminate process	Brute-force kill; process may not clean up - resources allocated may remain allocated until the system is restarted
15	SIGTERM	terminate process	Default- allows process to clean up before exiting

CPU Priority
"Niceness" values for CPU priority - remember that -20 is highest priority and 19 is the lowest. Default is 0
- Use **nice** for *starting* programs: **nice -19 ./test** (or PID) - sets to 19 (is not minus 19, which would be set with **--19**)
- Use **renice** for adjusting currently running processes: **renice <priority> -user** (or PID)
- **renice 17 -p 1234** changes the nice value of the job with process id 1134 to 17; no dash for command option
- Change the nice value of process 1234 to -3 with: **renice -3 -p 1234**
- Modify the priority of all processes that belong to a group or user with the -g and -u (instead of -p for process)
- It is recommended changing nice in steps of -5 at a time
- *Only root can apply negative nice values*
- You can set the default nice value of a particular user or group in the /etc/security/limits.conf
- It uses this syntax: [*username*] [hard | soft] priority [*nice value*]
  user12 hard priority 1

The **-n** modifier produces different behavior in nice and renice
- In nice, using -n option adjusts the value from the default (0) and the "-" will actually work as minus.
- Increase the priority: **nice -n -5 ./test.sh** - Decrease the priority: **nice -n 5 ./test.sh**
- So using the -n makes it less confusing with the whole issue of whether it is a minus or a simple hyphen
- In renice, the -n option specifies the actual number so **renice -n -19 -p 3534** sets the nice value to -19

If you have to mess with nice alot, then it is a sign to increase system resources (RAM, but primarily CPU) to handle the load on the server)

Using PS to Inspect Processes

The ps command 3 syntax styles. BSD style - options are not preceded with a dash (**ps aux**); UNIX/LINUX style - options are preceded by a dash (**ps -ef**) "ps aux" is not the same as "ps -aux". "-u" is used to show process of that user, but plain "u" means show detailed information in the other mode. Usually these can be mixed in Linux: For example "ps ax -f". Finally, some options use the GNU style with double-dashes preceding a word, like --forest.

Most often used is display all processes. The "u" or "-f" options display more detailed info

$ ps ax --or-- **$ ps aux**
$ ps -ef --or-- **$ ps -ef -f** *Using -F can sometimes be more effective than -f*

Other options (there are many):

-u <username>	Display process by user -Multiple usernames can be provided separated by a comma
-f --or-- -F	Show detailed info
-C <processname>	Search by name (must be exact)
-p	Search by PID (separate with comma for multiple e.g., **$ ps -f -p 3150,7298,6544**
--sort=	Prefix with a "-" or "+" to sort descending/ascending (e.g., **$ ps aux --sort=-pcpu,+pmem**)
--forest	Displays ASCII art of process tree. Sort and forest don't work well together
--ppid <PID>	Search by parent PID (show child processes)
-o	Show listed columns sep with commas (e.g., **$ ps -e -o pid,uname,pcpu,pmem**)
-L	Show threads of a process, (e.g., **$ ps -p 3150 -L or ps -C firefox-bin -L -o pcpu,state**)

- Piping to grep instead of using the -C option can be more convenient **$ ps -ef | grep apache**
- Fields for --sort and -o options are available in the manpage section "Standard Format Specifiers" (there are tons)
- Rename the column labels using the format "-o pid,uname=USERNAME,pcpu=CPU_USAGE,pmem,comm"
- When listing usernames, if the length is greater than 8 characters then ps will show the UID instead of username.
- As usual pipe to "less" to ease reading. If sorting remember to consider piping to head 5 or tail -5 as needed, etc.

Common attributes to query/ sort by, etc.
The man page lists many attributes you can interrogate using ps. Here are some of the commonly used ones. pid, comm (just the command name), cmd (the command with all its arguments as a string), uname (username running process), pmem (percent of physical memory being used), pcpu (percent of CPU being used), nice (the niceness value), etime (elapsed time running), state (one letter state code), stat (multiple character state code listing)

```
[root@localhost ~]# ps afx
  PID TTY      STAT   TIME COMMAND
    1 ?        Ss     0:00 init [5]
    2 ?        S<     0:00 [migration/0]
 2438 tty5     Ss+    0:00 /sbin/mingetty tty5
 2439 tty6     Ss+    0:00 /sbin/mingetty tty6
 2440 ?        Ss     0:00 /usr/sbin/gdm-binary -nodaemon
 2506 ?        S      0:00  \_ /usr/sbin/gdm-binary -nodaemon
 2511 tty7     Ss+    0:21      \_ /usr/bin/Xorg :0 -br -audit 0 -auth /var/gdm/:0.Xauth -nolist
 2525 ?        Ss     0:00      \_ /usr/bin/gnome-session
 2561 ?        Ss     0:00          \_ /usr/bin/ssh-agent /bin/sh -c exec -l /bin/bash -c "/usr/
```

The "f" option shows child process as shown above

```
[root@localhost ~]# ps alx
F   UID   PID  PPID PRI  NI    VSZ   RSS WCHAN  STAT TTY        TIME COMMAND
4     0     1     0  15   0   2072   588 -      Ss   ?          0:00 init [5]
1     0     2     1 -100  -      0     0 migrat S<   ?          0:00 [migration/0]
1     0     3     1  34  19      0     0 ksofti SN   ?          0:00 [ksoftirqd/0]
```

The "l" option lists more info

```
[root@localhost ~]# ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0   2072   588 ?        Ss   13:40   0:00 init [5]
root         2  0.0  0.0      0     0 ?        S<   13:40   0:00 [migration/0]
root         3  0.0  0.0      0     0 ?        SN   13:40   0:00 [ksoftirqd/0]
```

The "x" option shows info on CPU and RAM usage

PS output columns:
VSZ= virtual memory size, RSS= resident memory size, STAT= state (S=sleep, R=running, etc), F represents number of forked processes, PPID is parent PID. WCHAN is the name of the kernel function the process is sleeping in

States:
I - Idle (sleeping >20 seconds).
R - Running or runnable (on run queue)
S - Sleeping <20 seconds. (waiting for an event to complete)
T - A stopped process (either by a job control signal or because it is being traced)
D - Uninterruptible sleep (usually IO)
U - Process in uninterruptible wait.
Z - Marks a dead ("zombie") process
W - Paging (not valid since the 2.6.xx kernel)
X - Dead (should never be seen)
Additional state codes
+ - Is in the foreground process group of its control terminal.
< - Has raised CPU scheduling priority.
N - Has reduced CPU scheduling priority (see setpriority(2)).
> - Has specified a soft limit on memory requirements and is exceeding that limit; such a process is not swapped.
A - Process has asked for random page replacement
E - Process is trying to exit.
l - Is multi-threaded (using CLONE_THREAD, like NPTL pthreads do)
L - Has pages locked in memory (for example, for raw I/O).
s - Is a session leader.

Some examples

Display top 5 CPU using processes.	**ps aux --sort=-pcpu | head -5**
Display elapsed time of processes	**ps -e -o pid,comm,etime**
View process in realtime	**watch -n 1 'ps -e -o pid,pmem,pcpu --sort=-pmem | head -15'**
Output PID, command, and nice value	**ps -o pid,comm,nice -p 594**
List by % cpu with sed	**ps -e -o pcpu,cpu,nice,cputime,args –sort pcpu | sed '/^ 0.0 /d'**
List by mem (KB) usage	**ps -e -orss=,args= | sort -b -k1,1n | pr -TW$COLUMNS**
Display process hierarchy in a tree style	**ps -f --forest -C apache2**

Display child processes of a parent process
$ ps -o pid,uname,comm -C apache2

PID	USER	COMMAND
2359	root	apache2
4524	www-data	apache2
4525	www-data	apache2

The first process owned by root is the main apache2 process and all other apache2 processes have been forked out of this. We now interrogate that here:
$ ps --ppid 2359

PID	TTY	TIME	CMD
4524	?	00:00:00	apache2
4525	?	00:00:00	apache2

...

pstree - outputs a tree view of processes, which can be preferable over **ps --forest**

```
[root@localhost ~]# pstree
init─┬─acpid
     ├─atd
     ├─auditd─┬─audispd───{audispd}
     │        └─{auditd}
     ├─automount───4*[{automount}]
```

TOP

Combines **ps**, **uptime**, **free** and updates regularly; default sorts task by CPU usage- can kill(k), renice(r)
To hide / show header lines and increase display space, press 'l' for load avg (first line); pressing 't' toggles CPU states (2nd/ 3rd lines); and pressing 'm' toggles memory info (4th/ 5th lines)

The typical header:

```
top - 07:18:59 up 17:38,  2 users,  load average: 0.21, 0.15, 0.10
Tasks: 116 total,   3 running, 113 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.3%us,  0.0%sy,  0.0%ni, 99.7%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   1035108k total,  1014336k used,    20772k free,   137732k buffers
Swap:  2097144k total,        0k used,  2097144k free,   646340k cached
```

- First line displays same contents as **uptime** with CPU load average provided in 1, 5, and 15 minute readings
- Numbers are "by average how many runnable processes are in the runqueue waiting to be served by the kernel's scheduler" meaning one-at-a-time per cpu core.
- What is the "optimal setting?" If you have one CPU, the optimal setting would be 1, 16 cpus, 16
- Press "1" to list CPU lines by individual CPUs instead of a total (this is the third line down that says %CPU(#)

The CPU line displays the following:

us - user space
sy - system - kernel space
ni - user nice - time spent on low priority processes
id - idle - time spent idle
wa - I/O wait cpu time - time spent in wait (on disk)
hi - hardware irq - time spent servicing/handling hardware interrupts
si - software irq - time spent servicing/handling software interrupts
st - steal time - involuntary wait by virtual CPU while a hypervisor is servicing another processor (or) %CPU time stolen from a virtual machine

The memory line is identical to running the **free** command
- About 30% memory should always be available for buffer and cache. If you don't have that, or are loading too many programs, the least active programs will release the pages they have allocated to free up memory
- Cache - structured data: filenames, etc - Buffer- unstructured data like parts of files to write to disk
- Swap is allows cache and buffers dedicated to RAM for operations

```
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
    1 root      15   0  2072  596  512 S  0.0  0.1   0:00.36 init
    2 root      RT  -5     0    0    0 S  0.0  0.0   0:00.00 migration/0
    3 root      34  19     0    0    0 S  0.0  0.0   0:00.51 ksoftirqd/0
    4 root      RT  -5     0    0    0 S  0.0  0.0   0:00.00 watchdog/0
```

By default, processes are listed by CPU usage, < and > sort by the relatively adjacent column in that direction.
Press 'M' key to sort the process list by memory usage
Press 'P' - to sort the process list by CPU usage.
Press 'N' - to sort the list by process id
Press 'T' - to sort by the running time.
Press 'R' - to sort in reverse order
Type F/f for available fields. Added fields have to be saved to use later by typing 'W' (to write config to **~/.toprc**)
Update the output on-demand by pressing the space bar.
Typing 'x' will boldface the column the processes are sorted by. Use b for a bg color instead
c - will show or hide the command's absolute path, and arguments
n - change the number of processes to display (0 for maximum)
d - change output update frequency, will prompt to enter new time in seconds
O/o - to change the order the columns are displayed (left-to-right)
h or ?- display help for interactive top commands
1 - display all CPUs/ cores separately
A - Split output into 4 individually configurable/sortable views; then use "a" to cycle between different views.
b or z - highlight running process
Type 'k' to kill and send specific signal to process (default is 15 SIGTERM), 'r' to renice; 'S' to just send signal
U/u - Filter processes of a specific user
Typing V gives a 'forest' view similar to the same usage in ps

Typing 'F' brings up this window allowing to choose a column to sort processes by

```
Current Sort Field:  K  for window 3:Mem
Select sort field via field letter, type any other key to return
  a: PID        = Process Id                 u: nFLT       = Page Fault count
  b: PPID       = Parent Process Pid         v: nDRT       = Dirty Pages count
  c: RUSER      = Real user name             w: S          = Process Status
  d: UID        = User Id                    x: COMMAND    = Command name/line
  e: USER       = User Name                  y: WCHAN      = Sleeping in Function
  f: GROUP      = Group Name                 z: Flags      = Task Flags <sched.h>
  g: TTY        = Controlling Tty
  h: PR         = Priority                 Note1:
  i: NI         = Nice value                 If a selected sort field can't be
  j: P          = Last used cpu (SMP)        shown due to screen width or your
* K: %CPU       = CPU usage                  field order, the '<' and '>' keys
  l: TIME       = CPU Time                   will be unavailable until a field
  m: TIME+      = CPU Time, hundredths       within viewable range is chosen.
  n: %MEM       = Memory usage (RES)
  o: VIRT       = Virtual Image (kb)       Note2:
  p: SWAP       = Swapped size (kb)          Field sorting uses internal values,
  q: RES        = Resident size (kb)         not those in column display.  Thus,
  r: CODE       = Code size (kb)             the TTY & WCHAN fields will violate
  s: DATA       = Data+Stack size (kb)       strict ASCII collating sequence.
  t: SHR        = Shared Mem size (kb)       (shame on you if WCHAN is chosen)
```

Different ways of starting up top:
top -u <username> - display process from specific user
top -p 1208, 1425 - to display specific PIDs
top -n 1 -b > top-output.txt --use 'batch' output to a file or pipe (text rather than binary) -n is iterations
top -i --to display only active processes (ignore idle)

Be aware that the top command comes in various variants and each has a slightly different set of options and method of usage. To check your top command version and variant use the -v option
For example, in some versions of top, o/O provides a filter prompt to search processes by various criteria
HTOP offers more ease of use but uses 5x as many system resources

When reading the virt / resident memory columns in top, remember it is showing pages- memory pages by default are about 4k so multiply what you see by 4 to get actual memory size (resident memory is the physical memory)

How Much RAM is Really Free? (disk buffers/cache)

Remember that some RAM is available for applications, and simultaneously used for disk buffer/ cached

$ free -m

	total	used	free	shared	buffers	cached
Mem:	1504	1491	13	0	91	764
-/+ buffers/cache:		635	869			
Swap:	2047	6	2041			

If you don't know how to read the numbers, you'll think the ram is 99% full when it's really just 42% (635MB). Notice also 635+869-13=1491 "used" and 91+764+13=~869 "free"

$ free -m

	total	used	free	shared	buffers	cached
Mem:	1834	1757	76	9	0	1021
-/+ buffers/cache:		735	1098			
Swap:	2047	0	2047			

Here 735MB used.
735+1098-76=1757 called "used" - (used+free in buffers/cache minus "free" in mem = reported used in mem).
Shared is usually shared libs. Buffers is when data needs to be committed to disk
Cache is big because files moved to RAM (especially frequently used files) are kept in RAM as long as possible (optimization). Cache is generally freed up when resources are needed for reallocation.
Also consider 1757 "used"-1021 cached is = 735 really used. (and 735+1098 is 1834 total reported)
Here we really have 1098 available

The -/+ buffers/cache numbers are the real deal.

The sysstat Suite and sar

The sysstat suite includes the programs **iostat** (CPU, I/O stats), **mpstat** (CPU stats), **pidstat** (stats on processes), **nfsiostat** (NFS-related stats), and **cifsiostat** (CIFS stats). The flagship of the suite is the **sar** collection and reporting suite, described below. The other tools have their own expanded descriptions afterward

sadc - System Activity Data Collector. writes binary data to /var/log/sa/ named "sa<dd>" (d is day of month)
sa1- script to run sadc with cron or systemd. Collects and stores binary data in the system activity daily data file
sar - System Activity Reporter produces reports from sadc files. Writes ascii files to /var/log/sa/ named "sar<dd>"
sa2 - script to run sar with cron or systemd. Writes a summarized daily activity report
sadf outputs sar reports in CSV, XML, JSON, etc. so you can output to other programs. Can also SVG graphs.

Installation
- If installing from source: **./configure --enable-install-cron**, then **make**, then **make install**
- The option **--enable-install-cron** makes sure to creates **/etc/rc.d/*** stuff for you
- Installs executables under /usr/local/bin
- (old issue?) Run **echo alias sar='LANG=C sar' >> /etc/bashrc** to ensure sar uses 24 hour clock

Ensuring cron is set up
If your version of sysstat didn't do it, create a file under /etc/cron.d directory that will collect the historical sar data.
vi /etc/cron.d/sysstat
*/10 * * * * root /usr/local/lib/sa/sa1 1 1
53 23 * * * root /usr/local/lib/sa/sa2 -A

- If installed from source, sa1 and sa2 is probably in **/usr/local/lib/sa**
- If installed with a package manager, they may be located in be **/usr/lib/sa/sa1** and **/usr/lib/sa/sa2**
- The above runs sa1 to collect data every 10 minutes, and compile that data about midnight with sa2
- sa1 is executed with options "1 1" - run once with 1 second interval

Log management
Generated log files are only kept for a limited time (usually ~28 days). One option is to have a script back them up before deletion- however, the configuration file /etc/sysconfig/sysstat holds a HISTORY parameter to the number of days you want to keep the log files. The maximum is supposedly 28 days, but if set to more, log files will be stored in a month-by-month directory. Then, log files will be pointing to a symlink such as YYYYMM/saDD

$ sar -u ALL 1 3 - report today's CPU usage (so far) every 1 sec, 3 times; the ALL option gives extended info
$ sar -u -f /var/log/sa/sa10 - reports for a specific day (file); here the 10th day of this month (in the file sa10)
$ sar -P 1 - the -P specifies to report only on CPU core 1 (a quad-core would have 0-3); use **-P ALL** for all cores
$ sar -r -f /var/log/sa/sa10 - most queries use the same syntax as above - this one querying memory stats with **-r**
$ sar -p -d - This query for block devices uses the -p option ("pretty print") so drives listed have familiar names to us.

General options are, **-f** to specify a file (day), **ALL** for extended info or all items (in context), **-p** for human readable, then numbers to specify frequency to run followed by how many times to run before exiting. There is also an option to put in a start time, like **sar -s 09:30:00** for 9:30AM, can be used with **-f** to specify a day. Optional end time with **-e**

The most often used queries are **-u** (CPU), **-r** (memory), **-S** (swap), **-b** (I/O stats), **-d** (block devices), **-n** (network)
Here are some others:
- **-q** - run queue size and load average
- **-w** - processes created per second, and total number of context switches per second
- **-R** - number of memory pages freed, used, and cached per second by the system.
- **-B** - paging statistics. i.e Number of KB paged in (and out) from disk per second.
- -**W** - page swap statistics. i.e Page swap in (and out) per second.

Network activity reporting has a ton of options; **sar -n <keyword>** - where <keyword> can be any of these:

DEV - network devices vital stats for eth0, eth1
EDEV - network device failure stats
NFS - NFS client activities
NFSD - NFS server activities
SOCK - sockets in use for IPv4
IP - IPv4 network traffic
EIP - IPv4 network errors
ICMP - ICMPv4 network traffic
EICMP - ICMPv4 network errors
TCP - TCPv4 network traffic
ETCP - TCPv4 network errors
UDP - UDPv4 network traffic
SOCK6, IP6, EIP6, ICMP6, UDP6 for IPv6
ALL - all options in one output- very long.

vmstat - Memory, Processes, Paging etc

vmstat -a option also displays active and inactive memory. Display units in KB and MB with **--unit k** or **m**
vmstat 2 5 - at every 2 second interval, we want to see 5 polling groups (rows of data). Just like iostat, the first line is going to be bigger because it is generic overview of the last period of total system activity

proc			memory				swap		io		system		cpu			
r	b	w	swpd	free	buff	cache	si	so	bi	bo	in	cs	us	sy	id	wa
2	0	0	120	30180	1736	46696	0	0	424	25	134	77	5	9	86	

Field definitions:
Proc - r: Running waiting for CPU; b: Uninterruptable sleeping processes (waiting for I/O); w: Swapped out processes
Memory - swpd: swap areas (in /proc/swaps); free: Idle memory; buff: buffers; cache: cache used.
Swap - si/so: Memory swapped in and out from disk - use to get more info when top shows huge swap usage
IO - bi/bo: Blocks received from/ to block device (like a hard disk)
System - in: interrupts/ IRQs per second, including the clock; cs: context switches per second. (when the CPU switches tasks among programs)
CPU - us: user time, non-kernel code including nice time; sy: kernel code; system time - network, IO interrupts, etc; id: idle time. Prior to Linux 2.5.41, including IO-wait time.

vmstat -s helps too- lots more info dumped like this:

```
[root@server1 ~]# vmstat -s
      1878184 K total memory
      1804284 K used memory
       329884 K active memory
      1273128 K inactive memory
        73900 K free memory
      1021104 K buffer memory
       151208 K swap cache
      2097148 K total swap
            0 K used swap
      2097148 K free swap
        12824 non-nice user cpu ticks
            3 nice user cpu ticks
         7702 system cpu ticks
       410614 idle cpu ticks
          405 IO-wait cpu ticks
            1 IRQ cpu ticks
         1113 softirq cpu ticks
            0 stolen cpu ticks
     28249840 pages paged in
        49102 pages paged out
            0 pages swapped in
            0 pages swapped out
       552140 interrupts
      1095772 CPU context switches
   1409692776 boot time
         5257 forks
```

Resident memory is the physical memory. swap in/ out- if you see in **top** swap is being used you can check here to see if swap is being used actively. Similarly, bi/bo is blocked in/out so you can tell if blocked processes are spending time reading more or writing. Info is grabbed from **/proc/meminfo**, **/proc/stat**, **/proc/*/stat**

Regular mode is simply "VM mode", but there are others:
Disk Mode [-d, --disk] Use [-D, --disk-sum] for summary statistics
displays these fields for both reads and writes: total (total completed successfully), merged (grouped reads or writes, resulting in one I/O), sectors (sectors read or written successfully), ms (milliseconds spent reading, or writing). Then also displays cur (for I/O currently in progress, and s (seconds spent for I/O)

Disk Partition Mode [-p, --partition device] - Detailed statistics about partition (kernel 2.5.70 or above required). Fields include: reads (reads to this partition), read sectors (read sectors for partition), writes (total number of writes issued), requested writes (number of write requests made)

Slab Mode [-m, --slabs] - Displays the contents of **/proc/slabinfo** Fields include cache (cache name), num (number of currently active objects), total (number of available objects), size (of each object) and pages (the number of pages with at least one active object)

iostat - CPU and I/O Stats for Devices and Partitions

iostat [option] [interval] [count]
Interval specifies time in seconds between each report, and count specifies the number of reports generated before exiting. **iostat 2 5 - indicates** a 2 sec interval, 5 iterations. If you don't specify iterations it will run continuous. When run for the first time, the first report contains information since the system was boot, while each subsequent report covers the time period since the last report was generated, so you may want to ignore the first one.

The first part of the normal output reports the CPU utilization, and should be self-explanatory from other tools like **top** The last two columns show %CPU time idle with an outstanding disk I/O request (lag waiting for disk activity) and without an outstanding disk I/O request. If you only want to get this top part of the report, run **iostat -c**

Linux 2.6.31-17-generic (drt-laptop) 03/24/16 _ii686_ (1 CPU)

avg-cpu:	%user	%nice	%system	%iowait	%idle
	25.99	0.78	7.43	12.77	53.03

Device:	tps	Blk_read/s	Blk_wrtn/s	Blk_read	Blk_wrtn
sda	27.40	797.19	201.27	800902	202208
sr0	0.03	1.24	0.00	1248	0

The device report has transfers per second, number of blocks per second read, number of blocks per second written, total number of blocks read, and total number of blocks written. You can use the **-k** (kilobytes) or **-m** (megabytes) parameter to change the last four columns to be expressed in human-readable terms instead of blocks
For only the device report above, run with **iostat -d**
iostat -x gives stats in an extended report.
Adding a device identifier with **/dev/sdaX** syntax will limit the report to the specified device
Using **-p** in there will show data on partitions, and **-N** gives LVM names and stats.
For NFS info, use the **iostat -n** option. **iostat -z** omits inactive devices.
Like anything, do output redirection into a file, and/or use awk to extract columns you need.
Uses **/proc/stat**, **/proc/uptime**, **/proc/partitions**, **/proc/diskstats**, **/sys**, **/proc/self/mountstats**

mpstat - Multiprocessor Stats
If you don't specify a CPU, it will spit out a global average of all of them. Specify a processor with **-P** and put ALL to get stats on all CPUs.
$ mpstat -P ALL

1:30:26	IST	CPU	%usr	%nice	%sys	%iowait	%irq	%soft	%steal	%guest	%gnice	%idle
1:30:26	IST	all	37.33	0.01	4.57	2.58	0.00	0.07	0.00	0.00	0.00	55.44
1:30:26	IST	0	37.90	0.01	4.96	2.62	0.00	0.03	0.00	0.00	0.00	54.48

$ mpstat -P ALL 2 5 - Just like with iostat - add iteration and interval count The option **-I** shows interrupt stats for each processor, and **-u** says CPU stats in case the other options are used.
*Easiest way: **$ mpstat -A** is the quick and easy equivalent of mpstat **-I ALL -u -P ALL***

pidstat - Stats on Processes

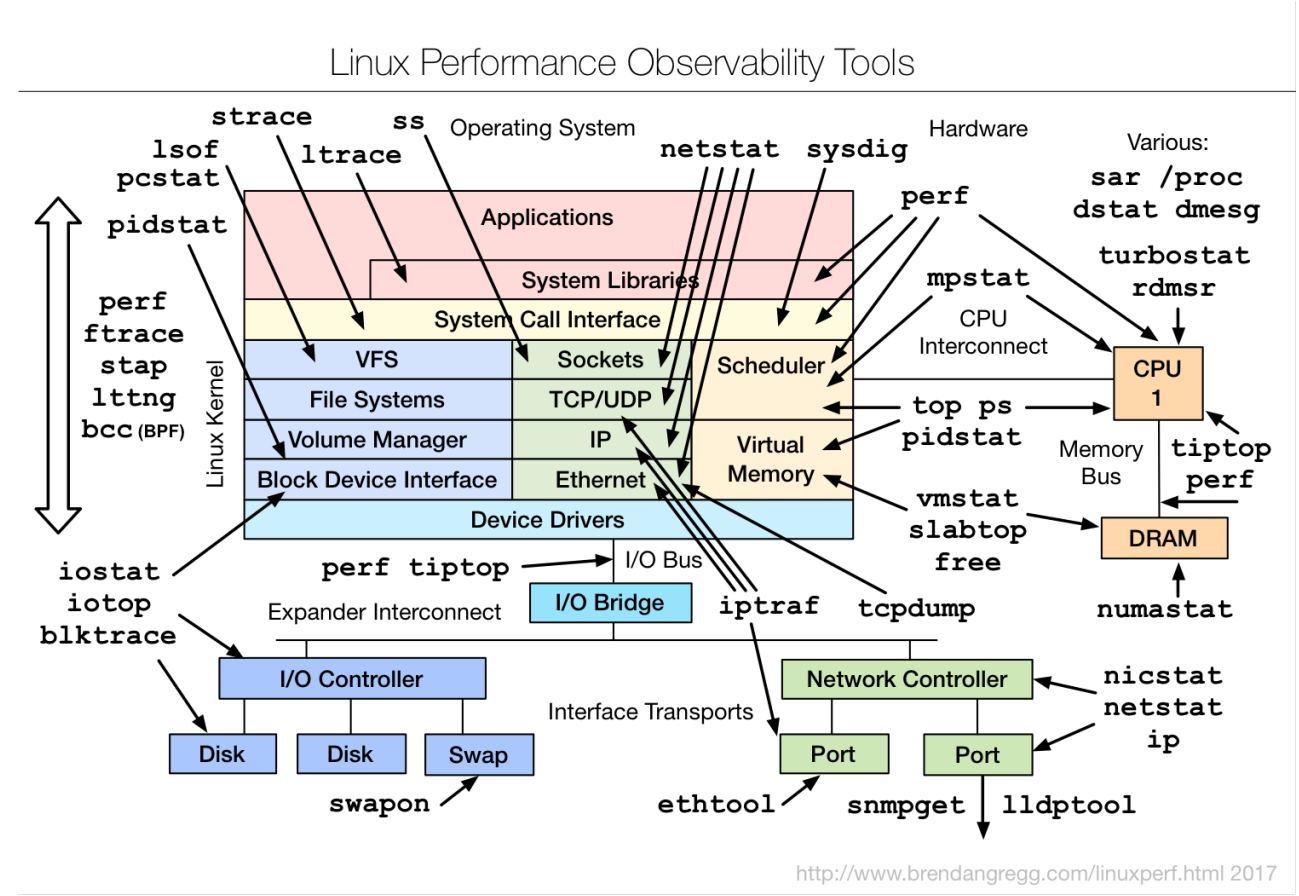
pidstat 2 5 - five reports of CPU statistics for every active task in the system at two second intervals.
pidstat -r - for a focus on page faults in memory statistics:

# Time	UID	PID	minflt/s	majflt/s	VSZ	RSS	%MEM	Command
1409816695	1000	3958	3378.22	0.00	707420	215972	5.32	cinnamon

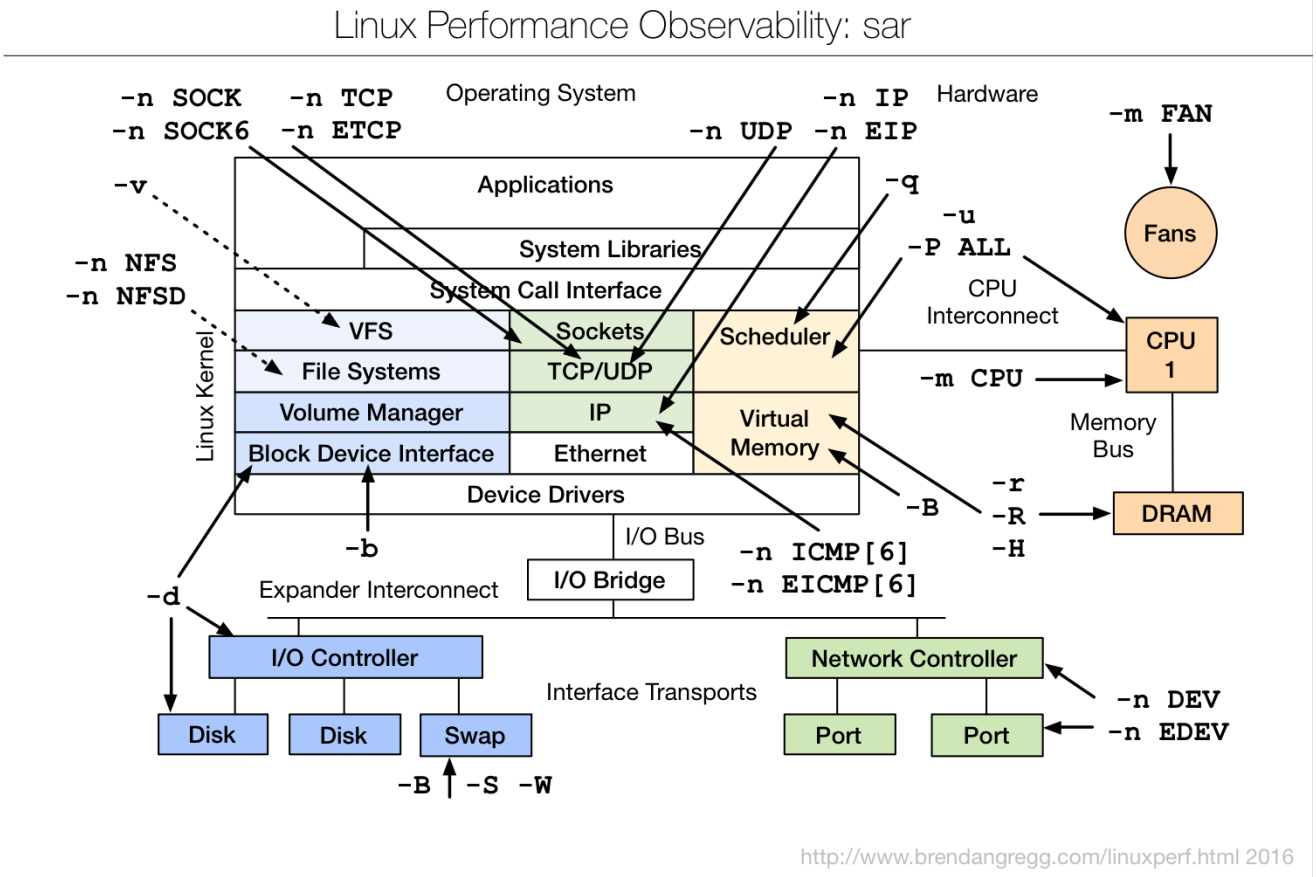
pidstat -d option for disk I/O stats:

	PID	kB_rd/s	kB_wr/s	kB_ccwr/s	Command
03:27:03 EDT	1	0.00	8.00	2.00	init

pidstat -p 1643 for PID 1643. Use **-p ALL** for all processes
Enter a string to match a process name like **-C httpd**; start an executable with **pidstat -e <command>** to monitor it.
pidstat -T CHILD -p 1643 - for the child processes of PID 1643. Only child processes with non-zero statistics values are displayed. **-T ALL** means all tasks and child processes.
pidstat with no options assumes **-p ALL**
For realtime stats use the **-R** option, or **-h** to output one horizontal line of output to easily parse with **awk**, etc.



Strongly recommended: Brendan Gregg's work: http://www.brendangregg.com/linuxperf.html



Command Equivalents - SysVinit and systemd

Action	SysVinit	systemd
Start/stop/restart/reload/status of a service	service ntpd [start | stop | etc...]	systemctl [start | stop etc...] ntpd.service
Restart a service only if already running	service ntpd condrestart	systemctl condrestart httpd.service
Enable or disable service on startup	chkconfig ntpd [off | on]	systemctl [enable | disable] ntpd.service
Is service enabled at startup (this runlevel)?	chkconfig ntpd	systemctl is-enabled ntpd.service
List services that can be started or stopped Used to list all the services and other units	ls /etc/rc.d/init.d/	systemctl OR systemctl list-unit-files --type=service OR ls /lib/systemd/system/*.service AND ls /etc/systemd/system/*.service
Print table of services listing runlevels each is configured on or off	chkconfig --list	systemctl list-unit-files --type=service ls /etc/systemd/system/*.wants/
Print a table of services that will be started when booting into graphical mode	chkconfig --list | grep 5:on	systemctl list-dependencies graphical.target
List what levels this service is config'd on/ off	chkconfig ntpd --list	ls /etc/systemd/system/*.wants/ntpd.service
Create a new service file or modify config	chkconfig ntpd --add	systemctl daemon-reload (this reloads systemd!)
Suspend the system	pm-suspend	systemctl suspend
Hibernate	pm-hibernate	systemctl hibernate
Follow the system log file	tail -f /var/log/messages (or /var/log/syslog)	journalctl -f
System halt	telinit 0, poweroff, halt	systemctl isolate poweroff.target | systemctl poweroff
Change to Single-user mode	telinit 1, s, single	systemctl isolate rescue.target (or runlevel1.target)
Change to Multi-user	telinit 2	systemctl isolate multi-user.target (or runlevel2.target*)
Change to Multi-user with Network	telinit 3	systemctl isolate multi-user.target (or runlevel3.target)
Change to RunLevel 4	telinit 4	systemctl isolate multi-user.target (or runlevel4.target*)
Change to Multi-user, w/ network, x11	telinit 5	systemctl isolate graphical.target (or runlevel5.target)
Reboot	telinit 6, reboot	systemctl isolate reboot.target | systemctl reboot
Emergency Shell	init emergency	emergency.target
Check current runlevel	runlevel	runlevel (deprecated) OR systemctl | **grep (script)**
Change default runlevel	sed s/^id:.*:initdefault:/id:3:initdefault:/	systemctl set-default multi-user.target
Set multi-user target on next boot	sed s/^id:.*:initdefault:/id:3:initdefault:/	ln -sf /lib/systemd/system/multi-user.target /etc/systemd/system/default.target
Execute a systemd cmd on remote host		systemctl dummy.service start -H user@host
Check boot time		systemd-analyze or systemd-analyze time
Kill all processes related to a service		systemctl kill dummy
Get logs for events for today		journalctl --since=today
Hostname and other host information		hostnamectl
Date and time		timedatectl

All recent versions of systemctl assume ".service" if left off. So, 'systemctl start myservicename.service' works like 'systemctl start myservicename'
Default systemd Fedora installs; 0, 1, 3, 5, and 6; have a 1:1 mapping with a specific systemd *target*.
** If you use runlevels 2 or 4 it is suggested that you make a new named systemd *target* as /etc/systemd/system/$YOURTARGET that takes one of the existing runlevels as a base (you can look at /lib/systemd/system/graphical.target as an example), make a directory /etc/systemd/system/$YOURTARGET.wants, and then symlink the additional services that you want to enable into that directory.
Runlevels 2 and 4 are by default just "multi-user" runlevel3 in systemd until defined otherwise

systemd Command Overview

Is systemd is installed on the system? Is it running?	**# systemd-run --version -- # ps -eaf | grep [s]ystemd**
List all the available units . [*.service, *.mount, *.socket, *.device]	**# systemctl list-unit-files**
List all running units. [*.service, *.mount, *.socket, *.device]	**# systemctl list-units**
List all failed units . [*.service, *.mount, *.socket, *.device]	**# systemctl --failed**
Analyze the systemd boot process.	**# systemd-analyze**
Analyze time taken by each process at boot.	**# systemd-analyze blame**
Analyze critical chain at boot (all or a specific service, etc)	**# systemd-analyze critical-chain** (OR **critical-chain httpd.service**)
	"@" = Time after unit is active or started. "+" = Time unit takes to start

Using systemd to Manage Mountpoints, Sockets, Devices Just Like Services

The general systemctl commands that work with services also do the same thing for mountpoints, sockets, and devices (which are seen as service types). Simply specify in the place of "name.service" the proper item, such as tmp.mount, cups.socket, or item.device. The list-unit-files option uses the **--type** directive such as **--type=device** or **--type=socket** accordingly. Starting stopping a mount point simply mounts and unmounts the mountpoint [**systemctl list-unit-files --type=mount** will list all mountpoints, for example]

 # systemctl list-unit-files --type=socket

 # systemctl [start | restart | stop | reload | status | is-active | enable | disable | is-enabled | mask | unmask] tmp.mount

How to enable, disable or check if turned on at boot time (auto start)	**# systemctl [is-active | enable | disable] httpd.service**
How to mask (making it impossible to start) or unmask a service	**# systemctl [mask | unmask] httpd.service**
List all services (including enabled and disabled).	**# systemctl list-unit-files --type=service**
Start, restart, stop, reload and check the status of a service	**# systemctl [start | restart | stop | reload | status] httpd.service**
Check all the configuration details of a service	**# systemctl show httpd**
Get a list of dependencies for a service	**# systemctl list-dependencies httpd.service**
How to a Kill a service using systemctl command.	**# systemctl kill httpd**
Is unit enabled or not right now ("is-active" is for the target's config)	**# systemctl is-enabled crond.service**
Get current CPU Shares of a Service (default CPUShare = 1024)	**# systemctl show -p CPUShares httpd.service**
Increase/decrease CPU share of a process	**# systemctl set-property httpd.service CPUShares=2000**
No unit specified means default.target - Requires=, RequiresOverridable=, Requisite=, RequisiteOverridable=, Wants=, BindsTo= dependencies	
List control groups hierarchically	**# systemd-cgls**
List control groups according to CPU, memory, Input and Output	**# systemd-cgtop**

To enable a service, you must be currently running the target you want the service to start in.
For example, to turn on bluetooth.service in the graphical.target, you have to change to the graphical.target first with **isolate**, then run **enable**.
systemctl isolate graphical.target ; **systemctl enable bluetooth.service**
Makes a symlink **/etc/systemd/system/graphical.target.wants/bluetooth.servic**e pointing to **/usr/lib/systemd/system/bluetooth.service**

How to start system rescue mode	**# systemctl rescue**
How to enter into emergency mode.	**# systemctl emergency**
List current default runlevel in use.	**# systemctl get-default**
Start Runlevel 5 aka graphical mode	**# systemctl isolate runlevel5.target** (OR **graphical.target**)
Set multiuser mode (runlevel 3) as default	**# systemctl set-default runlevel3.target** (OR **multiuser.target**)
*[This set-default line creates a symlink **/etc/systemd/system/default.target** pointing to **/usr/lib/systemd/system/multiuser.target**]*	
Reboot, halt, suspend, hibernate or put system in hybrid-sleep	**# systemctl [reboot | halt | suspend | hibernate | hybrid-sleep]**

Unit and target files in **/usr/lib/systemd/system/** are pointed to by symlinks placed in **/etc/systemd/system/**
Unit files enabled for a specific target will have a symlink in that target's "wants" directory, such as **/etc/systemd/system/multi-user.target.wants**

SysVinit - Directory Structures

Generally, you will find in the **/etc** directory some symlinks to stuff that is actually in the **/etc/rc.d/** directory. This can cause some confusion, since we have some other symlink stuff for backward compatibility for systems that once supported Upstart but no longer do so. This writing ignores all of that and sticks to CentOS 5.5

/etc/init.d is a symlink to the directory **/etc/rc.d/init.d** and the same with **/etc/rc#.d** linking to **/etc/rc.d/rc#.d**, also the same with scripts **rc**, **rc.local** and **rc.sysinit**, who's actual locations is also in the /etc/rc.d/ directory as well Even though you will often see these in **/etc.** Here is where they actually live:

/etc/rc.d/init.d/
/etc/rc.d/rc0.d/
/etc/rc.d/rc1.d/

...and so on....

/etc/rc.d/rc5.d/
/etc/rc.d/rc6.d/
/etc/rc.d/rc
/etc/rc.d/rc.local
/etc/rc.d/rc.sysinit

Service's scripts are in /etc/rc.d/init.d/ (often accessible via the symlink **/etc/init.d**/)

- Each service managed by SystemVinit needs a script in /etc/rc.d/init.d/
- Common elements to the scripts in **/etc/rc.d/init.d/<*servicename*>** are the top several lines, beginning in a prelude declaring the script processor (as in **#!/bin/bash**); followed by a line with name and brief description; another with chkconfig default runlevels the service should be started, and the start and stop priority levels.
- If default is to not be started in any runlevels, a "-" should be used in place of the runlevels list.
- Another entry contains service description (used by ntsysv)
- Finally the general functions container for init.d scripts is defined(usually **/etc/init.d/functions**), followed by lines setting and ENVVARS and functions for the service, (much like in bashrc does for it's purpose).

For example, the beginning of **/etc/rc.d/init.d/kudzu** has these line common to SysVinit scripts:

```
#!/bin/bash
# kudzu          This scripts runs the kudzu hardware probe.
# chkconfig: 345 05 95
# description:   This runs the hardware probe, and optionally configures \
#                changed hardware.
# Source function library.
. /etc/init.d/functions
```

Says that the script should be started in levels 3, 4, and 5, start priority 5, stop priority 95

/etc/rc.d/init.d directory contents:

/etc/rc.d/init.d/acpid
/etc/rc.d/init.d/anacron

...

/etc/rc.d/init.d/ypbind
/etc/rc.d/init.d/yum-updatesd

Files in the**/etc/rc.d/rc#.d** directories are symlinks to the actual scripts for all of SysVinit's managed programs in **/etc/rc.d/init.d** For example, **/etc/rc.d/rc0.d/K99cpuspeed** links to **/etc/rc.d/init.d/cpuspeed**
With those links, the naming convention of K or S means "kill" or "start" and the number (like 99) indicates the numerical order that it is executed in that runlevel's directory, when that runlevel starts. This way, it is directed that things are stopped and started in the proper order.

As an example, here is a sample of some filenames in **/etc/rc.d/rc3.d/**

K88wpa_supplicant	S02lvm2-monitor
K89netplugd	S04readahead_early
K89rdisc	S05kudzu
K91capi	S08ip6tables
K99readahead_later	S08iptables
S00microcode_ctl	S08mcstrans

rc.local is to execute commands during the startup without needing symlinks. "Local system initialization script"
S99local -> softlink for /etc/rc.local in 2,3,4 and 5 runlevels
You can optionally have a similar shutdown items script in **/etc/rc.d/rc.local_shutdown**
rc.sysinit seems to be redhat specific and is executed very early in the process while rc.local is executed later.
rc is typically not used by linux distributions but is used in BSD

The rc stands for "run commands"; runcom (as in .cshrc or /etc/rc) comes from the runcom facility from the MIT CTSS system, ca. 1965. From Kernighan and Ritchie, as told to Vicki Brown: "There was a facility that would execute a bunch of commands stored in a file; it was called runcom for "run commands", and the file began to be called "a runcom". rc in Unix is a fossil from that usage."
The idea of having the command processing shell be an ordinary slave program came from the Multics design, and a predecessor program on CTSS by Louis Pouzin called RUNCOM. The first time I remember the name "shell" for this function was in a Multics design document by Doug Eastwood (of BTL). Commands that return a value into the command line were called "evaluated commands" in the original Multics shell, which used square brackets where Unix uses backticks.

/etc/inittab Main config file for SysVinit

- Specifies runlevels, scripts to run when certain runlevels are selected, and items to respawn (getty).
- Syntax for an entry in inittab: id:runlevels:action:process.
- First is a unique arbitrary identifier, second indicates what runlevels invoke the command, third is how to handle this entry (like execute command once or respawn whenever it exits, fourth is the command and it's arguments
- x:5:respawn:/etc/X11/prefdm -nodaemon Runlevel 5, specify default login screen for X11
- 3:2345:respawn:/sbin/mingetty tty3 Virtual terminal 3, available for runlevels 2 through 5

Sample /etc/initab (truncated):

```
# Set the default runlevel to three - points to line below "l3:3:wait:/etc/rc.d/rc 3"
id:3:initdefault:

# Execute /etc/rc.d/rc.sysinit when the system boots
# starts network, establishes mounted systems, starts SELinux, encryption
si:S:sysinit:/etc/rc.d/rc.sysinit

# Run /etc/rc.d/rc with the runlevel as an argument - e.g., a 5 points it to /etc/rc5.d/
# Runlevels are designated in /etc/rc.d/
l0:0:wait:/etc/rc.d/rc 0
l1:1:wait:/etc/rc.d/rc 1
...
l5:5:wait:/etc/rc.d/rc 5
l6:6:wait:/etc/rc.d/rc 6

# Executed when we press ctrl-alt-delete
ca::ctrlaltdel:/sbin/shutdown -t3 -rf now

# Start agetty for virtual consoles 1 through 6
c1:12345:respawn:/sbin/agetty 38400 tty1
c2:12345:respawn:/sbin/agetty 38400 tty2
...
c6:45:respawn:/sbin/agetty 38400 tty6
```

Default runlevel is determined, then scripts in appropriate **/etc/rc.d/rcX/** directory are run.
When SysVinit is instructed to change runlevels, it reads initab for what **/etc/rc.d** directory belongs to that runlevel.
The **rc.d** directory contains the daemon scripts which run at boot and when switching runlevels.
Contents in **/etc/rc.d/rcX/** directories are just symlinks to the files in **/etc/rc.d/init.d/** and are named to either start with an S (start) or a K (kill), in order of the number to be processed
For example: take a symlink **S45dhcpd** in in **/etc/rc.d/rc3/** - This means the **/etc/rc.d/init.d/dhcpd** script will be 45th in order to start for that runlevel directory containing this symlink- in this case runlevel 3.

Some types of Linux using SysVinit don't even use this system of symlinks. Slackware uses something similar to BSD where all directives for a runlevel are only put in a runlevel script.

Runlevel Service Management Tools -SysVinit initscript utilities

service <*servicename*> [**start** | **stop** | **restart** | **status** | **list**] Activate, etc., a daemon in current runlevel
OR Go to **/etc/rc.d/init.d/** directory and type ***./<servicename>* start**.
init OR **telinit [0-6]** switches to the specified runlevel
runlevel tells you your runlevel- returns two numbers - 3 5 means that current runlevel is 5 and previous was 3.
chkconfig --list gives you this type of output:

```
[root@localhost rc.d]# chkconfig --list
NetworkManager  0:off   1:off   2:off   3:off   4:off   5:off   6:off
acpid           0:off   1:off   2:on    3:on    4:on    5:on    6:off
anacron         0:off   1:off   2:on    3:on    4:on    5:on    6:off
apmd            0:off   1:off   2:on    3:on    4:on    5:on    6:off
atd             0:off   1:off   2:off   3:on    4:on    5:on    6:off
auditd          0:off   1:off   2:on    3:on    4:on    5:on    6:off
autofs          0:off   1:off   2:off   3:on    4:on    5:on    6:off
avahi-daemon    0:off   1:off   2:off   3:on    4:on    5:on    6:off
```

chkconfig --list <*servicename*> will output the same on one line for that one service
chkconfig --list | grep <*servicename*> might be more helpful to list multiple matches (e.g. "avahi-" services)
chkconfig --level 35 <*servicename*> off | **on** | **reset** | **resetpriorities** - affects service in runlevels 3 and 4
chkconfig --level <*servicename*> on -turns service on in levels 2-5, if 'off' affects 0-6
chkconfig --add OR **--del** adds or removes scripts from **/etc/rc.d/init.d/**
chkconfig --override <*servicename*> - files in **/etc/chkconfig.d/<*servicename*>** can override init service scripts

ntsysv is just a Red Hat TUI interface to turn off and on services in the currently active runlevel. Debian has **rcconf**
ntsysv --runlevel 35 (OR **--level 35**) manages services on levels 3 and 5
redhat-config-services or **system-config-services** - graphical Services Configuration Tool

/sbin/telinit is linked to /sbin/init - takes a one-character argument (0-6 for switching runlevels, s/S/1 all work for single user mode, U/u to resart current runlevel init scrips without checking inittab; Q/q to do so forcing checking inittab. The init binary checks if it is init or telinit by looking at its process id; the real init's process id is always 1.

- ls /etc/init.d/ will also list all of the currently available service files
- Scripts to stop and start processes can be used as an alternative to running **kill**.
- neither ntsysv or chkconfig starts or stops services- only dictates runlevel. The service command does that.
- xinetd services are immediately affected by ntsysv, unlike others

SysVinit Runlevel	***Systemd Target***	***Description***
0	*poweroff.target*	*Halts the system*
1	*rescue.target*	*Single-user mode* (everything mounted, minimal services)
2	*multi-user.target*	*Multiuser mode without networking*
3	*multi-user.target*	*Multiuser mode with networking*
4	*multi-user.target*	*User configurable*
5	*graphical.target*	*Used for the GUI (X11 multiuser mode)*
6	*reboot.target*	*Reboots the system*

systemd's emergency.target
- Is like init=/bin/sh on the kernel command line
- Has no corresponding sysvinit runlevel- would just boot your machine to a shell with really nothing started.
- You get a shell, but almost nothing else (except for systemd in the background)
- No services are started, no mount points mounted, no sockets established.
- Useful for running specific scripts which could then be started independently.
- Allows booting bit-by-bit, starting the various services and other units step-by-step manually.

In SysVinit, services can define arbitrary commands. Examples would be service iptables panic, or service httpd graceful. Native systemd services do not have this ability. Any service that defines an additional command in this way would need to define some other, service-specific, way to accomplish this task when writing a native systemd service definition.Check the package-specific release notes for any services that may have done this.

Example systemd Unit Script Contents

It is *not* advised to edit unit scripts in **/usr/lib/systemd/system/** so they remain default/as-installed by packages. For custom changes, make copy to edit in **/etc/systemd/system** - this directory overrides those defaults for you.

Example Service Unit Script - /usr/lib/systemd/system/httpd.service contents:

```
[Unit]
Description=The Apache HTTP Server
After= network.target remote-fs.target nss-lookup.target

[Service]
Type=notify
EnvironmentFile=/etc/sysconfig/httpd
ExecStart=/usr/sbin/httpd $OPTIONS -DFOREGROUND
ExecReload=/usr/sbin/httpd $OPTIONS -k graceful
ExecStop=/bin/kill -WINCH ${MAINPID}
KillSignal=SIGCONT
PrivateTmp=true

[Install]
WantedBy=multi-user.target
```

Example Target Unit Script - /usr/lib/systemd/system/multi-user.target contents:

```
[Unit]
Description= Multi-User System
Documentation=man:systemd.special(7)
Requires=basic.target
Conflicts=rescue.service rescue.target
After=basic.target rescue.service rescue.target
AllowIsolate=yes
[Install]
Alias=default.target
```

"After" is what is loaded after this finishes activating
"Requires" should be what needs to be loaded before

Example Custom Mount Scripts - /etc/systemd/system/lvdisk.mount and lvdisk.automount

Needs a mount file and automount file with a matching name (mydisk5.automount for mydisk5.mount)
This is the way future versions of RHEL will likely do automount instead of the old /etc/fstab method

```
# vim /etc/systemd/system/lvdisk.mount
[Unit]
Description= Example test mount
[Mount]
what  = /dev/vgdisk/lvdisk
where = /lvdisk
type = xfs
[Install]
WantedBy=multi-user.target
```

```
# vim /etc/systemd/system/lvdisk.automount
[Unit]
Description= Example test automount
[Automount]
where = /lvdisk
[Install]
WantedBy=multi-user.target
```

Test it out - systemctl enable lvdisk.automount; systemctl start lvdisk.automount; mount | grep lvdisk

Contents of the systemd Package

Installed programs: bootctl, busctl, coredumpctl, halt, hostnamectl, init, journalctl, kernel-install, localectl, loginctl, machinectl, networkctl, poweroff, reboot, runlevel, shutdown, systemctl, systemd-analyze, systemd-ask-password, systemd-cat, systemd-cgls, systemd-cgtop, systemd-delta, systemd-detect-virt, systemd-escape, systemd-hwdb, systemd-inhibit, systemd-machine-id-setup, systemd-mount, systemd-notify, systemd-nspawn, systemd-path, systemd-resolve, systemd-run, systemd-socket-activate, systemd-stdio-bridge, systemd-tmpfiles, systemd-tty-ask-password-agent, telinit, timedatectl, and udevadm

Installed libraries: libnss_myhostname.so.2, libnss_mymachines.so.2, libnss_resolve.so.2, libnss_systemd.so.2, libsystemd.so, libsystemd-shared-231.so, and libudev.so

Installed directories: /etc/binfmt.d, /etc/init.d, /etc/kernel, /etc/modules-load.d, /etc/sysctl.d, /etc/systemd, /etc/tmpfiles.d, /etc/udev, /etc/xdg/systemd, /lib/systemd, /lib/udev, /usr/include/systemd, /usr/lib/binfmt.d, /usr/lib/kernel, /usr/lib/modules-load.d, /usr/lib/sysctl.d, /usr/lib/systemd, /usr/lib/tmpfiles.d, /usr/share/doc/systemd-234, /usr/share/factory, /usr/share/systemd, /var/lib/systemd, and /var/log/journal

bootctl	Query the firmware and boot manager settings
busctl	Review logs and monitor the D-Bus bus
coredumpctl	Retrieve coredumps from the systemd Journal
halt	Normally invokes **shutdown** with the *-h* option, except when already in run-level 0, then it tells the kernel to halt the system; it notes in the file /var/log/wtmp that the system is being brought down
hostnamectl	Query and change the system hostname and related settings
init	The first process to be started when the kernel has initialized the hardware which takes over the boot process and starts all the proceses it is instructed to
journalctl	Query the contents of the systemd Journal
kernel-install	Add and remove kernel and initramfs images to and from /boot
localectl	Query and change the system locale and keyboard layout settings
loginctl	Review logs and control the state of the systemd Login Manager
machinectl	Review logs and control the state of the systemd Virtual Machine and Container Registration Manager
networkctl	Review logs and state of the network links as seen by systemd-networkd
poweroff	Tells the kernel to halt the system and switch off the computer (see **halt**)
reboot	Tells the kernel to reboot the system (see **halt**)
runlevel	Reports the previous and the current run-level, as noted in the last run-level record in /var/run/utmp
shutdown	Brings the system down in a secure way, signaling all processes and notifying all logged-in users
systemctl	Review logs and control the state of the systemd system and service manager
systemd-analyze	Determine system boot-up performance of the current boot
systemd-ask-password	Query a system password or passphrase from the user, using a question message specified on the command line
systemd-cat	Connect STDOUT and STDERR of a process with the Journal
systemd-cgls	Recursively shows the contents of the selected Linux control group hierarchy in a tree
systemd-cgtop	Shows the top control groups of the local Linux control group hierarchy, ordered by their CPU, memory and disk I/O load
systemd-delta	Identify and compare configuration files in /etc that override default counterparts in /usr
systemd-detect-virt	Detects execution in a virtualized environment
systemd-escape	Escape strings for inclusion in systemd unit names
systemd-hwdb	Manage hardware database (hwdb)
systemd-inhibit	Execute a program with a shutdown, sleep or idle inhibitor lock taken
systemd-machine-id-setup	Used by system installer tools to initialize the machine ID stored in /etc/machine-id at install time with a randomly generated ID
systemd-mount	A tool to temporarily mount or auto-mount a drive.
systemd-notify	Used by daemon scripts to notify the init system about status changes

systemd-nspawn	Run a command or OS in a light-weight namespace container
systemd-path	Query system and user paths
systemd-resolve	Resolve domain names, IPV4 and IPv6 addresses, DNS resource records, and services
systemd-run	Create and start a transient .service or a .scope unit and run the specified command in it
systemd-socket-activate	A tool to listen on socket devices and launch a process upon connection.
systemd-tmpfiles	Creates, deletes and cleans up volatile and temporary files and directories, based on the configuration file format and location specified in tmpfiles.d directories
systemd-tty-ask-password-agent	Used to list or process pending systemd password requests
telinit	Tells **init** which run-level to change to
timedatectl	Query and change the system clock and its settings
udevadm	Generic Udev administration tool: controls the udevd daemon, provides info from the Udev database, monitors uevents, waits for uevents to finish, tests Udev configuration, and triggers uevents for a given device
libsystemd	systemd utility library
libudev	A library to access Udev device information

-- from http://www.linuxfromscratch.org/lfs/view/systemd/chapter06/systemd.html

You may have noticed that installed items listed contains telinit, and an /etc/init.d directory. According to the Fedora Wiki, the 'service' and 'chkconfig' commands will (surprisingly) mostly continue to work as expected in the systemd world. Presumably, this would be for backward compatibility support for old scripts, etc.

Target unit directories hold symlinks to the real unit files like this:	**/etc/systemd/system/XXXXXX.target.wants/bluetooth.service**
Those symlinks point to the actual service (etc) unit files that reside here:	**/usr/lib/systemd/system/bluetooth.service**
The **default.target** file here is a target/runlevel symlink:	**/etc/systemd/system/default.target**
And the default.target symlink points to the actual target here:	**/usr/lib/systemd/system/XXXXXX.target**

Running **systemctl isolate graphical.target** will not affect the default.target symlink, and merely switches the current runlevel (use **set-default**).

Running **systemctl disable myservice** basically does the same as **rm '/etc/systemd/system/multi-user.target.wants/service.myservice'**
Running **systemctl enable myservice** basically does the same as
ln -s '/usr/lib/systemd/system/myservice.service' '/etc/systemd/system/multi-user.target.wants/service.myservice'

The "**target.wants**" directories in **/usr/lib/systemd/system/** hold symlinks to the corresponding runlevel's unit files just like init's **/etc/rc.d/rc#.d/**
A target is itself a unit file, manages other unit files. Defaults are multi-user.target, graphical.target, rescue.target, emergency.target, poweroff.target, and reboot.target

The ifconfig Command

This command is deprecated in Linux, but it is important to keep sharp with when working with Solaris, AIX, HP-UX, BSD, and SCO UNIX.

- With no arguments will display details all the active interfaces, use **-a** to also list inactive ones
- You can specify the interface to view, like **ifconfig eth0**
- Bring interfaces up or down with **ifconfig eth0 up** (or **down**) or **ifup eth0** - (or **ifdown**)
- (note- routes to interfaces disabled with ifconfig and ifdown are not automatically disabled)

Changes with ifconfig are NOT persistent after reboot
In BSD you may have to edit /etc/rc.conf appropriately
In Linux edit appropriate interface file in /etc/sysconfig/network-scripts/

When you specify an interface, you can add these to change it's properties:
- Set an ip address with the syntax **ifconfig eth0 172.16.25.125**
- Just like setting the address you can add **netmask 255.255.255.224** and/or **broadcast 172.16.25.63**
- Set an mtu with **mtu 1000**
- Change a MAC address with the "hw ether" argument - **ifconfig eth0 hw ether 00:BB:22:DD:44:FF**
- The states broadcast, multicast, and allmulti can all be turned off and on like promisc and -promisc
- When considering packet capture, enter promiscuous mode with **promisc** and disable with **-promisc**
- Similarly you can disable ARP with **-arp** and re-enable it with **arp**
- Adding an alias for an interface is possible, but it's ip address has rules on masks (see ifconfig addendum)
  ifconfig eth0:0 172.16.25.127 - to disable the alias simply bring it down **ifconfig eth0:0 down**
  note: some versions of ifconfig require the use of the keyword **alias**

delete - Removes the specified *network address*. This is used when an alias is incorrectly specified or when it is no longer needed. Incorrectly setting an ns address has the side effect of specifying the host portion of the network address. Removing all ns addresses allows you to re-specify the host portion.

detach - Removes an *interface* from the network interface list. If the last interface is detached, the network interface driver code is unloaded. In order for the interface route of an attached interface to be changed, that interface must be detached and added again with ifconfig.

Other options:

group ID and **-group ID**
Adds/removes a group ID to the group ID list for the interface (used in determining the route to use when forwarding packets that arrived on the interface.

metric <*value*> - Sets the routing metric of the interface to the value specified by the Number variable. The default is 0 (zero). The routing metric is used by the routing protocol (the routed daemon). Higher metrics have the effect of making a route less favorable. Metrics are counted as addition hops to the destination network or host.

monitor and **-monitor**
Enables/disables the underlying adapter to notify the interface layer of link status changes. The adapter must support link status callback notification. If multipath routing is used, alternate routes are selected when a link goes down.

checksum_offload and **-checksum_offload**
Enables/disables the flag to indicate that transmit TCP checksum should be offloaded to the adapter. Also resets the per-interface counter that determines whether TCP should dynamically enable or disable offloading of checksum computation.

The ifconfig command in Linux is part of the net-tools package, which has been deprecated (and not updated since April 2001. Other commands worked with ifconfig, like ifdown, ifup, iwconfig, arp, route, iptunnel, ipmaddr, tunctl, brctl, etc. and even netstat were generally replaced with ip, ss, and more in the iproute2 package, which here has a separate section devoted to it.

netstat

- gets values from /proc/net*
- often generically used as **netstat -netulp** or **-tulpen** (same thing, different mnemonic)
- netstat's successor **ss** (for "socket stats") uses almost identical command options (less to remember)

-a list all ports
-t for tcp
-u for udp
-l listening
-p includes a PID/Program name field
-s statistics for each protocol (separate listings for each protocol udp/tcp/icmp/etc in one listing)
-n numeric only - says don't use hostnames, port names, usernames. **-N** does the opposite (-r in the ss command)
To apply the **-n** option more specific, instead use **--numeric-ports**, **--numeric-hosts**, or **--numeric-users**
-e for extended on certain other subcommands
-c for continuous output, sort of like using watch command
-r displays the routing table
-i displays a list of all network interfaces (kernel interface table)
-ie displays interfaces with extended option, which looks like ifconfig output
--verbose includes info like " netstat: no support for `AF IPX' on this system"
-g will display the multicast group information
-M or **--masquerade** to show NAT info
-x lists legacy UNIX process sockets that are listed as their own protocol
-w to list packets of the type raw

As usual pipe out to other tools. Use grep to search for specific items i.e., **netstat -ap | grep ssh** or use **grep "ESTAB"** - Continuous list of active tcp connections: **watch -d -n0 "netstat -atnp | grep ESTA"**
netstat -l | grep 1000 | wc -l - to see if there is traffic on a certain port

Some options in netstat vary slightly among different versions of UNIX- check your man page

Migrating from the traditional net-tools package to iproute2 package

The net-tools package, which included ifconfig and many other tools, was slated for deprecation over a decade ago, and finally started getting dropped around 2009. This applies to most distributions of Linux- if you are using BSD, Solaris, or another version of UNIX, you will want to stick to traditional net-tools package and ifconfig. On Linux, you can still install it, but there are things it won't work well with modern Linux networking

Summary of changes to be aware of that I'll try to cover here:
- *The net-tools package replacements provided by iproute2*
- *Network Monitor (nmcli) to manage persistent configuration changes*
- *The abandoning of udev naming schemes for devices; a return to hardware-specific names*

Deprecated	*Replacement command(s)*
arp	ip n (ip neighbor)
ifconfig	ip a (ip addr), ip link, ip -s (ip -stats)
iptunnel	ip tunnel
iwconfig	iw
nameif	ip link, ifrename
netstat	ss, ip route (for netstat-r), ip -s link (for netstat -i), ip maddr (for netstat-g)
route	ip r (ip route)

For an exhaustive comparison of these, please see Doug Vitale's blog entry at https://dougvitale.wordpress.com/2011/12/21/deprecated-linux-networking-commands-and-their-replacements/

The ip command

ip address (or addr or a) [add | del | set] dev [interface]
    Address, IPv4 or v6, on the interface
ip link (or l) [set | show] dev [interface]
    Link generally refers to interfaces/ devices
ip route (or r) [add | chg | repl | del | show] cache
    Route mostly replaces route commands
ip neighbor (or n) [add | chg | repl | del | show] dev [interface]
    Neighbor mostly replaces arp commands and also shows IPv6 NDP info

The ip command has a HUGE list of other "objects" like link, route, neighbor, and address that can also be queried or configured. Explained more later, here is a list: addrlabel, rule, ntable, tunnel, tuntap, l2tp,maddr, mroute, mrule, monitor, xfrm, netns, tcp_metrics. The link object also supports a huge list of interface types with their own help pages.

Using the ip command applies settings but will not save the configuration - it's not persistent. In order to make persistent changes, either use the Network Manager package or edit the network scripts directly

Device naming:
Traditionally, we have seen devices use udev naming like eth0 or usb0. udev provides persistent naming for some device types out of the box to make things more human-readable (like hard drives- /dev/sdb1, /dev/hda2). It has been used so long some people don't know or forgot it was an add-on.

BIOS naming based on HW properties (physical naming) has returned, and here is what you will see more of:
- em[1-N] for embedded NICs
- p[slot-number]p[port-number] - p6p1 = pci slot 6 port 1

You might also see logical naming with VLAN and alias naming. You may even see a udev name being used.

If you prefer not to, you can add to your boot options in GRUB:
Add "net iframes=0 biosdevnames=0" to boot options
Then write to grub config on disk after boot : grub2-mkconfig -o /boot/grub2/grub.cfg

Examples of tasks - iproute2 and net-tools package equivalents

Show All Connected Network Interfaces
With net-tools: $ ifconfig -a
With iproute2: $ ip link show
See also: "ip addr" for "ifconfig" and "ip -s link" for "netstat -i"

Show IPv4 Address(es) of a Network Interface
With net-tools: $ ifconfig eth1
With iproute2: $ ip addr show dev eth1

Show IPv6 address(es) of a Network Interface
With net-tools: $ ifconfig eth1
With iproute2: $ ip -6 addr show dev eth1

View the IP Routing Table
With net-tools: $ route -n --or-- $ netstat -rn
With iproute2: $ ip route show

View Socket Statistics
With net-tools: $ netstat --AND-- $ netstat -l
With iproute2: $ ss --AND-- $ ss -l

View the ARP Table
With net-tools: $ arp -an
With iproute2: $ ip neigh

Activate or Deactivate a Network Interface
With net-tools: $ ifconfig eth1 [up | down]
With iproute2: $ ip link set [up | down] eth1

Assign IPv4 address(es) to a Network Interface
With net-tools: $ ifconfig eth1 10.0.0.1/24
With iproute2: $ ip addr add 10.0.0.1/24 dev eth1

Remove an IPv4 address from a Network Interface
In net-tools you end up assigning 0 to the interface. iproute2 can properly remove it.
With net-tools: $ ifconfig eth1 0
With iproute2: $ ip addr del 10.0.0.1/24 dev eth1

Assign or Remove an IPv6 address on a Network Interface
With net-tools: $ ifconfig eth1 inet6 [add | del] 2002:0db5:0:f102::1/64
With iproute2: $ ip -6 addr [add | del] 2002:0db5:0:f102::1/64 dev eth1

Assign Multiple IP Addresses to an Interface
With net-tools (ip subinterface aliases workaround):
$ ifconfig eth0:1 192.168.10.10 netmask 255.255.255.0 up
$ ifconfig eth0:2 192.168.10.15 netmask 255.255.255.0 up
With iproute2:
$ ip addr add 10.0.0.1/24 dev eth1
$ ip addr add 10.0.0.2/24 dev eth1

Change the MAC Address of a Network Interface
Before changing the MAC address, you need to deactivate the interface first.
With net-tools: $ ifconfig eth0 [down | up]; ifconfig eth1 hw ether 08:00:27:75:2a:66
With iproute2: $ ip link set dev eth0 [down | up]
$ ip link set dev eth1 address 08:00:27:75:2a:67

Add or Modify a Default Route
With net-tools: $ route [add | del] default gw 192.168.1.2 eth0
With iproute2: $ ip route [add | replace] default via 192.168.1.2 dev eth0 *(replace is a command)*

Add or Remove a Static Route
With net-tools: $ route add -net 172.16.32.0/24 gw 192.168.1.1 dev eth0
$ route del -net 172.16.32.0/24
With iproute2: $ ip route add 172.16.32.0/24 via 192.168.1.1 dev eth0
$ ip route del 172.16.32.0/24

Add or Remove a Static ARP Entry
With net-tools: $ arp -s 192.168.1.100 00:0c:29:c0:5a:ef
$ arp -d 192.168.1.100
With iproute2: $ ip neigh add 192.168.1.100 --OR-- ip addr 00:0c:29:c0:5a:ef dev eth0
$ ip neigh del 192.168.1.100 dev eth0

Add, Remove or View Multicast Addresses
With net-tools: $ ipmaddr [add | del] 33:44:00:00:00:01 dev eth0
$ ipmaddr show dev eth0 --OR-- $ netstat -g
With iproute2: $ ip maddr [add | del] 33:44:00:00:00:01 dev eth0
$ ip maddr list dev eth0

NetworkManager Service - Persistent Changes with nmcli

In order to make persistent changes, you should either use Network Manager, or manually edit the files it uses When running NM, manually editing those files is not recommended to do unless you have to, such as in a script or something, but saying that isn't a way of babysitting us- Network Manager often clobbers what we put in manually with it's own info, so telling it through it's own mechanisms can avoid that.
[Unsurprisingly, Network Manager disgusts a lot of sysadmins for being so resistant to manual edits]

Network Manager has a GUI, and a text interface quite similar to it. It can actually be effective for doing a good range of tasks, but (as usual) the command line is much more flexible and granular. Usually, we get things done by entering individual commands, but it also has a command prompt of it's own for advanced operations.

nmcli [OPTIONS] OBJECT { COMMAND | help }
When using nmcli, the most important component in the command definition above is "object."
Connections and devices are the most often used object components.
- a device is a network interface
- a connection is a collection of configurations (e.g., home, work configs with different settings for everything)
- so, you can have multiple connections for a device but only one can be active at one time

The general options pertain to output styles, facilitating use by external scripts, etc. These include -t [erse] or -p [retty] for ease of viewing, -m [mode] tabular | multiline, -f [ields] <field1, field2, ...> if you only want to output some columns, etc.

The "device" object is how you refer to specific devices, like your wireless or ethernet interfaces. They are what you are going to add to your various "connection" objects
Common device commands: status, show, connect, set, reapply, disconnect, delete, monitor, wifi, and lldp.
Generally those commands would be followed by the interface name. The set command allows setting autoconnect and/or managed to on or off.
Wifi devices have more specific direcives as illustrated in this excerpt:

```
  wifi [list [ifname <ifname>] [bssid <BSSID>]]
  wifi connect <(B)SSID> [password <password>] [wep-key-type key|phrase] [ifname <ifname>]
                         [bssid <BSSID>] [name <name>] [private yes|no] [hidden yes|no]
  wifi hotspot [ifname <ifname>] [con-name <name>] [ssid <SSID>] [band a|bg] [channel <#>] [password
<password>]
  wifi rescan [ifname <ifname>] [[ssid <SSID to scan>] ...]
```

The nmcli "connection" object has a variety of common commands: show, up, down, add, modify, clone, edit, delete, monitor, reload, load, import and export

The "general" object has 4 commands: status, hostname, permissions, and logging
Status comes up whenever you call nmcli general by itself:

```
[user@localhost ~]$ nmcli g
STATE      CONNECTIVITY  WIFI-HW  WIFI     WWAN-HW  WWAN
connected  full          enabled  enabled  enabled  enabled
```

Typing "nmcli general hostname" outputs your hostname, and if you give put one at the end it sets it to that.
Typing "nmcli general logging" outputs or changes the logging level and domains the same way..
Typing "nmcli general permissions" outputs "caller permissions for authenticated operations," as seen below.

```
[user@localhost ~]$ nmcli g permissions
PERMISSION                                        VALUE
org.freedesktop.NetworkManager.enable-disable-network      yes
org.freedesktop.NetworkManager.enable-disable-wifi         yes
org.freedesktop.NetworkManager.enable-disable-wwan         yes
org.freedesktop.NetworkManager.enable-disable-wimax        yes
org.freedesktop.NetworkManager.sleep-wake                  no
org.freedesktop.NetworkManager.network-control             yes
org.freedesktop.NetworkManager.wifi.share.protected        yes
org.freedesktop.NetworkManager.wifi.share.open             yes
org.freedesktop.NetworkManager.settings.modify.system      yes
org.freedesktop.NetworkManager.settings.modify.own         yes
org.freedesktop.NetworkManager.settings.modify.hostname    auth
org.freedesktop.NetworkManager.settings.modify.global-dns  unknown
org.freedesktop.NetworkManager.reload                      unknown
```

The "networking" object is so succinct it is almost disappointing. It merely lets you turn networking on or off with "nmcli networking [on | off]," and lets you query "nmcli net connectivity" and it reports "full" if it's working ok.

The "radio" object also doesn't do a lot. Like the networking object, most of the controls are over in the "device" object. Just by itself, it will give the output below, with WWAN referring to mobile network service and interface. You can turn things off with "nmcli radio [wlan | wan | all] [on | off]"

```
[user@localhost ~]$ nmcli radio
WIFI-HW  WIFI     WWAN-HW  WWAN
enabled  enabled  enabled  disabled
```

The "agent" object allows you to use policy management like polkit to govern permissions on things like turning the network on or off. Mechanisms like polkit are outside of the scope of this document, but that's what the "agents" object enables.

Finally, typing "nmcli monitor" simply turns on (or off) a facility that prints a line to stdout when something in Network Manager changes.

General Use and Examples

So, Network manager and it's NMCLI is when you need persistent configuration solutions, even if you are just testing things out. The ip command is for when you need things done at the moment quick, just don't care if something is going to stick after a reboot or possibly get lost after you log out- or, maybe you just want some information in a different format.

Quick NMCLI examples

As you can see below, most nmcli objects and commands can be truncated down a lot,
nmcli con show - *Find different connections for different devices*
nmcli con show eno1 - *Get details on eno1*
nmcli dev status - *Get status of all devices*
--------We know we have an ethernet device object called "eno1" so let's do stuff with it:
nmcli con add con-name dhcp type ethernet ifname eno1
nmcli con add con-name static ifname eno1 autoconnect no type ethernet ip4 192.168.122.102 gw4 192.168.122.1
nmcli dev status - *Find out which is used*
nmcli con up static; nmcli con up dhcp - *Bring these connections up*
nmcli con show static - *Get this connection's status*
--------So this made two connection objects (static and dhcp) pointing to the device object en01
--------Here, we are going to add more information and a second IP address to the "static" connection object:
nmcli con mod static ipv4.dns 192.168.122.1 --- to specify a dns server
nmcli con mod static +ipv4.dns 8.8.8.8 --- to add a dns server (+ is needed if one has already been defined)
nmcli con mod static +ipv4.addresses "192.168.100.10/24 192.168.100.1" -- modify IP and gateway
nmcli con mod static +ipv4.addresses 10.0.0.10/24 -- add a secondary IP addy
--------All of this writes the settings but doesn't activate them - you have to reload the connection for them to work
nmcli con reload - *Re-reads the config file if you can't take down conn and bring it back up*

So, a few interesting things about this example.
- Note that it uses dotted notation to add the ip4 properties addresses and dns. These are listed in the man page for nm-settings(5). [http://manpages.ubuntu.com/manpages/zesty/man5/nm-settings.5.html]
- If you do not specify a connection name when creating it, one is auto-generated as "type-ifname[-number"

The nmcli Prompt

When you choose the "edit" option on an object, you get the nmcli prompt, and can issue directives that way instead. To turn the connection “net-eth1” to DHCP (auto) instead of static:

```
[user@localhost ~]$  nmcli con edit net-eth1
> print all
> remove ipv4.gateway
> remove ipv4.address
> set ipv4.method auto
> set ipv4.dns 8.8.8.8 8.8.4.4
> verify all
> save persistent
> quit
```

Getting Rid of Network Manager

To disable Network Manager on a systemd system:
    $ sudo systemctl stop NetworkManager.service AND systemctl disable NetworkManager.service
On a systemVinit system:
    $ sudo service NetworkManager stop AND chkconfig NetworkManager off
In Debian 7 or earlier:
    $ sudo /etc/init.d/network-manager stop AND update-rc.d network-manager remove
In Ubuntu or Linux Mint:
    $ sudo stop network-manager AND echo "manual" | sudo tee /etc/init/network-manager.override
Slackware:
    $ /etc/rc.d/rc.networkmanager stop AND chmod a-x /etc/rc.d/rc.networkmanager

In some versions of NM, issuing "stop" may also kill dhcp and wpa_supplicant, so be sure to check.
After disabling Network Manager on Debian or Ubuntu, use /etc/network/interfaces to configure network interfaces.
After disabling Network Manager on Fedora or CentOS, use /etc/sysconfig/network-scripts/ifcfg-ethX files to configure network interfaces.

To disable Network Manager only for eth1 (for example)
Network Manager automatically ignores any interfaces specified in the file /etc/network/interfaces (Debian/Ubuntu), or the proper config file inside the directory /etc/sysconfig/network-scripts/ (RHEL/CentOS/Fedora)
Let's say your interface is eth0.

 - For RHEL-compatible, make a file for your interface in /etc/sysconfig/network-scripts/ifcfg-eth0
DEVICE="eth0"
NM_CONTROLLED="no" # this is most important
ONBOOT=yes
HWADDR=A4:BA:DB:37:F1:04
TYPE=Ethernet
BOOTPROTO=static
NAME="System eth0"
UUID=5fb06bd0-0bb0-7ffb-45f1-d6edd65f3e03
IPADDR=192.168.1.44
NETMASK=255.255.255.0
# Optionally put these in or add them to this file: **# vi /etc/sysconfig/network**
NETWORKING=yes
HOSTNAME=centos6
GATEWAY=192.168.1.1
# Same thing with this - or add into in resolv.conf **#vi /etc/resolv.conf**
nameserver 8.8.8.8 # Replace with your nameserver ip
nameserver 192.168.1.1 # Replace with your nameserver ip
DNS1=8.8.8.8 # another optional format if it works better for you
DNS2=8.8.4.4
# The reason these are optional is this is how you would specify per-interface file
# different default gateways and DNS if you had too. Not always guaranteed to work but there it is.

 - For Debian/Ubuntu - In /etc/network/interfaces, add information about the interface you want to disable NM on
$ sudo vi /etc/network/interfaces
# Find/add your eth0 entry to disable Network Manager
allow-hotplug eth0
iface eth0 inet static
address 10.0.0.10
netmask 255.255.255.0
gateway 10.0.0.1
dns-nameservers 8.8.8.8

For this to work you need to ensure the network service will bring up eth1 upon boot (since NM isn't doing it)
On systemd systems run: $ sudo systemctl enable network.service
On SysVinit systems run: $ sudo chkconfig network on

Upon rebooting, verify that Network Manager is successfully disabled for eth0 with nmcli command.

systemd-networkd

For some time lacked features offered by NetworkManager (check the version you have). Predictably integrated with the rest of systemd (e.g., resolved for DNS, timesyncd for NTP, udevd for naming), and of course shares the rejection of many sysadmins who despise systemd. The command networkctl to show what networkd sees. It features subcommands list, status, and lldp to display info - query a specific device (such as ens128) or --all

To switch from Network Manager to systemd-networkd run:
$ sudo systemctl disable NetworkManager --AND-- sudo systemctl enable systemd-networkd

You also need to enable systemd-resolved service
$ sudo systemctl enable systemd-resolved --AND-- $ sudo systemctl start systemd-resolved
This daemon will create its own resolv.conf - but many programs still look to /etc/resolv.conf, so it is recommended to create a symlink to /etc/resolv.conf
$ sudo rm /etc/resolv.conf --AND-- $ sudo ln -s /run/systemd/resolve/resolv.conf /etc/resolv.conf

To configure network devices you specify configuration information in text files named *.network - to be stored and loaded from /etc/systemd/network. Use **networkctl list** to see available devices on the system.
$ sudo mkdir /etc/systemd/network

To configure DHCP networking (below "yes" can be "ipv4"):
$ sudo vi /etc/systemd/network/20-dhcp.network
[Match]
Name=enp3*
[Network]
DHCP=yes

[Match] obviously says which network device(s) are configured- this matches any interface whose name *starts with* ens3. For static IP on enp3o2 the network block would contain the following with name= enp3o2. Processed in lexical order - a file named 10-static.network, would take precedence over 20-dhcp.network and retain a static IP.
[Network]
Address=192.168.10.50/24
Gateway=192.168.10.1
DNS=8.8.8.8

Wireless interfaces don't have any special differences in the [match] and [network] fields, but they need configuration from another service (wpa_supplicant). An (example) device named wlp2s0, the corresponding systemd service file to enable would be wpa_supplicant@wlp2s0.service, with the configuration file /etc/wpa_supplicant/wpa_supplicant-wlp2s0.conf. If that file doesn't exist, the service won't start.
When you are done, restart networkd to make the changes take effect - $ sudo systemctl restart systemd-networkd

Virtual Network Devices (bridges, VLANs, tunnel, VXLAN, bonding, etc)
These files have the naming *.netdev (rather than *.network). Here is a bridge (br0) with physical interface (eth1):
Create the bridge file: $ sudo vi /etc/systemd/network/bridge-br0.netdev
[NetDev]
Name=br0
Kind=bridge
Eth1 slave config file named *.network as before $ vi /etc/systemd/network/bridge-br0-slave.network
[Match]
Name=eth1
[Network]
Bridge=br0
The *.netdev file declared a bridge - config with a *.network file $ vi /etc/systemd/network/bridge-br0.network
[Match]
Name=br0
[Network]
Address=192.168.10.100/24
Gateway=192.168.10.1
DNS=8.8.8.8
All done, do restart systemd-networkd: $ systemctl restart systemd-networkd
You can use brctl tool to verify that a bridge br0 has been created.

Using iw and wpa_supplicant

Replacing iwconfig with iw

Action	iwconfig (outdated)	iw replacement
Getting info on wlan0	iwconfig wlan0	iw dev wlan0 link
Connecting	iwconfig wlan0 essid foo	iw wlan0 connect foo
Set channel	iwconfig wlan0 essid foo freq 2432M	iw wlan0 connect foo 2432
WEP	iwconfig wlan0 essid foo key s:abcde	iw wlan0 connect foo keys 0:abcde
Join ad-hoc ibss	iwconfig wlan0 mode ad-hoc iwconfig wlan0 essid foo-adhoc	iw wlan0 set type ibss iw wlan0 ibss join foo-adhoc 2412
Leave ad-hoc ibss	iwconfig wlan0 essid off (sometimes worked)	iw wlan0 ibss leave (always works)

For WPA/WPA2 encryption, you should use wpa_supplicant.

Managing connections with wpa_supplicant / wpa-cli

1. Run **ip a** to get name of the wireless interface. If not showing, the driver might need installing
2. Create a file in /etc/wpa_supplicant named *.conf containing this basic configuration line:
   ctrl_interface=DIR=/run/wpa_supplicant GROUP=wheel update_config=1
   GROUP specifies which groups can manage wpa_supplicant, and leaving blank means only root.
3. Initialize by running **wpa_supplicant -B -i w1linksys7 -c /etc/wpa_supplicant/example.conf**
   -B means run in background, -i specifies interface, and -c points to config file
4. Running **wpa_cli** gives an interactive prompt.

wpa-cli commands

scan - will run a scan

scan_results - will dump the scan results of available networks including ssid, security mode, and bssid/ MAC

add_network - to specify a network - provide ssid listed in scan, the key. Number given at the beginning is arbitrary

```
> add_network
0
> set_network 0 ssid "LOCAL_WIFI"
> set_network 0 psk "passcode"
> enable_network 0     - will attempt to associate with the network just configured
> save_config
```

Running **ip a** should then show new IP info

After running save_config, the following will be appended to your configuration file:

```
network={
        ssid="LOCAL_WIFI"
        psk="passcode"
}
```

Now, just running **wpa_supplicant -B -i w1linksys7 -c /etc/wpa_supplicant/example.conf** will connect.

Notes on using Kismet

Most things are self-explanatory with Kismet so there isn't much to cover. When running, typing "h" gives help screen with most info for current screen. In the network panel, W is WEP (yes, none, other); <no_ssid> means the AP isn't broadcasting it's ssid, T is type: P (probe request- no associated connection); A (access point); H (ad-hoc); T (turbocell aka Karlnet or Lucent); G (group); D (data-only network with no control packets).

Flags field includes F (AP using factory default settings/ not configured); T#, U#, A#, D mean an address range of # octets found via type of traffic, being TCP, UDP, ARP, or DHCP respectively

APs listed are color-coded as follows: yellow: unencrypted network; red: factory default settings in use; green: secure networks (WEP, WPA etc..); and blue means SSID cloaking on / SSID not broadcast

The kismet layout can be modified in **/etc/kismet/kismet_ui.conf**

Helpful key functions: type "c" to see clients on an AP, "i" for detailed info on an AP, "r" can show a stats graph, "a" for general stats on all APs, "w" to show all alerts that have come up in the status window

The program LinSSID is a Linux alternative to inSSIDer -- https://sourceforge.net/projects/linssid/

WiFi Pentesting Tools

The details of these tools can be found online or in man pages,

The Aircrack-ng Package - https://www.aircrack-ng.org

Aircrack-ng is the granddaddy of all wireless CLI suites, and has added a lot since I was first using it in 2005-6

- Monitoring: Packet capture and export of data to text files for further processing by third party tools.
- Attacking: Replay attacks, deauthentication, fake access points and others via packet injection.

airbase-ng - Configure fake access points
aircrack-ng - Wireless password cracker
airdecap-ng - Decrypt WEP/WPA/WPA2 capture files
airdecloak-ng - Removes wep cloaking from a pcap file
airdriver-ng - Provides status information about the wireless drivers on your system
aireplay-ng - Primary function is to generate traffic for the later use in aircrack-ng
airmon-ng and airmon-zc - This script can be used to enable monitor mode on wireless interfaces
airodump-ng - Used for packet capturing of raw 802.11 frames
airodump-ng-oui-update - Downloads and parses IEEE OUI list
airolib-ng - Designed to store and manage essid and password lists
airserv-ng - A wireless card server
airtun-ng - Virtual tunnel interface creator
besside-ng - Automatically crack WEP & WPA network
easside-ng - An auto-magic tool which allows you to communicate via an WEP-encrypted access point
buddy-ng - echoes back decrypted packets to the system running easside-ng in order to access the wireless network without knowing the WEP key
ivstools - This tool handles .ivs files. You can either merge or convert them.
makeivs-ng - Generates initialization vectors
packetforge-ng - Create encrypted packets that can subsequently be used for injection
tkiptun-ng - This tool is able to inject a few frames into a WPA TKIP network with QoS
wesside-ng - Auto-magic tool which incorporates a number of techniques to seamlessly obtain a WEP key

Typical WPA-PSK cracking involves taking down the wireless driver, bringing it back up in monitor mode with airmon-ng, firing up airodump-ng to capture packets, and using aireplay-ng to inject deauthentication packets at a client, so that the four-way handshake can be captured when it attempts to reathenticate with the AP. You then run aircrack-ng to crack the pre-shared key in the pcap against a dictionary file. Chances are slim if it won't match in a dictionary.

The WifiTap Package

- http://sid.rstack.org/static/articles/w/i/f/Wifitap_EN_9613.html and https://github.com/gdssecurity/wifitap/

- - traffic capture and injection over a WiFi network by configuring interface **wj0**

Includes wifiarp, wifidns, wifiping, wifitap

- set an IP address consistent with target network address range and route desired traffic through it
- arbitrary packet injection without specific library.
- bypass inter-client communications prevention systems (e.g. Cisco PSPF), reach SSIDs handled by AP

wifitap - WiFi injection tool through tun/tap device
wifiarp - WiFi injection ARP answering tool based on Wifitap
wifidns - WiFi injection DNS answering tool based on Wifitap
wifiping - WiFi injection based answering tool based on Wifitap

wifite - attack multiple WEP, WPA - made to be automated, crack passwords later, grab as much from APs with strongest signal strength so you can come get the gathered stuff and work with it later

Fern Wifi Cracker - a GUI offering the following which isn't limited to wireless attacks. They advertise:

WEP cracking, WPA/WPA2 Cracking with wordlist or WPS based attacks
Automatic AP attacks possible
Session hijacking (Passive and Ethernet Modes)
Internal MITM engine, bruteforce attacks (HTTP, HTTPS, TELNET, FTP)

cowpatty - This is strictly for WPA-PSK - needs aircrack-ng to grab things- provide with a wordlist and captured hash- it generates hashes from wordlist using SSID as seed. Includes genpmk that can precompute hashes

reaver, bully, pixiewps - Bully is faster, more effective for WPS attacks. Reaver was released as a proof of concept back when WPS attack was discovered. On the other hand pixiewps does an offline attack that is super-fast.

Netfilter/ iptables

service iptables [start | save | stop]
/sbin/iptables-save > filename ---- saves rules to STDOUT by default, so send to file
/sbin/iptables-restore < filename ---- restores rules from STDIN by default, so give it the file
/sbin/iptables ----primary ACL modifier utility
- Packet-processing is done top-down by chain order (so are rules in a chain). Duplicate rules can reside in the same chain.
- Rules that are more likely to be matched should be put in the chain higher up in the list than others (line numbers)
- Modules to proxy or function with other OSI layers: [/usr/lib/iptables/*.so]
- Check if enabled: [grep -i config_netfilter /boot/config*] -- you'll find CONFIG_NETFILTER=y
- /etc/sysconfig/iptables is where rules are stored
- Opening /etc/sysconfig/iptables-config:
    IPTABLES_SAVE_ON_RESTART and ON_STOP set to "yes" to save written and implemented rules.
    IPTABLES_SAVE_COUNTER turn on to keep packet counters going from where they left off after a stop or restart

iptables -t table <chain><action/direction><packet pattern><segment pattern> -j <fate>

Table:	filter (default), NAT (change IP addresses/ports), mangle (alter packets/segments TOS/TTL, etc)
Rule handling actions:	-A (append) -D (delete) -L (list) -F (flush) -I (insert) -R (replace) -N (new) -E (rename)
Chain:	INPUT, OUTPUT, FORWARD; PREROUTING and POSTROUTING
Interface:	-i [eth0 | eth1 | eth+] "+" is always wildcard, use ! for negation. Use -o for output interface
L3 Packet Pattern:	-s ip-addr (source), -d ip-addr (destination), "+" is wildcard
L4 Segment Pattern:	-p [tcp | udp | icmp] ; -dport, -sport ---- port #, or any name in /etc/services
State/ statefullness:	-m for matching; -m state --state NEW(syn), ESTABLISHED(syn-ack), RELATED, INVALID
fate (destination):	DROP, ACCEPT, REJECT/DENY, REDIRECT (NAT prerouting chain- local ports), LOG (syslog)

When applicable, -v or --verbose, --line-numbers use with -L list, enumerate lines (for insert and replace operations)

4 default tables (can't remove) and their chains in processing order:

- Filter (default- inbound and outbound traffic rules)	INPUT, FORWARD, OUTPUT
- NAT (change IP addresses, ports)	PREROUTING, OUTPUT and POSTROUTING
- Mangle (packet alteration)	PREROUTING, INPUT, FORWARD, (OUTPUT) and POSTROUTING
- Raw (special- rarely used)	PREROUTING, OUTPUT

The **NAT** table, consulted when a packet that creates a new connection is seen
- PREROUTING (for altering packets as soon as they come in),
- OUTPUT (for altering locally-generated packets before routing)
- POSTROUTING (for altering packets as they are about to go out)

The **mangle** table, used for specialized packet alteration.
- PREROUTING (for altering incoming packets before routing)
- INPUT (for packets coming into the box itself)
- FORWARD (for altering packets being routed through the box)
- OUTPUT (not recommended to use)
- POSTROUTING (for altering packets as they are about to go out).

The **raw** table, used mainly for configuring exemptions from connection tracking along with the NOTRACK target.
- It registers at the netfilter hooks with higher priority and is thus called before ip_conntrack, or any other IP tables.
- Has built-in chains: PREROUTING (packets arriving, any interface) OUTPUT (packets generated by local processes)

Permit SSH	iptables -A INPUT -p tcp --dport 22 -j ACCEPT
Deny telnet	iptables -A INPUT -p tcp --dport telnet -j DROP
Block all traffic NOT from specified IP	iptables -A INPUT -s ! 192.168.1.72 -j DROP
Drop all inbound traffic	iptables -A INPUT -j DROP

Inserts this as rule #1 in the INPUT chain	iptables -I INPUT 1 -p tcp --dport 22 -j DROP
Delete rule #1 in INPUT chain	iptables -D INPUT 1
Deletes first match of this	iptables -D INPUT -p tcp --dport telnet -j DROP
Replace rule 1 with this rule	iptables -R INPUT 1 -p tcp --dport 22 -j ACCEPT
Z option zeros out byte count for chain	iptables -Z INPUT
Zero byte count in PREROUTING chain's mangle table	iptables -Z PREROUTING -t mangle
Flush all rules from output filter chain	iptables -F OUTPUT
Flush all chains	iptables -F
Lists ICMP types for us to view	iptables -p icmp --help
Deny echo-reply from all hosts	iptables -A INPUT -p icmp --icmp-type echo-reply -j DROP
Don't reply to any hosts	iptables -A OUTPUT -p icmp --icmp-type echo-request -j DROP

Allow ALL Incoming SSH on eth0	iptables -A INPUT -i eth0 -p tcp --dport 22 -m state --state NEW,ESTABLISHED -j ACCEPT iptables -A OUTPUT -o eth0 -p tcp --sport 22 -m state --state ESTABLISHED -j ACCEPT
Restrict syslog access (gateway IP should have access	iptables -A INPUT -p udp --dport 514 -s 192.168.1.1 -j ACCEPT iptables -A INPUT -p udp --dport 514 -s ! 192.168.1.1 -j DROP
Lock down NTP (129.6.15.28 and 29 are NIST)	iptables -A INPUT -p udp --dport 123 --sport 123 -s 129.6.15.2+ -j ACCEPT iptables -A INPUT -p udp --dport 123 --sport 123 -s ! 192.168.1.0/24 -j DROP
MAC address filtering	iptables -A INPUT -p tcp -m mac --mac-source 40:6c:8f:47:84:d0 -dport 8080,23 -j DROP
Set default policy to DENY	iptables -P INPUT DROP
NAT- port redirection	iptables -t nat -A PREROUTING -p tcp --dport 2323 -j REDIRECT --to-ports 23
Drop all webserver connections Multiport matches (15 ports maximum per rule)	iptables -A INPUT -m state --state NEW,ESTABLISHED -p tcp -m multiport --dport 80,443 -j DROP iptables -A OUTPUT -m state --state NEW,ESTABLISHED -p tcp -m multiport --dport 80,443 -j DROP
Make a new chain in mangle table	iptables -N INTRANET -t mangle
Rename a chain	iptables -E INTRANET MY_SUBNET
Replace/modify, all from INPUT from subnet go to MY_SUBNET	iptables -R INPUT 1 -s 192.168.1.0/24 -j MY_SUBNET
If NOT matched, pass back to origin chain, start filtering right after the rule where it left off	iptables -A MY_SUBNET -p tcp --dport 22 -j DROP

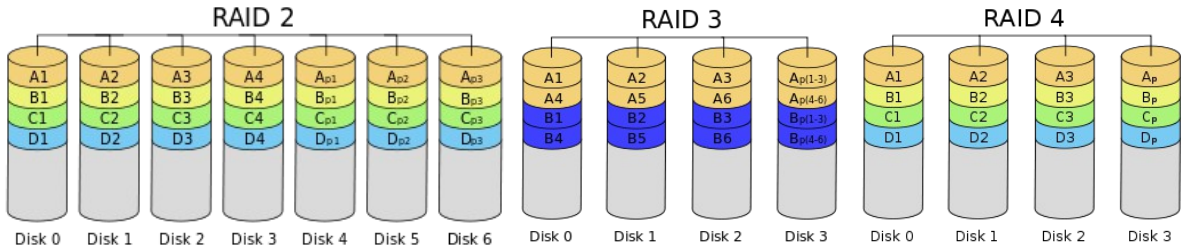
Logging goes in /var/log/messages by default. Add exception "kern.none" to line logging anything over "info" level and add line to provision further instructions
/etc/syslog.conf
uncomment #kern.* /dev/console/ and change to /var/log/firewall

iptables -A INPUT -p tcp --dport telnet -j LOG
Put these at the top of the INPUT chain (or even better, a subchain pointed to there) Default level is "warning"
iptables -A INPUT -p tcp -m multiport --dport 8080,23 -j LOG
iptables -A INPUT -p tcp -m multiport --dport ! 8080,23 -j LOG
iptables -A INPUT -p tcp --dport 22 -j LOG --log-prefix "SSH ACCESS ATTEMPT: "
iptables -A INPUT -p tcp --dport 23 -j LOG --log-prefix "UNAUTHORIZED TELNET ACCESS ATTEMPT "
--log-level debug emerg etc
--log-tcp-options
--log-ip-options
--log-tcp-sequence

Striping increases data retrieval *performance* by allowing multiple data readers and writers to work on a single data set at the same time. **Mirroring** provides *redundancy* for recovery. **Parity** *ensures* that complete data can be retrieved from an array even if one or more disks fail

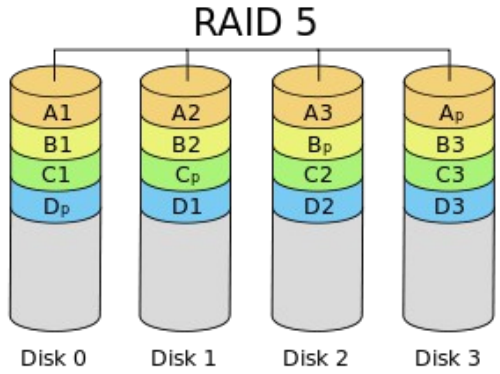
RAID 0: Disk striping
Multiple sources can access bits of data at the same time, performance can be improved.
No redundancy. One disk failure will result in lost data.

RAID 1: Disk mirroring - redundancy - no interruption of data availability
Description: Data is written to two or more disks. No master or primary, the disks are peers.
Performance: Fast read (simultaneous), slower write (writes twice)

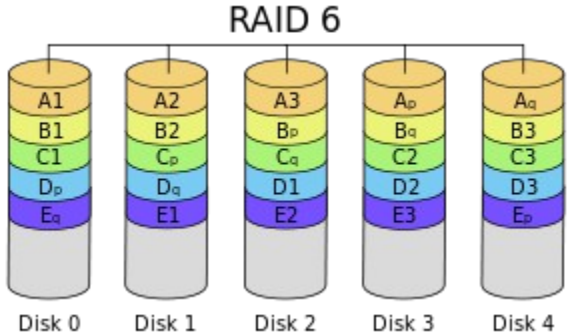


OBSOLETE! RAID2 and 3 - Byte-level striping; RAID4: Block level striping, dedicated parity disk
RAID3 used an additional disk for parity. Since every write touches multiple disks, Obsolete! Slow! All disks spin in synch (lockstep). For highest transfer rates in long sequential reads and writes Raid 4: Block-level striping, added a cache to increases performance over RAID3. **RAID5 killed off all three.**

RAID 5: Striped with distributed parity (3-5 drive minimum)
Description: A second drive failure during drive rebuild is fatal- need a hot spare.
Performance: No single parity disk bottleneck, but rebuilding drives will degrade performance. Balances data availability and read/write performance. During drive rebuilds, write performance suffers if cache isn't used.

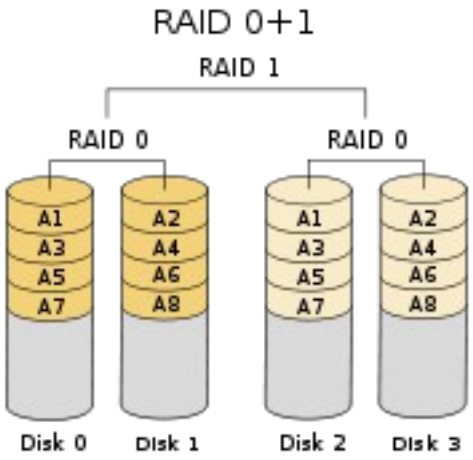


RAID 6: Dual parity
Description: RAID 5 with an additional drive to guard against a second drive failure during a drive rebuild. In the above example, it allows for 2 parity blocks per drive, instead of one.
Performance: Since each parity region is calculated separately, the RAID 5 performance impact is doubled. Some performance loss during multi-drive parity calculations and background drive rebuilds.



Nested RAID Levels

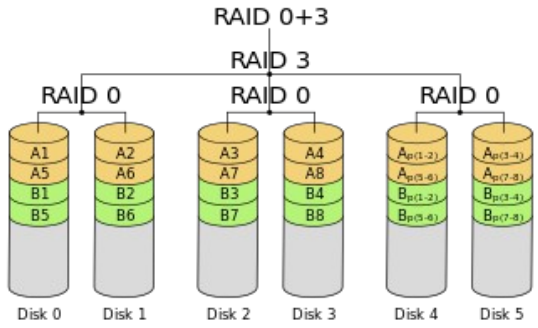
RAID01 (RAID 0+1): Mirror of Stripes - A RAID0 array is Mirrored



RAID01 uses mirror of stripes, achieving both replication and sharing of data between disks. Better performance than RAID1, and better redundancy than RAID0. Data is mirrored and then striped. If you lose a drive in a stripe set, all access to data must be from the other stripe set. Read operations are better because of striping, but write operations mirror the performance degradation of RAID 1.

The usable capacity of a RAID 01 array is the same as in a RAID 1 array made of the same drives, in which one half of the drives is used to mirror the other half. At least four disks are required in a standard RAID 01 configuration, but larger arrays are also used.

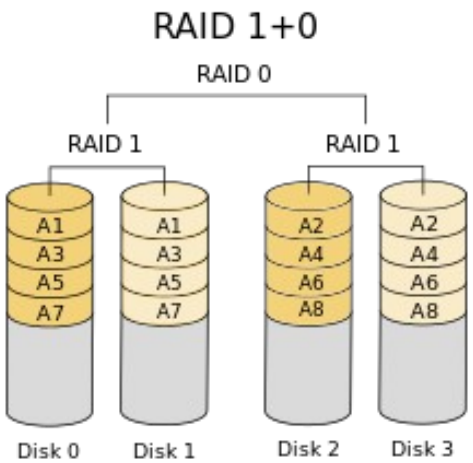
RAID03



RAID 03, also called **RAID 0+3** is byte-level striping with dedicated parity is used. Essentially, a RAID3 array is striped across RAID0 elements

RAID 53 is an accepted term for a series of RAID-5 arrays (striping with distributed parity) striped across a RAID-3 array. For all practical definition it is mostly the same thing, and it's benefits aren't important with other major types.

RAID 10 (RAID 1+0) - Stripe of Mirrors - a RAID0 array is striped across RAID1 elements

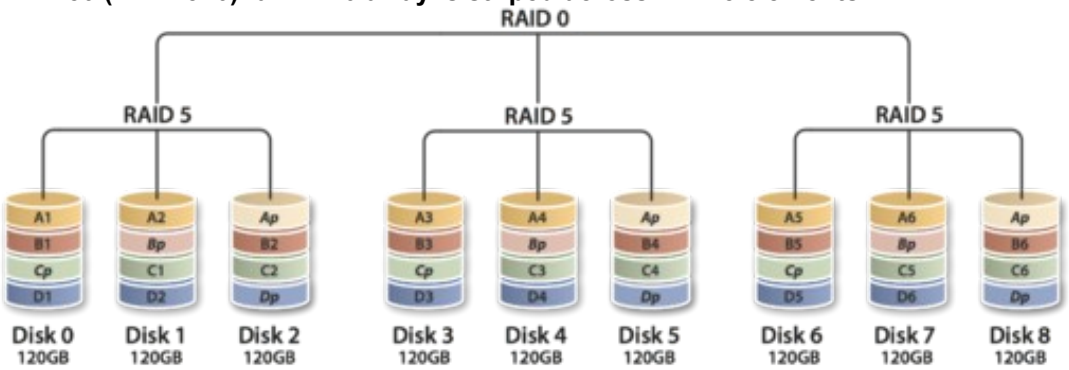


RAID10 is similar to RAID01 with an exception that two used standard RAID levels are layered in the opposite order; thus, RAID 10 is a stripe of mirrors (RAID0 array of RAID1s), which may be two- or three-way mirrors and requires a minimum of four drives.

A nonstandard Linux "RAID10" can be implemented with as few as two disks, and arrays of more than four disks are also possible.

RAID 10 provides better throughput and latency than all other RAID levels except RAID 0 (which wins in throughput). Thus, it is the preferable RAID level for I/O-intensive applications such as database, email, and web servers, as well as for any other use requiring high disk performance.

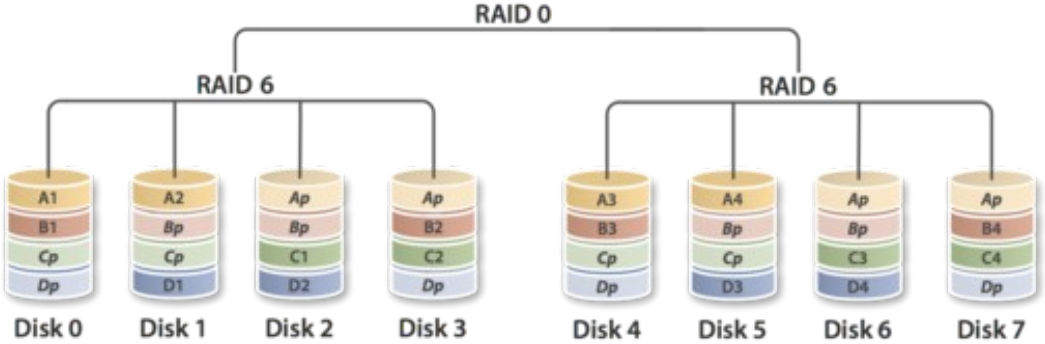
RAID 50 (RAID 5+0): a RAID0 array is striped across RAID5 elements



RAID50 aka **RAID 5+0**, combines the straight block-level striping of RAID0 with the distributed parity of RAID5. As a RAID0 array is striped across RAID5 elements, minimal RAID50 configuration requires six drives. Example shows collections of 120 GB RAID5s striped together to make 720 GB of total storage space.

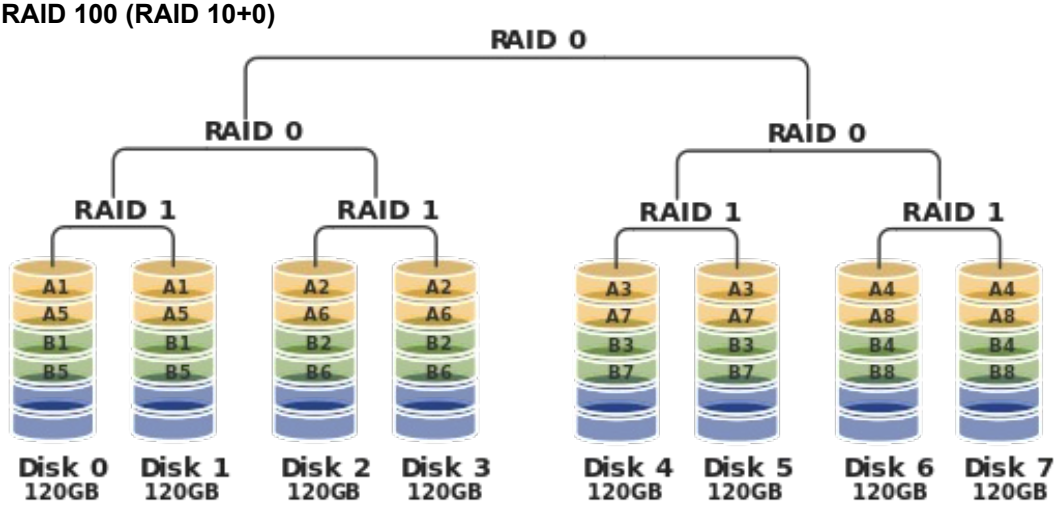
One drive from each of the RAID5 sets could fail without loss of data; for example, a RAID50 configuration including three RAID5 sets can only tolerate three maximum potential drive failures. There is still RAID5's inherent strain to rebuild a drive. As RAID5 was improved by RAID6, so was its nested counterpart, RAID 60

RAID 60 (RAID 6+0): a RAID0 array is striped across RAID6 elements



RAID 60, also called **RAID 6+0**, combines the straight block-level striping of RAID 0 with the distributed double parity of RAID 6, resulting in a RAID 0 array striped across RAID 6 elements. It requires at least eight disks.

RAID 100 (RAID 10+0)



RAID100, sometimes also called **RAID 10+0**, is a stripe of RAID10s. This is logically equivalent to a wider RAID10 array, but is generally implemented using software RAID0 over hardware RAID 10. Being "striped two ways" a RAID100 is described as a "plaid RAID"