Contenido

1 Commands

Comando top , para que sirve ¿??

	File Redirection	
	`> file`	create (overwrite) file
	`>> file`	append to file
	`< file`	read from file
	`a	b`	Pipe 'a' as input to 'b'
	Numeric Tests	
	`lt`	less than
	`gt`	greater than
	`eq`	equal to
	`ne`	not equal
	`ge`	greater or equal
	`le`	less or equal
	File Tests	
	`nt`	newer than
	`d`	is a directory
	`f`	is a file
	`x`	executable
	`r`	readable
	`w`	writeable
	String Tests	
	`=`	equal to
	`z`	zero length
	`n`	not zero length
	Logical Tests	
	`&&`	logical AND
	`||`	logical OR
	`!`	logical NOT
	Arguments	
	`$0`	program name

	`$1`	1st argument
	`$2`	2nd argument
	`$#`	no. of arguments
	`$*`	all arguments
	Variable Substitution	
	`${V:-default}`	$V, or ?default? if unset
	`${V:=default}`	$V (set to ?default? if unset)
	`${V:?err}`	$V, or ?err? if unset
	Conditional Execution	
	`cmd1 || cmd2`	run cmd1; if fails, run cmd2
	`cmd1 && cmd2`	run cmd1; if ok, run cmd2
	Preset Variables	
	`$SHELL`	what shell am I running?
	`$RANDOM`	provides random numbers
	`$$`	PID of current process
	`$?`	return code from last cmd
	`$!`	PID of last background cmd
	General	
•	`apropos whatis`	Show commands pertinent to string. See also threadsafe
•	`man -t ascii | ps2pdf - > ascii.pdf`	make a pdf of a manual page
	`which command`	Show full path name of command
	`time command`	See how long a command takes
•	`time cat`	Start stopwatch. Ctrl-d to stop. See also sw
	dir navegation	
•	`cd -`	Go to previous directory
•	`cd`	Go to $HOME directory
	`(cd dir && command)`	Go to dir, execute command and return to current dir
•	`pushd .`	Put current dir on stack so you can **popd** back to it
	File Searching	
•	`alias l='ls -l --color=auto'`	quick dir listing. See also l
•	`ls -lrt`	List files by date. See also newest and find_mm_yyyy
•	`ls /usr/bin | pr -T9 -W$COLUMNS`	Print in 9 columns to width of terminal
	`find -name '*.[ch]' | xargs grep -E 'expr'`	Search 'expr' in this dir and below. See also findrepo
	`find -type f -print0 | xargs -r0 grep -F 'example'`	Search all regular files for 'example' in this dir and below
	`find -maxdepth 1 -type f | xargs grep -F 'example'`	Search all regular files for 'example' in this dir
	`find -maxdepth 1 -type d | while read dir; do echo $dir; echo cmd2; done`	Process each item with multiple commands (in while loop)
•	`find -type f ! -perm -444`	Find files not readable by all (useful for web site)
•	`find -type d ! -perm -111`	Find dirs not accessible by all (useful for web site)
•	`locate -r 'file[^/]*\.txt'`	Search cached index for names. This re is like glob *file*.txt
•	`look reference`	Quickly search (sorted) dictionary for prefix
•	`grep --color reference /usr/share/dict/words`	Highlight occurances of regular expression in dictionary
	Archives and Compression	
	`gpg -c file`	Encrypt file
	`gpg file.gpg`	Decrypt file
	`tar -c dir/ | bzip2 > dir.tar.bz2`	Make compressed archive of dir/
	`bzip2 -dc dir.tar.bz2 | tar -x`	Extract archive (use gzip instead of bzip2 for tar.gz files)
	`tar -c dir/ | gzip | gpg -c | ssh user@remote 'dd of=dir.tar.gz.gpg'`	Make encrypted archive of dir/ on remote machine

	find dir/ -name '*.txt' | tar -c --files-from=- | bzip2 > dir_txt.tar.bz2	Make archive of subset of dir/ and below
	find dir/ -name '*.txt' | xargs cp -a --target-directory=dir_txt/ --parents	Make copy of subset of dir/ and below
	(tar -c /dir/to/copy) | (cd /where/to/ && tar -x -p)	Copy (with permissions) copy/ dir to /where/to/ dir
	(cd /dir/to/copy && tar -c .) | (cd /where/to/ && tar -x -p)	Copy (with permissions) contents of copy/ dir to /where/to/
	(tar -c /dir/to/copy) | ssh -C user@remote 'cd /where/to/ && tar -x -p'	Copy (with permissions) copy/ dir to remote:/where/to/ dir
	dd bs=1M if=/dev/sda | gzip | ssh user@remote 'dd of=sda.gz'	Backup harddisk to remote machine
	rsync (Network efficient file copier: Use the --dry-run option for testing)	
	rsync -P rsync://rsync.server.com/path/to/file file	Only get diffs. Do multiple times for troublesome downloads
	rsync --bwlimit=1000 fromfile tofile	Locally copy with rate limit. It's like nice for I/O
	rsync -az -e ssh --delete ~/public_html/ remote.com:'~/public_html'	Mirror web site (using compression and encryption)
	rsync -auz -e ssh remote:/dir/ . && rsync -auz -e ssh . remote:/dir/	Synchronize current directory with remote one
	ssh (Secure SHell)	
	ssh $USER@$HOST command	Run command on $HOST as $USER (default command=shell)
•	ssh -f -Y $USER@$HOSTNAME xeyes	Run GUI command on $HOSTNAME as $USER
	scp -p -r $USER@$HOST: file dir/	Copy with permissions to $USER's home directory on $HOST
	scp -c arcfour $USER@$LANHOST: bigfile	Use faster crypto for local LAN. This might saturate GigE
	ssh -g -L 8080:localhost:80 root@$HOST	Forward connections to $HOSTNAME:8080 out to $HOST:80
	ssh -R 1434:imap:143 root@$HOST	Forward connections from $HOST:1434 in to imap:143
	ssh-copy-id $USER@$HOST	Install public key for $USER@$HOST for password-less log in
	wget (multi purpose download tool)	
•	(cd dir/ && wget -nd -pHEKk http://www.pixelbeat.org/cmdline.html)	Store local browsable version of a page to the current dir
	wget -c http://www.example.com/large.file	Continue downloading a partially downloaded file
	wget -r -nd -np -l1 -A '*.jpg' http://www.example.com/dir/	Download a set of files to the current directory
	wget ftp://remote/file[1-9].iso/	FTP supports globbing directly
•	wget -q -O- http://www.pixelbeat.org/timeline.html | grep 'a href' | head	Process output directly
	echo 'wget url' | at 01:00	Download url at 1AM to current dir
	wget --limit-rate=20k url	Do a low priority download (limit to 20KB/s in this case)
	wget -nv --spider --force-html -i bookmarks.html	Check links in a file
	wget --mirror http://www.example.com/	Efficiently update a local copy of a site (handy from cron)
	networking (Note ifconfig, route, mii-tool, nslookup commands are obsolete)	
	ethtool eth0	Show status of ethernet interface eth0
	ethtool --change eth0 autoneg off speed 100 duplex full	Manually set ethernet interface speed
	iw dev wlan0 link	Show link status of wireless interface wlan0
	iw dev wlan0 set bitrates legacy-2.4 1	Manually set wireless interface speed
•	iw dev wlan0 scan	List wireless networks in range
•	ip link show	List network interfaces
	ip link set dev eth0 name wan	Rename interface eth0 to wan
	ip link set dev eth0 up	Bring interface eth0 up (or down)
•	ip addr show	List addresses for interfaces
	ip addr add 1.2.3.4/24 brd + dev eth0	Add (or del) ip and mask (255.255.255.0)
•	ip route show	List routing table
	ip route add default via 1.2.3.254	Set default gateway to 1.2.3.254
•	ss -tupl	List internet services on a system
•	ss -tup	List active connections to/from system
•	host pixelbeat.org	Lookup DNS ip address for name or vice versa
•	hostname -i	Lookup local ip address (equivalent to host `hostname`)

•	`whois pixelbeat.org`	Lookup whois info for hostname or ip address
	windows networking (Note samba is the package that provides all this windows specific networking support)	
•	`smbtree`	Find windows machines. See also findsmb
	`nmblookup -A 1.2.3.4`	Find the windows (netbios) name associated with ip address
	`smbclient -L windows_box`	List shares on windows machine or samba server

	`mount -t smbfs -o fmask=666,guest`
	`//windows_box/share /mnt/share`
		Mount a windows share

	`echo 'message' | smbclient -M windows_box`	Send popup to windows machine (off by default in XP sp2)
	text manipulation (Note sed uses stdin and stdout. Newer versions support inplace editing with the -i option)	
	`sed 's/string1/string2/g'`	Replace string1 with string2
	`sed 's/\(.*\)1/\12/g'`	Modify anystring1 to anystring2
	`sed '/^ *#/d; /^ *$/d'`	Remove comments and blank lines
	`sed ':a; /\\$/N; s/\\\n//; ta'`	Concatenate lines with trailing \
	`sed 's/[ \t]*$//'`	Remove trailing spaces from lines
	``sed 's/\([`"$\]\)/\\\1/g'``	Escape shell metacharacters active within double quotes
•	`seq 10 | sed "s/^/      /; s/ *\(.\{7,\}\)/\1/"`	Right align numbers
•	`seq 10 | sed p | paste - -`	Duplicate a column
	`sed -n '1000{p;q}'`	Print 1000th line
	`sed -n '10,20p;20q'`	Print lines 10 to 20
	`sed -n 's/.*<title>\(.*\)<\/title>.*/\1/ip;T;q'`	Extract title from HTML web page
	`sed -i 42d ~/.ssh/known_hosts`	Delete a particular line
	`sort -t. -k1,1n -k2,2n -k3,3n -k4,4n`	Sort IPV4 ip addresses
•	`echo 'Test' | tr '[:lower:]' '[:upper:]'`	Case conversion
•	`tr -dc '[:print:]' < /dev/urandom`	Filter non printable characters
•	`tr -s '[:blank:]' '\t' </proc/diskstats | cut -f4`	cut fields separated by blanks
•	`history | wc -l`	Count lines
•	`seq 10 | paste -s -d ' '`	Concatenate and separate line items to a single line
	set operations (Note you can export LANG=C for speed. Also these assume no duplicate lines within a file)	
	`sort file1 file2 | uniq`	Union of unsorted files
	`sort file1 file2 | uniq -d`	Intersection of unsorted files
	`sort file1 file1 file2 | uniq -u`	Difference of unsorted files
	`sort file1 file2 | uniq -u`	Symmetric Difference of unsorted files
	`join -t'\0' -a1 -a2 file1 file2`	Union of sorted files
	`join -t'\0' file1 file2`	Intersection of sorted files
	`join -t'\0' -v2 file1 file2`	Difference of sorted files
	`join -t'\0' -v1 -v2 file1 file2`	Symmetric Difference of sorted files
	math	
•	`echo '(1 + sqrt(5))/2' | bc -l`	Quick math (Calculate φ). See also bc
•	`seq -f '4/%g' 1 2 99999 | paste -sd-+ | bc -l`	Calculate π the unix way

•	`echo 'pad=20; min=64; (100*10^6)/((pad+min)*8)' |`
	`bc`
		More complex (int) e.g. This shows max FastE packet rate

•	`echo 'pad=20; min=64; print (100E6)/((pad+min)*8)'`
	`| python`
		Python handles scientific notation

•	`echo 'pad=20; plot [64:1518]`
	`(100*10**6)/((pad+x)*8)' | gnuplot -persist`
		Plot FastE packet rate vs packet size

•	`echo 'obase=16; ibase=10; 64206' | bc`	Base conversion (decimal to hexadecimal)
•	`echo $((0x2dec))`	Base conversion (hex to dec) ((shell arithmetic expansion))
•	`units -t '100m/9.58s' 'miles/hour'`	Unit conversion (metric to imperial)
•	`units -t '500GB' 'GiB'`	Unit conversion (SI to IEC prefixes)
•	`units -t '1 googol'`	Definition lookup
•	`seq 100 | paste -s -d+ | bc`	Add a column of numbers. See also add and funcpy
	calendar	
•	`cal -3`	Display a calendar
•	`cal 9 1752`	Display a calendar for a particular month year

•	date -d fri	What date is it this friday. See also day
•	[$(date -d '12:00 today +1 day' +%d) = '01'] || exit	exit a script unless it's the last day of the month
•	date --date='25 Dec' +%A	What day does xmas fall on, this year
•	date --date='@2147483647'	Convert seconds since the epoch (1970-01-01 UTC) to date
•	TZ='America/Los_Angeles' date	What time is it on west coast of US (use tzselect to find TZ)
•	date --date='TZ="America/Los_Angeles" 09:00 next Fri'	What's the local time for 9AM next Friday on west coast US
	locales	
•	printf "%'d\n" 1234	Print number with thousands grouping appropriate to locale
•	BLOCK_SIZE=\'1 ls -l	Use locale thousands grouping in ls. See also l
•	echo "I live in \`locale territory\`"	Extract info from locale database
•	LANG=en_IE.utf8 locale int_prefix	Lookup locale info for specific country. See also ccodes
•	locale -kc $(locale | sed -n 's/\(LC_.\{4,\}\)=.*/\1/p') | less	List fields available in locale database
	recode (Obsoletes iconv, dos2unix, unix2dos)	
•	recode -l | less	Show available conversions (aliases on each line)
	recode windows-1252.. file_to_change.txt	Windows "ansi" to local charset (auto does CRLF conversion)
	recode utf-8/CRLF.. file_to_change.txt	Windows utf8 to local charset
	recode iso-8859-15..utf8 file_to_change.txt	Latin9 (western europe) to utf8
	recode ../b64 < file.txt > file.b64	Base64 encode
	recode /qp.. < file.qp > file.txt	Quoted printable decode
	recode ..HTML < file.txt > file.html	Text to HTML
•	recode -lf windows-1252 | grep euro	Lookup table of characters
•	echo -n 0x80 | recode latin-9/x1..dump	Show what a code represents in latin-9 charmap
•	echo -n 0x20AC | recode ucs-2/x2..latin-9/x	Show latin-9 encoding
•	echo -n 0x20AC | recode ucs-2/x2..utf-8/x	Show utf-8 encoding
	CDs	
	gzip < /dev/cdrom > cdrom.iso.gz	Save copy of data cdrom
	mkisofs -V LABEL -r dir | gzip > cdrom.iso.gz	Create cdrom image from contents of dir
	mount -o loop cdrom.iso /mnt/dir	Mount the cdrom image at /mnt/dir (read only)
	wodim dev=/dev/cdrom blank=fast	Clear a CDRW
	gzip -dc cdrom.iso.gz | wodim -tao dev=/dev/cdrom -v -data -	Burn cdrom image (use --prcap to confirm dev)
	cdparanoia -B	Rip audio tracks from CD to wav files in current dir
	wodim -v dev=/dev/cdrom -audio -pad *.wav	Make audio CD from all wavs in current dir (see also cdrdao)
	oggenc --tracknum=$track track.cdda.wav -o track.ogg	Make ogg file from wav file
	disk space (See also FSlint)	
•	ls -lSr	Show files by size, biggest last
•	du -s * | sort -k1,1rn | head	Show top disk users in current dir. See also dutop
•	du -hs /home/* | sort -k1,1h	Sort paths by easy to interpret disk usage
•	df -h	Show free space on mounted filesystems
•	df -i	Show free inodes on mounted filesystems
•	fdisk -l	Show disks partitions sizes and types (run as root)
•	rpm -q -a --qf '%10{SIZE}\t%{NAME}\n' | sort -k1,1n	List all packages by installed size (Bytes) on rpm distros
•	dpkg-query -W -f='${Installed-Size;10}\t${Package}\n' | sort -k1,1n	List all packages by installed size (KBytes) on deb distros
•	dd bs=1 seek=2TB if=/dev/null of=ext3.test	Create a large test file (taking no space). See also truncate
•	> file	truncate data of file or create an empty file
	disk space (See also FSlint)	
•	tail -f /var/log/messages	Monitor messages in a log file
•	strace -c ls >/dev/null	Summarise/profile system calls made by command
•	strace -f -e open ls >/dev/null	List system calls made by command
•	strace -f -e trace=write -e write=1,2 ls	Monitor what's written to stdout and stderr

	`>/dev/null`	
•	`ltrace -f -e getenv ls >/dev/null`	List library calls made by command
•	`lsof -p $$`	List paths that process id has open
•	`lsof ~`	List processes that have specified path open
•	`tcpdump not port 22`	Show network traffic except ssh. See also tcpdump_not_me
•	`ps -e -o pid,args --forest`	List processes in a hierarchy
•	`ps -e -o pcpu,cpu,nice,state,cputime,args --sort pcpu | sed '/^ 0.0 /d'`	List processes by % cpu usage
•	`ps -e -orss=,args= | sort -b -k1,1n | pr -TW$COLUMNS`	List processes by mem (KB) usage. See also ps_mem.py
•	`ps -C firefox-bin -L -o pid,tid,pcpu,state`	List all threads for a particular process
•	`ps -p 1,$$ -o etime=`	List elapsed wall time for particular process IDs
•	`watch -n.1 pstree -Uacp $$`	Display a changing process subtree
•	`last reboot`	Show system reboot history
•	`free -m`	Show amount of (remaining) RAM (-m displays in MB)
•	`watch -n.1 'cat /proc/interrupts'`	Watch changeable data continuously
•	`udevadm monitor`	Monitor udev events to help configure rules
	system information (see also sysinfo) ('#' means root access is required)	
•	`uname -a`	Show kernel version and system architecture
•	`head -n1 /etc/issue`	Show name and version of distribution
•	`cat /proc/partitions`	Show all partitions registered on the system
•	`grep MemTotal /proc/meminfo`	Show RAM total seen by the system
•	`grep "model name" /proc/cpuinfo`	Show CPU(s) info
•	`lspci -tv`	Show PCI info
•	`lsusb -tv`	Show USB info
•	`mount | column -t`	List mounted filesystems on the system (and align output)
•	`grep -F capacity: /proc/acpi/battery/BAT0/info`	Show state of cells in laptop battery
#	`dmidecode -q | less`	Display SMBIOS/DMI information
#	`smartctl -A /dev/sda | grep Power_On_Hours`	How long has this disk (system) been powered on in total
#	`hdparm -i /dev/sda`	Show info about disk sda
#	`hdparm -tT /dev/sda`	Do a read speed test on disk sda
#	`badblocks -s /dev/sda`	Test for unreadable blocks on disk sda
	interactive (see also linux keyboard shortcuts)	
•	`readline`	Line editor used by bash, python, bc, gnuplot, ...
•	`screen`	Virtual terminals with detach capability, ...
•	`mc`	Powerful file manager that can browse rpm, tar, ftp, ssh, ...
•	`gnuplot`	Interactive/scriptable graphing
•	`links`	Web browser
•	`xdg-open .`	open a file or url with the registered desktop application
•	`grep . /proc/sys/net/ipv4/*`	List the contents of flag files
•	`set | grep $USER`	Search current environment
•	`tr '\0' '\n' < /proc/$$/environ`	Display the *startup* environment for any process
•	`echo $PATH | tr : '\n'`	Display the $PATH one per line
•	`kill -0 $$ && echo process exists and can accept signals`	Check for the existence of a process (pid)
•	`find /etc -readable | xargs less -K -p'*ntp' -j $((${LINES:-25}/2))`	Search paths and data with full context. Use **n** to iterate
•	`namei -l ~/.ssh`	Output attributes for all directories leading to a file name
	Low impact admin	
#	`apt-get install "package" -o Acquire::http::Dl-Limit=42 \`	Rate limit apt-get to 42KB/s
	`-o Acquire::Queue-mode=access`	
	`echo 'wget url' | at 01:00`	Download url at 1AM to current dir
#	`apache2ctl configtest && apache2ctl graceful`	Restart apache if config is OK
•	`nice openssl speed sha1`	Run a low priority command (openssl benchmark)
•	`chrt -i 0 openssl speed sha1`	Run a low priority command (more effective than nice)
•	`renice 19 -p $$; ionice -c3 -p $$`	Make shell (script) low priority. Use for non interactive tasks

	Interactive monitoring	
•	watch -t -n1 uptime	Clock with system load
•	htop -d 5	Better top (scrollable, tree view, lsof/strace integration, ...)
•	iotop	What's doing I/O
#	watch -d -n30 "nice ps_mem.py | tail -n $((${LINES:-12}-2))"	What's using RAM
#	iftop	What's using the network. See also iptraf
#	mtr www.pixelbeat.org	ping and traceroute combined
	Useful utilities	
•	pv < /dev/zero > /dev/null	Progress Viewer for data copying from files and pipes
•	wkhtml2pdf http://.../linux_commands.html linux_commands.pdf	Make a pdf of a web page
•	timeout 1 sleep inf	run a command with bounded time. See also timeout
	Networking	
•	python -m SimpleHTTPServer	Serve current directory tree at http://$HOSTNAME:8000/
•	openssl s_client -connect www.google.com:443 </dev/null 2>&0 |	Display the date range for a site's certs
	openssl x509 -dates -noout	
•	curl -I www.pixelbeat.org	Display the server headers for a web site
#	lsof -i tcp:80	What's using port 80
#	httpd -S	Display a list of apache virtual hosts
•	vim scp://user@remote//path/to/file	Edit remote file using local vim. Good for high latency links
•	curl -s http://www.pixelbeat.org/pixelbeat.asc | gpg --import	Import a gpg key from the web
•	tc qdisc add dev lo root handle 1:0 netem delay 20msec	Add 20ms latency to loopback device (for testing)
•	tc qdisc del dev lo root	Remove latency added above
	Notification	
•	echo "DISPLAY=$DISPLAY xmessage cooker" | at "NOW +30min"	Popup reminder
•	notify-send "subject" "message"	Display a gnome popup notification
	echo "mail -s 'go home' P@draigBrady.com < /dev/null" | at 17:30	Email reminder
	uuencode file name | mail -s subject P@draigBrady.com	Send a file via email
	ansi2html.sh | mail -a "Content-Type: text/html" P@draigBrady.com	Send/Generate HTML email
	Better default settings (useful in your .bashrc)	
#	tail -s.1 -f /var/log/messages	**Display file additions more responsively**
•	seq 100 | tail -n $((${LINES:-12}-2))	Display as many lines as possible without scrolling
#	tcpdump -s0	Capture full network packets
	Useful functions/aliases (useful in your .bashrc)	
•	md () { mkdir -p "$1" && cd "$1"; }	Change to a new directory
•	strerror() { python -c "import os; print os.strerror($1)"; }	Display the meaning of an errno
•	plot() { { echo 'plot "-"' "$@"; cat; } | gnuplot -persist; }	Plot stdin. (e.g: • seq 1000 | sed 's/.*/s(&)/' | bc -l | plot)
•	hili() { e="$1"; shift; grep --col=always -Eih "$e|$" "$@"; }	highlight occurences of expr. (e.g: • env | hili $USER)
•	alias hd='od -Ax -tx1z -v'	Hexdump. (usage e.g.: • hd /proc/self/cmdline | less)
•	alias realpath='readlink -f'	Canonicalize path. (usage e.g.: • realpath ~/../$USER)
•	ord() { printf "0x%x\n" "'$1"; }	shell version of the ord() function
•	chr() { printf $(printf '\\%03o\\n' "$1"); }	shell version of the chr() function
	Multimedia	
•	DISPLAY=:0.0 import -window root orig.png	Take a (remote) screenshot
•	convert -filter catrom -resize '600x>' orig.png 600px_wide.png	Shrink to width, computer gen images or screenshots

	`mplayer -ao pcm -vo null -vc dummy /tmp/Flash*`	Extract audio from flash video to audiodump.wav
	`ffmpeg -i filename.avi`	Display info about multimedia file
•	`ffmpeg -f x11grab -s xga -r 25 -i :0 -sameq demo.mpg`	Capture video of an X display
	DVD	
	`for i in $(seq 9); do ffmpeg -i $i.avi -target pal-dvd $i.mpg; done`	Convert video to the correct encoding and aspect for DVD
	`dvdauthor -odvd -t -v "pal,4:3,720xfull" *.mpg;dvdauthor -odvd -T`	Build DVD file system. Use 16:9 for widescreen input
	`growisofs -dvd-compat -Z /dev/dvd -dvd-video dvd`	Burn DVD file system to disc
	Unicode	
•	`python -c "import unicodedata as u; print u.name(unichr(0x2028))"`	Lookup a unicode character
•	`uconv -f utf8 -t utf8 -x nfc`	Normalize combining characters
•	`printf '\300\200' | iconv -futf8 -tutf8 >/dev/null`	Validate UTF-8
•	`printf 'ŨTF8\n' | LANG=C grep --color=always '[^ -~]\+'`	Highlight non printable ASCII chars in UTF-8
•	`fc-match -s "sans:lang=zh"`	List font match order for language and style
	Development	
•	`gcc -march=native -E -v -</dev/null 2>&1|sed -n 's/.*-mar/-mar/p'`	Show autodetected gcc tuning params. See also gcccpuopt
•	`for i in $(seq 4); do { [ $i = 1 ] && wget http://url.ie/6lko -qO-||`	Compile and execute C code from stdin
	`./a.out; } | tee /dev/tty | gcc -xc - 2>/dev/null; done`	
•	`cpp -dM /dev/null`	Show all predefined macros
•	`echo "#include <features.h>" | cpp -dN | grep "#define __USE_"`	Show all glibc feature macros
	`gdb -tui`	Debug showing source code context in separate windows
	udev	
•	`udevadm info -a -p $(udevadm info -q path -n /dev/input/mouse0)`	List udev attributes of a device, for matching rules etc.
•	`udevadm test /sys/class/`input/mouse0	See how udev rules are applied for a device
#	`udevadm control --reload-rules`	Reload udev rules after modification
	Extended Attributes (Note you may need to (re)mount with "acl" or "user_xattr" options)	
•	`getfacl .`	Show ACLs for file
•	`setfacl -m u:nobody:r .`	Allow a specific user to read file
•	`setfacl -x u:nobody .`	Delete a specific user's rights to file
	`setfacl --default -m group:users:rw- dir/`	Set umask for a for a specific dir
	`getcap file`	Show capabilities for a program
	`setcap cap_net_raw+ep your_gtk_prog`	Allow gtk program raw access to network
•	`stat -c%C .`	Show SELinux context for file
	`chcon ... file`	Set SELinux context for file (see also restorecon)
•	`getfattr -m- -d .`	Show all extended attributes (includes selinux,acls,...)
•	`setfattr -n "user.foo" -v "bar" .`	Set arbitrary user attributes
	BASH specific	
•	`echo 123 | tee >(tr 1 a) | tr 1 b`	Split data to 2 commands (using process substitution)
	`meld local_file <(ssh host cat remote_file)`	Compare a local and remote file (using process substitution)
	Multicore	
•	`taskset -c 0 nproc`	Restrict a command to certain processors
•	`find -type f -print0 | xargs -r0 -P$(nproc) -n10 md5sum`	Process files in parallel over available processors
	`sort -m <(sort data1) <(sort data2) >data.sorted`	Sort separate data files over 2 processors

2 Main directories (FHS standard)

The standard Ubuntu directory structure mostly follows the Filesystem Hierarchy Standard, which can be referred to for more detailed information.

Here, only the most important directories in the system will be presented.

/bin is a place for most commonly used terminal commands, like ls, mount, rm, etc.

/boot contains files needed to start up the system, including the Linux kernel, a RAM disk image and bootloader configuration files.

/dev contains all *device files*, which are not regular files but instead refer to various hardware devices on the system, including hard drives.

/etc contains system-global configuration files, which affect the system's behavior for all users.

/home home sweet home, this is the place for users' home directories.

/lib contains very important dynamic libraries and kernel modules

/media is intended as a mount point for external devices, such as hard drives or removable media (floppies, CDs, DVDs).

/mnt is also a place for mount points, but dedicated specifically to "temporarily mounted" devices, such as network filesystems.

/opt can be used to store additional software for your system, which is not handled by the package manager.

/proc is a virtual filesystem that provides a mechanism for kernel to send information to processes.

/root is the superuser's home directory, not in /home/ to allow for booting the system even if /home/ is not available.

/run is a *tmpfs* (temporary file system) available early in the boot process where ephemeral run-time data is stored. Files under this directory are removed or truncated at the beginning of the boot process. (It deprecates various legacy locations such as /var/run, /var/lock, /lib/init/rw in otherwise non-ephemeral directory trees as well as /dev/.* and /dev/shm which are not device files.)

/sbin contains important administrative commands that should generally only be employed by the superuser.

/srv can contain data directories of services such as HTTP (/srv/www/) or FTP.

/sys is a virtual filesystem that can be accessed to set or obtain information about the kernel's view of the system.

/tmp is a place for temporary files used by applications.

/usr contains the majority of user utilities and applications, and partly replicates the root directory structure, containing for instance, among others, /usr/bin/ and /usr/lib.

/var is dedicated to variable data, such as logs, databases, websites, and temporary spool (e-mail etc.) files that persist from one boot to the next. A notable directory it contains is /var/log where system log files are kept.

3 User Profiles

1.1 Systemwide Profile For All Users

/etc/profile: You need to update /etc/profile which is systemwide initialization profile file. All changes made to this file applies to all users on the system.

/etc/bash.bashrc : The systemwide per-interactive-shell startup file. This file is called from /etc/profile. Edit this file and set settings such as JAVA PATH, CLASSPATH and so on

2.1 Profile For Individual Users

Use the following shell startup files to customize each user profile. The following files are located in users $HOME directory such as /home/vivek.

$HOME/.bash_profile – The personal initialization file, executed for login shells. Add *PATH settings and other user specific variables* to this file.

$HOME/.bashrc – The individual per-interactive-shell startup file. Add user specific *aliases and functions* to this file.

$HOME/.bash_logout – The individual login shell cleanup file, executed when a login shell exits.

4 BASH SHELL CONSTRUCTS

http://www.informit.com/articles/article.aspx?p=350778&seqNum=6

The shbang line	The "shbang" line is the very first line of the script and lets the kernel know what shell will be interpreting the lines in the script. The shbang line consists of a `#!` followed by the full pathname to the shell, and can be followed by options to control the behavior of the shell. ***EXAMPLE:*** `#!/bin/bash`
Comments	Comments are descriptive material preceded by a `#` sign. They are in effect until the end of a line and can be started anywhere on the line. ***EXAMPLE:*** `# This is a comment`

Wildcards	There are some characters that are evaluated by the shell in a special way. They are called shell metacharacters or "wildcards." These characters are neither numbers or letters. For example, the `*`, `?`, and `[ ]` are used for filename expansion. The `<`, `>`, `2>`, `>>`, and `|` symbols are used for standard I/O redirection and pipes. To prevent these characters from being interpreted by the shell they must be quoted.
	EXAMPLE:
	`rm *; ls ??; cat file[1-3];`
	`echo "How are you?"`

Displaying	To print output to the screen, the `echo` command is used. Wildcards

output	must be escaped with either a backslash or matching quotes. **E*XAMPLE*** `echo "How are you?"`

Local variables	Local variables are in scope for the current shell. When a script ends, they are no longer available; i.e., they go out of scope. Local variables can also be defined with the built-in `declare` function. Local variables are set and assigned values. **E*XAMPLE***
	`variable_name=value`
	`declare variable_name=value`
	`name="John Doe"`
	`x=5`

Global variables	Global variables are called environment variables and are created with the `export` built-in command. They are set for the currently running shell and any process spawned from that shell. They go out of scope when the script ends.
	The built-in `declare` function with the `-x` option also sets an environment variable and marks it for export. **E*XAMPLE***
	`export VARIABLE_NAME=value`
	`declare -x VARIABLE_NAME=value`
	`export PATH=/bin:/usr/bin:.`

Extracting values from variables	To extract the value from variables, a dollar sign is used.**E*XAMPLE***
	`echo $variable_name`
	`echo $name`
	`echo $PATH`

Reading user input	The user will be asked to enter input. The `read` command is used to accept a line of input. Multiple arguments to `read` will cause a line to be broken into words, and each word will be assigned to the named variable. **E*XAMPLE***
	`echo "What is your name?"`
	`read name`
	`read name1 name2 ...`

Arguments	Arguments can be passed to a script from the command line. Positional parameters are used to receive their values from within the script. **E*XAMPLE*** At the command line: `$ scriptname arg1 arg2 arg3 ...` In a script: `echo $1 $2 $3` Positional parameters `echo $*` All the positional paramters `echo $#` The number of positional parameters
Arrays	The Bourne shell utilizes positional parameters to create a word list. In addition to positional parameters, the Bash shell supports an array syntax whereby the elements are accessed with a subscript, starting at 0. Bash shell arrays are created with the `declare -a` command. **E*XAMPLE*** `set apples pears peaches (positional parameters)`

```
echo $1 $2 $3

declare -a array_name=(word1 word2 word3 ...)
declare -a fruit=( apples pears plums )
echo ${fruit[0]}
```

Command substitution

Like the C/TC shells and the Bourne shell, the output of a UNIX/Linux command can be assigned to a variable, or used as the output of a command in a string, by enclosing the command in backquotes. The Bash shell also provides a new syntax. Instead of placing the command between backquotes, it is enclosed in a set of parentheses, preceded by a dollar sign. ***EXAMPLE***

```
variable_name=`command`
variable_name=$( command )
echo $variable_name

echo "Today is `date`"
echo "Today is $(date)"
```

Arithmetic

The Bash shells support integer arithmetic. The `declare -i` command will declare an integer type variable.The Korn shell's `typeset` command can also be use for backward compatibility. Integer arithmetic can be performed on variables declared this way. Otherwise the `(( ))` (`let` command) syntax is used for arithmetic operations. ***EXAMPLE***

```
declare -i variable_name    used for bash
typeset -i variable_name    can be used to be compatible
with ksh
(( n=5 + 5 ))
echo $n
```

Operators

The Bash shell uses the built-in `test` command operators to test numbers and strings, similar to C language operators. ***EXAMPLE***

Equality:	**Logical:**
`==` *equal to*	`&&` *and*
`!=` *not equal to*	`||` *or*
	`!` *not*

Relational:
`>` *greater than*
`>=` *greater than, equal to*
`<` *less than*
`<=` *less than, equal to*

Conditional statements

The `if` construct is followed by an expression enclosed in parentheses. The operators are similar to C operators. The `then` keyword is placed after the closing paren. An `if` must end with an `endif`. The new `[[ ]]` test command is now used to allow pattern matching in conditional expressions. The old `[ ]` test command is still available for backward compatibility with the Bourne shell. The `case` command is an alternative to `if/else`. ***EXAMPLE***

The `if` construct is:

```
if  command
then
   block of statements
fi


if  [[ expression  ]]
then
```

The `if/else/else if` construct is:

```
if  command
then
   block of statements
elif  command
then
   block of statements
else if  command
```

```
   block of statements
fi


if  (( numeric expression
))
then
   block of statements
else
   block of statements
 fi
```

The `if/else` construct is:

```
if  command
then
   block of statements
else
   block of statements
fi


if  [[ expression ]]
then
   block of statements
else
   block of statements
fi


if  ((  numeric expression
))
then
   block of statements
else
   block of statements
fi
```

The `case` construct is:

```
case variable_name in
   pattern1)
      statements
         ;;
   pattern2)
      statements
         ;;
   pattern3)
         ;;
esac


case "$color" in
   blue)
      echo $color is blue
         ;;
   green)
      echo $color is green
         ;;
   red|orange)
      echo $color is red or
orange
         ;;
   *) echo "Not a matach"
         ;;
esac
```

```
then
   block of statements
else
   block of statements
fi
-------------------------
if  [[ expression ]]
then
   block of statements
elif  [[  expression ]]
then
   block of statements
else if  [[  expression ]]
then
   block of statements
else
   block of statements
fi
--------------------------
if  ((  numeric expression ))
then
   block of statements
elif  ((  numeric expression
))
then
   block of statements
else if  ((numeric
expression))
then
   block of statements
else
   block of statements
fi
```

Loops	There are four types of loops: `while`, `until`, `for`, and `select`. The `while` loop is followed by an expression enclosed in square brackets, a `do` keyword, a block of statements, and terminated with the `done` keyword. As long as the expression is true, the body of statements between `do` and `done` will be executed. The compound test operator `[[ ]]` is new with Bash, and the old-style test operator `[ ]` can still be used to evaluate conditional expressions for backward

compatibility with the Bourne shell.

The until loop is just like the while loop, except the body of the loop will be executed as long as the expression is false.

The for loop is used to iterate through a list of words, processing a word and then shifting it off, to process the next word. When all words have been shifted from the list, it ends. The for loop is followed by a variable name, the in keyword, a list of words, then a block of statements, and terminates with the done keyword.

The select loop is used to provide a prompt and a menu of numbered items from which the user inputs a selection. The input will be stored in the special built-in REPLY variable. The select loop is normally used with the case command.

The loop control commands are break and continue. The break command allows control to exit the loop before reaching the end of it, and the continue command allows control to return to the looping expression before reaching the end.

E*XAMPLE*

```
while command                                until
➥ command
               do                                        do
               block of statements
               ➥ block of statements
               done
done
               -------------------------
               ➥ ---------------------------
               while [[ string expression ]]
until
               ➥ [[ string expression ]]
               do
do
               block of statements
block
               ➥ of statements
               done
done
               -------------------------
               ➥ ----------------------------
               while (( numeric expression ))
until
               ➥ (( numeric expression ))
               do
do
               block of statements
               ➥ block of statements
               done
done

               for variable in word_list
               ➥ select variable in word_list
               do
do
               block of statements
block
               ➥ of statements
               done
done
               --------------------------
               ➥ ----------------------------
               for color in red green b
               ➥ PS3="Select an item from the menu"
               do
do
               ➥ item in blue red green
               echo $color
echo
               ➥ $item
```

	`                  done`
	`done`
	
	Shows menu:
	
	1. *blue*
	2. *red*
	
	green

Functions	Functions allow you to define a section of shell code and give it a name. There are two formats, one from the Bourne shell, and the Bash version that uses the `function` keyword. ***EXAMPLE***
	
	`function_name() {`
	`   block of code`
	`}`
	
	`function  function_name {`
	`   block of code`
	`}`
	`------------------------`
	`function  lister {`
	``   echo Your present working directory is `pwd` ``
	`   echo Your files are:`
	`   ls`
	`}`

The bash Shell example

```
   1   #!/bin/bash
   # GNU bash versions 2.x
   2   # The Party Program--Invitations to friends from the "guest" file
   3   guestfile=~/shell/guests
   4   if [[ ! -e "$guestfile" ]]
   then
   5       printf "${guestfile##*/} non-existent"
   exit 1
   fi
   6   export PLACE="Sarotini's"
   7   (( Time=$(date +%H) + 1 ))
   8   declare -a foods=(cheese crackers shrimp drinks `"hot dogs"`
sandwiches)
   9   declare -i  n=0
   10  for person in $(cat $guestfile)
   do
   11      if  [[ $person == root ]]
   then
   continue
   else
   # Start of here document
   12            mail -v -s "Party" $person <<- FINIS
   Hi $person! Please join me at $PLACE for a party!
   Meet me at $Time o'clock.
   I'll bring the ice cream. Would you please bring
   ${foods[$n] and anything else you would like to eat?
   Let me know if you can make it.
   Hope to see you soon.
   Your pal,
   ellie@$(hostname)
   FINIS
   13            n=n+1
   14            if (( ${#foods[*]} ==  $n ))
   then
   15               declare -a foods=(cheese crackers shrimp drinks `"hot
dogs"`
   sandwiches)
   16            n=0
   fi
   fi
   17  done
   printf "Bye..."
```

1. This line lets the kernel know that you are running a Bash shell script.
2. This is a comment. It is ignored by the shell, but important for anyone trying to

understand what the script is doing.

3. The variable `guestfile` is set to the full pathname of a file called `guests`.
4. This line reads: If the file `guests` does not exist, then print to the screen "`guests nonexistent`" and exit from the script.
5. The built-in `printf` function dipslays only the filename (pattern matching) and the string "`non-existent`".
6. An environment (global) variable is assigned and exported.
7. A numeric expression uses the output of the UNIX/Linux `date` command to get the current hour. The hour is assigned to the variable, `Time`.
8. A Bash array, `foods`, is defined (`declare -a`) with a list of elements.
9. An integer, `n`, is defined with an initial value of zero.
10. For each person on the guest list, except the user `root`, a mail message will be created inviting the person to a party at a given place and time, and assigning a food from the list to bring.
11. If the value in `$person` is `root`, control goes back to the top of the `for` loop and starts at the next person on the list.
12. The mail message is sent. The message body is contained in a `here document`.
13. The integer, `n`, is incremented by 1.
14. If the number of foods is equal to the value of the last number in the array index, the list is empty.
15. The array called `foods` is reassigned values. After a message has been sent, the food list is shifted so that the next person will get the next food on the list. If there are more people than foods, the food list will be reset, ensuring that each person is assigned a food.
16. The variable `n`, which will serve as the array index, is reset back to zero.
17. This marks the end of the looping statements.