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.

systemd-networkd seems more suitable for a server environment where network configurations are relatively stable. For desktop/laptop environments which involve various transient wired/wireless interfaces, NetworkManager may still be a preferred choice.
http://xmodulo.com/switch-from-networkmanager-to-systemd-networkd.html

netctl sidebar
Another alternative that seems to not have caught on was netctl. Online posts suggest many dropped it for Network Manager or networkd. netctl uses profiles (stored in /etc/netctl/) to manage network connections and different modes of operation to start profiles automatically or manually on demand. Example profile files are provided, copy the one you need from /etc/netctl/examples/ to /etc/netctl/ and configure it the copy

---
If you have network drives, and use auto mounting for them, I would suggest using systemd-networkd. NetworkManager has issues with bringing the network down before the network drives have been unmounted leading to a 90 second hang per mount on shutdown/reboot/suspend.

----------ALIASES - BSD
BSD man page entry says "If the address is on the same subnet as the first network address for this interface, a non-conflicting netmask must be given. Usually 0xffffffff is most appropriate"
This means, an IP address in a certain network gets its proper netmask (e.g. 255.255.255.248). An alias in that same network on the same interface gets a 255.255.255.255 netmask.
Repeat for every separate network, whether it's on the same interface or not.
So it is configured like this:
ifconfig_em0="inet **x.x.x**.20 netmask 255.255.255.**248**"
ifconfig_em0_alias0="inet **x.x.x**.21 netmask 255.255.255.**255**"
ifconfig_em0_alias1="inet **x.x.x**.22 netmask 255.255.255.**255**"

ifconfig_em0_alias2="inet **y.y.y**.44 netmask 255.255.255.**248**"
ifconfig_em0_alias3="inet **y.y.y**.45 netmask 255.255.255.**255**"
ifconfig_em0_alias4="inet **y.y.y**.46 netmask 255.255.255.**255**"

Note: WiFi stuff thrown in with layer 1 part of OSI doc

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.

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 good Linux alternative to inSSIDer -- https://sourceforge.net/projects/linssid/