

Linux Hax

This started out as a general unix shell primer/reference page for noobs, inspired by a lot of emails written to help colleagues that were getting used to unix based operating systems. I tried to focus on features and cultural norms that are so fundamental to experienced users that they often dont mention them. I have seen this manifest in stark realities such as people that are unaware of tab-to-complete after an embarassingly long period of daily linux use. It has widened in scope a little to include brief tutorial on important basic userland tools, and reference notes for myself and other advanced users on things that see helpful but infrequent use.

Convention

This is notation and syntactic commonalities reflected in this document and others like it, not strict syntax but some generally unspoken entrenched cultural features that might confuse unix noobs when they take a look at a man page or a document like this.

- `<x>` - a common notation for unspecified parameters in unix man pages and such is surrounding a name or description of a quantity or string with the `<` and `>`. es: `ls <folder>`
- `<cmd> --help` - common, quite standard, basically all modern command line utils have this arg to give you a refresh on the syntax, args available. This is however, a feature of the package itself and only ubiquitous due to cultural convention and voluntary adherence thereof.
- `<cmd> --arg-name -a <VAL1> --arg2 <VAL2>` - it is very common for single character args to use a single `-` and multi-char to use two like `--arg`, and use another `-` to separate words. This is also cultural convention, command line args can be anything and are tokenized on whitespace. flag options like `-a -b` are generally commutative(order doesnt matter) and can be grouped. so `-ab` and `-ba` and `-b -a` would all be the same to the former. argument values (denoted `<VAL`>` and `<VAL2>` above, are not commutative and sometimes option flags are grouped with argument values so they are only commutative within that block rather than on the whole line.
- RTFM - means read the fucking manual IE check `man`, common use context is in a response to someone who wants to be spoon fed like a baby and cant read his own error messages... (you know who you are)
- `[expression]` - **same as** `<expression>` **described above, this is common in man pages**
 `-[expression] ...` - entire `[expression]` is repeatable
- `[ctrl]-x` - hold control and x both for a moment, `x-y z` hold x and y for a moment, release both, hit z
- `[BUTTON]` - hit a button labeled BUTTON on your keyboard

Basic general unix shell commands

- `ls` - **list files**
 - `ls -al` - list all files with extra information
- `mv` - move file/folder
- `cp` - **copy file/folder**
 - `cp -r` - copy folder recursively
- `pwd` - gives u the current directory, like which, full path
- `rm` - delete file
- `rm -rf` - remove folder recursively and force, IE, ignore all warnings. yes it will delete the whole drive if run as sudo on /

- `mkdir` - makes an empty directory
- `touch` - makes an empty file
- `rmdir` - remove empty directory only(safety feature)
- `less` - read a file that is longer than the screen. scroll by hitting enter, space, arrows, pgdown, search with / use q to quit
- `su` - setuser/superuser its supposed to stand for, su bob will make your user bob, you need his password. su makes u root(if you have a root password)
- `sudo` - run a command as root, became the normal way to do things in administration after a while. before it was just get a root shell with su. it will ask for a password and if you are an admin user yours will work. you have to be in the group sudoers.
- `more` - basically the same as less but slightly different in an unmemorable and barely perceivable way
- `top` - like task manager, list everything. full featured interface, can kill things, sort everything, etc `htop` is very similar with a more attractive ncurses interface
- **`rsync` - sync directories locally or over ssh or other transport**
 - `rsync -av --progress <folder1> <folder2>` - copy large folder with progress bar, preserve permissions
 - `rsync -avP --append-verify -rsh=ssh user@host:/path/ user@host2:/path/backup` - copy, omitting files with matching size that already exist in host2 at said path,
 - use `-c` to use a checksum instead of just file size compare. omit ssh args to use locally
- **`ps` - list processes defaults to ones in your shell**
 - `ps aux` - lists processes from all users with more information
- **`grep` - search files for string or regular expression, print whole line**
 - `grep -v` - exclude files
 - `grep -A n -B m` - print lines n after matching line and m before matching line
- **`kill` - end process with signal 15, smooth exit**
 - `kill -s 9` - end it right now, no shutdown sequence
- `cat` - spit entire file to stout
- `curl` - send http request and spit output to stdout
- **`nc` - netcat, same as cat but uses raw tcp socket. can work on udp too**
 - `nc -l <n>` - listen on on tcp port <n> , write received data to stdout, add `-u` for udp
- `sed` - more advanced regular expression oriented grep with in-place editing focus
- `awk` - similar to sed, complex grep type thing regexps in-place editing etc
- `perl` - a whole language like python, partially specialized for the tasks sed and awk do, can write one liners in shell. regexps
- **`chmod` - modify permissions, uses a number code of 3 digits or letter**
 - `chmod +x file` - set file to be executable
 - `chmod 777` - let all users read write and execute. don't do it
 - `chmod 666` - all users read and write,
 - `chmod 600` - your user can read and write

- `chmod 770` - owner user and owner group can read write and execute
- `man` - manual page, `man <command>` shows the page, it is the help files, it is the best reference for arguments of commands. YOU SHOULD REFERENCE THE MAN PAGE COMMANDS. it is the only source you need for these base commands u see here, and old software. it is not necessarily the best way to learn how to use vim.
- `screen` - make a new screen. `ctrl-a (release) d` detaches/exits from it, `ctrl-a c` closes. this is one way you run things in the background
- `tmux` - terminal multiplexer, lets you squeeze multiple terminals into one screen. like a super old school window manager
- `nohup` - precedes command and prevents hangup signals from hitting it so it will run until killed or closed from internal logic. alternative to `screen` for background process that will persist on logout
- `md5sum` - jsut called md5 on mac/bsd just does an md5 checksum hash of a file. for comparison of files of any size
- `sha256sum` - same as above with sha256 algorithm. also exists others.
- `who` - lists out the current logins/screens. shows u who is logged in(which users and where)
- `whoami` - tells u which user u are. used to check if you've successfully hacked things and became root. or in innocent shell scripts
- `lsof` - spit out data about various things going on with processes and devices and filesystem. example `lsof -i:8000` gives u info about proc using port 8000
- `lsusb` - list the usb devs. good to check if it can see a device
- `lspci` - same but for pci devices
- `sort` - sorts text file line by line
- `find` - for searching the file system. most stupid way can be done like `find . |grep filenameiwant`. recursive list of full dir tree is the default behavior
- `uniq` - deletes duplicate lines that appear next to eachother in text.
- `echo` - prints whatever is in its args to stdout
- `which` - gives total path to an executable in the shell path
- `strings` - spits strings out from binary file
- `hexdump` - spits out hex of a file
- `diff` - gives u the difference of 2(text) files line by line. yes this is where the term diff comes from in git repos etc
- `tar` - deals with tar archives. to untar a tar.gz `tar xvzf file.tar.gz`, for tar.bz2, `tar xvjf`
- `gzip` - compression. works on one file, takes input from file or stdout(!) good on text, fast
- `bzip2` - slower more intense compression
- `gunzip` - un-gzipps file
- `bunzip2` - unbz2 a file
- `zcat` - gunzip and contents to stdout
- `zgrep` - greps compressed data, IE same as `zcat <file> |grep <word>`
- `bzgrep` - grep a bzfile, handy, exists also `bzless bzcat bzexe...` same as with above
- `lsblk` - list block devices. handy to se drives that are not mounted
- `df -h` - lists mounted drives with size ad free space in human readable format

- `du -h` - check file size. it is recursive by default so it is good to set the max view depth with `-d 0`.
`du -h -d 0 file`
- `lsmod` - list kernel modules(generally are drivers), which are code that can be hotplugged into the kernel. this is used when troubleshooting hardware and driver issues
- `modprobe` - load up a module, they have a path thing built in so you can tab tab to see whats available
- `time` - TIMES A COMMAND in human readable down to ms
- `date` - the timestamp in a human readable format, can spit out other formats check man page
- `ln` - typically invoked as `ln -s <target> <link_name>`, which creates a symbolic link
- `fsck` - checks hard drives
- `fdisk` - partition hard drives
- `parted` - more up to date and full featured alternative to the archaic `fdisk`, graphical interface is `gparted`
- `testdisk` - advanced hard drive configuration, partitioning, analysis, forensic and data recovery tool. allows you to change things like logical sector size while `fdisk` and `parted` seem impotent to this effect
- `mkfs` - makes the default fs, `ext4` or whatever your system thinks is the default, for other fs do `mkfs.<x>` or `mkfs -t <x>`, examples for `<x>` are `vfat`, `ext2`, `ext3`, `ext4`, `exfat`, `xfs`
- `yes` - endless loop of 'y'... for dealign with annoying menus with the y/n? prompts using pipe
- `wipefs` - removed disk label
- `shred` - destroy files by writing random data to the location they were stored on disk(doesnt work on some filesystems) or write random data to a whole disk
- `cron` - service for running periodic tasks.
- `ranger` - file explorer command line tool. vim bindings, written in python. navigate filesystem in ncurses text interface
- `lfm` - shitty version of ranger seems really old
- `lf` - newer unfinished version of ranger lighter and focused on the use of external tools to open things, not in repos <https://github.com/gokcehan/lf>
- `head`- get top 10 lines of the file, use `-n` to specify numlines
- `tail`- some as above, last 10 lines as default
- `cut`- more general than the 2 above, check the manpage, cuts on chars, bytes, lines, delimiter separated fields....
- `fold`- chop up input from stdin and wrap it with newlines to enforce a certain width on text.
- `last` - show log of your users logins
- `lslogins` - list login statistics for all accounts
- `bc` - **basic calculator, supports arbitrary precision**
 - `echo 1 + 1 | bc`
- `tee <file>` - output stdin to stdout and to file

editors:

- `vi` - the old version of vim. it sucks. if u have a new install and type `vi` this is what is usually there. it makes people hate vim. dont use it. install vim and it will clobber the path to this

- **vim** - the new version of vi, if installed will alias as vi overriding above command, for serious people only. perfect for people that hate their mouse. extensible to the point of absurdity. it is a modal editor, meaning it has modes of interaction with the file. hit escape to disassociate from a mode, hit a letter to change to that mode. in this case the letter i is insert (normal edit mode), v is visual(select and delete copy and stuff large blocks to text). in the default mode and in visual d is delete, hit it twice to delete a line. visual mode d deletes selection. u is undo. the : char (yes use shift) lets u type in commands for user defined things and interactions with filesystem. :w is write. :wq is write and quit. :q is quit. q! is quit RTFN with no confirmation. :r <file> is read(a file and output it at current cursor position). :read !<commands> does the same for a shell command ! <cmd> opens the shell and hides the editor, returning when you exit
- **elvis** - this is another editor, a better version of vi, lighter than vim(if i remember correctly)
- **neovim** - a new and cooler vim that people who think theyre cool use. also has qt graphical neovim-qt, apparently feature-rich and more efficient cleaner codebase as it was written more recently
- **pico** - simple old editor not sure its ever used anymore.
- **nano** - a fork/copy/something of pico, newer, good for noobs, often used and well respected. commands are on the screen when using it and ctrl-X based.
- **emacs** - a complex and extensible editor, bulky for a command line utility. generally serious editor nerds that use stuff in this section use either emacs or vim, and have strong convictions about it.
- **ed** - the simplest editor from extremely long time ago, only used in extreme emergencies. the kind of editor a eunuch would use.
- **gedit** - simple graphical editor, good, basically notepad with syntax highlighting.

system things(debian based mint/ubuntu):

- **sudo** - run following command as root (admin)
- **su** - set user, defaults to root. can specify shell with -s
- **service** - control a service's ephemeral state and status check. service <name of it> <start, stop, restart, reload> ex: sudo service postgresql restart
- **systemctl** - controls systemd services state and settings. This includes everything that you can control with the above command, plus user services, startup behavior of system and user services. systemctl <start, stop, enable, disable, mask, unmask> <Service-name> covers most of the stuff you use
- **hostname** - prints hostname, if given arg it will set the hostname to the arg. if u do this, should also manually change /etc/hostname and make sure /etc/hosts reflects that change if necessary
- **adduser** -adduser <newusername> makes a new user. many options. none are really required, even a password. interactive walk through
- **useradd** - more l33t version of adduser. more useful noninteractively and non-user-friendly
- **usermod** - mod shell and stuff of a given user usermod -aG common for adding group
- **passwd** - password change, passwd <user> does it for user when u are admin
- **dd** - writes raw data. dd if=indevice of=outdevice bs=1M. if is a filesystem object to be read, of is the filesystem object to be written and bs is the block size which can be written human readable like 1M 2M 4M and in bytes like 1024(the old way). you use this when wiping disks with random data. you use it when 'burning' a flash drive with a disk image like dd if=linux.iso of=/dev/sdc bs=4M. If you mess up with this as root you can easily overwrite your hard drive. do not do it to mounted filesystem
- **chsh** - change the shell for a user
- **chgroup** - change group of file... group ownership

- `chmod`- change permissions of file `chmod 777 file` makes everyone read write ex it, `chmod 666` is read write for all.... `chmod 600` is another common one `ls -al` will show the perms
- `mount` - attaches a block device to a folder, allowing you to browse the filesystem
- `umount`- unmounts somethign takes mountpoint or `/dev /device` as target
- `dmesg`- prints messages generated at boot
- `env`- show ur environment vars, set them then run command(too)
- `uptime`- time up
- `wipefs`- removed disk label
- `cryptsetup` - setup luks volumes
- `cron`- service for running periodic tasks.

shells:

- `bash` - common, youre prob on it. "bourne again shell" whatever that means
- `csh` - different, advanced too - C shell
- `tcsh` - mac uses it? freebsd? its good too
- `zsh` - another shell that some nerds are all about, like the previous 2
- `sh` - the most simple bare bones one used when there is nothing else in some broke-ass embedded system or something, no tab to complete, no features, you run it because its always there on every system, common hack entrypoint to spawn a shell in a priv upgrade or somesort of remote code exe sploit

env vars:

the shell and other software uses many environment vars

these give background information about your system and things to software that needs it

this information is stored here because it doesnt need to be changed often, but always needs to be specified

type `env` to see them all. `echo $VAR` to see VAR. `export VAR=sgfsgs` to set VAR to sgfsgs for your session. setting `VAR=5 someprogram`, will modify VAR for that single line running someprogram.

shell vars in general have a `$` infront of them when you access them. but not when you set them

- `$PATH` - path to binaries, default is `/bin /usr/bin /usr/local/bin` etc
- `$DISPLAY` - x11/xorg display, typically `:0`. machines can have multiple displays, like all unix things, its multiuser
- `$PYTHONPATH` - where python looks for modules
- `$USER`, `$HOME`, - username and home directory path
- `$_` - last arg from previous shell command run
- `$?` - exit value/signal from prev command (0 if success which you manually throw in scripts with `exit 0`)
- `alias` - it is a command that tells the shell to make a macro for other commands, generally default `bashrc` will have some use of it and generally anything you want to do like this is done better with a function def
- `env` shows your env

- `export` - declare env var for remainder of session until u close this shell
- `jobs` - lists the jobs in shell(if you have paused with ctrl-z) with jobid
- `bg <jobid>` and `fg <jobid>` - background a paused job or foreground a paused job respectively.

strange obscure barely useful:

- `motd` - message of the day, displayed on login, not all systems have this command, its old school, but having an MOTD is not a dead art.
- `links` - text only browser
- `lynx` - older more useless text only browser
- `irssi` - irc client ncurses flavor. leet af only good program in this section
- `rexima` - command line sound volume control mixer thingy
- `beep` - makes a console beep

graphical, featureful

- `xterm` - old school bare bones terminal emulator for x11
- `xorg/x11` - always started by scripts, but it is the name of the service that runs the GUI in linux generally. x1 was the old name xorg is the new one. there are forks...
- `xv` - old and simple image viewer. seems to be somehow replaced by `xviewer` and some systems may have it as `xview`
- `mplayer` - old simple and great media player. no GUI, just do `mplayer file.mp4` or whatnot
- `mpv` - like `mplayer` but better, has no interface other than key bindings and cmdline
- `gimp` - powerful image editing, old school MIT project, shit interface, opens any format basically
- `ibus` - this is a package for controlling advanced input methods that are a lot more than a change of layout; like Chinese, Korean,
- `display` - another nice CLI for `imagemagick`. functionally same/similar to `xviewer` only it will take input from STDIN which is great.
- `librewolf` - probably best browser at time of writing this, firefox with telemetry removed and other security enhancements
- `zathura` - -good pdf viewer, cool kids use it these days, suckless minimalist
- `xpra` - like screen but for graphical apps. useful for video editing on a server with a big GPU remotely. normal x forwarding over ssh just forwards the X11 instructions and renders on the client, but this can render on the server and compress it, send it to you as a video stream.

crypto

- `gnupg` - `gpg` a gnu implementation of `pgp` aka 'pretty good privacy' the first common userland well adopted implementation of modern cryptographic protection, mainly for emails and the like. has `rsa` and the like, `MAC` methods and all that. as per gnu naming conventions, its name is a goofy acronym based pun of sorts.
- `cryptsetup` - setup luks volumes. rtfm on it
- `openssl` - CLI for `openssl` library functionality, very handy for some specialty tasks, generating keys and hashking things

- `pass` - password manager that uses gnupg. integrates with git, can be used to run google auth type 2fa, responds to tab to complete well. extensible with plugins. basic commands are `pass insert`, `pass show <name>`, `pass edit <name>`. initialize with `pass init` after making a keyring with gnupg
- `openpgp-tool` - openpgp smartcard device control.
- `fido2-token` - manage, manipulate fido2 security keys

network & hax

- `nmap` - port scanner highly advanced, many modes and options
- `masscan` - speed optimized port scanner for large volume scanning, target acquisition. usually preceeds the use of `nmap` which yields more detailed information
- `nc` - previously mentioned, netcat, raw conns `nc <host> <port>` does tcp conn. `-u` arg does udp and `-l` is listen
- `ettercap` - manipulation of ARP, DNS, other protocols, generally for the purpose of man in the middle attack. it is bad to the bone, it is a cyberweapon
- `wireshark` - watch network packets go by. need to change group to work properly. can run as root and always works that way, but not recommended. used to be called `ethereal` - the new name sucks. still hate them for it. the new name reads like it should be the name of a chinese electrician tool or a korean children's cartoon
- `ngrep` - network grep, just reads packets going by your box and spits that out to stdout if it matches what ur looking for
- `tcpdump` - captures and dumps packets, dump files can be reloaded, minor dissection available with some classification, can load the dumps up with anything
- `ifconfig` - old network interface config command line utility. windows `ipconfig` is the ripoff version with a weird name
- `ip` - the newer, 'better' network interface and routing table configuration tool
- `route` - routing table edit and explore
- `htping` - sends a http packet to a server on default port of 80, gives response time
- `ping` - normal old school icmp ping. not what it used to be
- `telnet` - old school shell/terminal over the wire. completely unencrypted, not much more complex than netcat. helpful for testing connections, manual single port probing like `tenet <host> 80` to connect to port 80 on <host>
- `nslookup` - look up an ip or hostname in DNS
- `john` - old school powerful password hash cracker. supports extensions and a lot of hash algorithms. parallelism exists too, not sure about GPU kernels. likely better things these days. called john the ripper(after the famous amateur serial hooker-vivisection enthusiast)
- `whois` - information on domain ownership, reverse look up of IP addresses. just an entry from a database about the owner and registrar stuff for IPs and domains.
- `traceroute` - old school packet routing trace, not sure if it really works the same anymore, but shows you the path packets take to a server. seems like maybe routers out in the wild drop the packets it uses now often? not sure. don't use it much and it's not what it used to be is the word
- `arping` - executes a ping-analogous function using the arp protocol. v nice.
- `tssocks` - wrap any protocol through socks generally config in etc
- `htping` - ping a http server. IE, give the response time to a http service

- `aircrack-ng` - a suite of utilities for security analysis of wifi networks
- `iwconfig`-ike `ifconfig` but with specific features for wifi adapters/driver interfaces. it is old school
- `iw` - same as above but not as old school
- `bluetoothctl`- shell style interface to bluetooth hardware. quite good
- `yersinia`- a powerful security analysis tool that i am not too familiar with, but worth a mention. some kid in vegas looked at me like i was insane for not using it. appears very powerful.
- `netstat`- usually i invoke as `netstat -n`, lists the connections in and out of the machine. godo stuff is by the top so try `netstat -n|head`
- `fido2-token` - manipulate and probe fido2 auth tokens such as yubikey etc
- `opensc-tool` + `opensc-explorer` - cli util and interactive shell interface for smart card interactions a-la iso7816 and iso14443(contact chip and nfc interfaces respectively)
- `pcsc_scan` - report basic diagnostic info on connected smart cards

services

these are the names used if you were to `service <name> <start|stop|status>` services are started stopped etc by scripts which are used by `systemd` and this command or in general your setup might use a different service manager, which will be similar. This is because some services need a sequence of commands and checks etc before starting or stopping safely.

- `fail2ban` - great utility that watches update of logs from whatever you want and responds to predined events (you set up in `/etc/fail2ban`. modularized to actions filters and jails. where actions are responses, filters define events and jails define groups of events and how they trigger actions and expire. all bans are cleared on restart by default.
- `nginx` - nice simple lightweight webserver, often used as a proxy to a web app run with python-flask or similar, to provide robust features that come with a real web server.
- `snort` - network util for traffic capture and parsing, logging. can be run in the background as a system service to construct intrusion detection functionality, or used like `ngrep`

SSH STUFF

- `ssh <user>@<remotehost>` - secure shell, replaced telnet when people realized doing password based auth and all your work over cleartext in telnet was retarded and more dangerous than working in a liberian brothel
- `ssh-keygen` - generates keypairs for ssh auth - `ssh-keygen -lf .ssh/id_rsa -E sha256` - generate fingerprint of key -
`ssh-keygen -t ed25519-sk -O resident -O application=ssh:<description> -f ~/.ssh/id_ed_sk` - generate key on fido2 token as resident on key, type can alternatively be `ecdsa-sk`, omitting `-O resident` makes a key that requires the fido token but is not stored on it. not discoverable from the key. `-O verify-required` or `-O no-touch-required` control the physical prescence requirements(touching the key) - `ssh-keygen -K` - importing resident keys to new machine from security token
- `ssh-add -L` - print all your public keys in `.ssh`
- `scp localfile <user>@<remotehost>:/path/file` - copies files over ssh bidirectionally, will default to copy locally for composibility/compatibility and uses same args generally, which must be before the locations provided. typical use `scp user@host:/home/user/stuff stuff`. username is often needed. tab to complete works if you have passwordless ssh set up. USE IT PASSWORDLESS AND USE TAB. tab is slow though(it must open auth and close a ssh session in the background silently to achieve this). remember you can copy to `/tmp` always, too, if perms are giving you grief.

- `ssh -X <remotehost>` - this arg will forward x11, IE, let u run graphical programs over ssh (if u have x11 on both sides) `ssh -Y` is equivalent but was meant to be a more lightweight connection
- `ssh -A <remotehost>` - forward ssh agent to foreign server, allowing secure access to local keys on foreign server, including hardware tokens
- `ssh -D 8888 <remotehost>` - runs a socks5 proxy on port 8888 that tunnels connections from localhost through the remote host
- `ssh -L<bindaddress>:<listen_port>host:<port> user@remotehost` - tunnel localhost lport to remote host's view of host:port
- `ssh -R<bindaddress>:<lport>:host:<port> user@remotehost` - reverse tunnel, goes from remote host to view of host:<port>
- `sftp` - ftp style shell client for scp-like and other extended functionality
- `sshfs` - smount - use the above sftp facilities to emulate a mounted filesystem
- `ssh-copy-id`, `ssh-keyscan`, `ssh-agent` - other useful key management tools

operators in shell(bash)

- **| pipe, puts stdout into stdin like `ps aux | grep <word>` looks for <word> in output of `ps aux` (list of running processes for all users)**
 - `ls | tee bob` - example use of `tee`, this will write the directory contents to file `bob` while outputting them to stdout as well
- `&` runs concurrently with following command.
- `&&` run next program sequentially, if the first succeeds
- `||` run command after only if the previous command fails
- `>` stdout into a file `cat bob > file_name`. OVERWRITES THE FILE
- `>>` APPENDS TO THE FILE like `ls >> listfile` will append contents of current directory to file `listfile`
- `2>` same as `>` but does stderr, where `1>` is just the default that `>` alone reverts to
- `&>` - writes both stderr and stdout to filename after it
- `<` file on right into stdin of command on left
- `<<<` string on the right into stdin on the left
- `ctrl-z` pause - immediate effect always
- `ctrl-c` exit, doesn't leave shell (that's logout) clears the line though. sends a `kill -s 15` to the thread in foreground
- `ctrl-d` logout
- `[TAB]` tab - hit this key a lot, it works to complete MANY things. used to just be files, now almost anything. `git add [TAB] [TAB]` lists your changed files, for instance
- back quotes - `kill `pgrep firefox`` - inserts stdout from the command in backquotes into the shell as if you had typed it. `pgrep` outputs a list of pids that match the string you give it, here that is being picked up by `kill` so that it kills anything that matches `firefox`
- `*` wildcard, `ls *.py` gives list of python scripts in current directory
- `!!` the last command, `!n` nth command in history, `!-n` n commands back, IE `!-2` executes second last
- `!*` args from previous command

- `[0-9]` matches digits in shell, `ls [0-9]*` list everything that starts with a digit. can use comma separated singletons, works with letters too `[a-z]`...

patrician word processing

if youre writing text then please be a man and use a text based interface

- `latex` - compiles to dvi, pics gotta be eps(a vector format)
- `pdflatex`- compiles latex pics must be png and jpg i think. cant use vector format eps
- `htlatex`- good compiles latex to html with pics for equations and other floats
- `latex2html` - sucks. honorable mention thought
- `dvipdf` - turn dvi to pdf common for use of `latex`
- `rst2html` - restructurtd text to tml
- `rst2latex` - restructurtd text to latex
- `rst2man` - restructurtd text to man page
- `rst2odt` - restructurtd text to odt
- `rst2pdf` - restructurtd text to pdf
- `convert` - very smartly interfaced command line front end for imagemagick. just convert `bob.<ext> bobout.jpg` etc to convert between any image format. this is helpful for latex etc.

root filesystem synopsis

Int the past many of these were separate partitions, hence some of the seemingly redundant things. Now this is not as important with solid state drives and (i supposed) more modern file systems

- `/tmp` - temp folder, anyone can write in it. it is there on every system and great place to copy things to if you are not sure where to do it
- `/etc` - pronounced et-SEE. all the configuration files and global settings are in here by default. in the past administration could be done exclusively by modificaion of files here, more or less. programs like `passwd` and `usermod` are tools to automatically edit files here. Disk usage is small as it is mostly text files and it is definitely something you want to back up, as it contains any system settings you took time configuring.
- `/var` - various data here, `var/log` is a default global spot for logs. often home to global data storage, such as the root of a webserver with static content, or database disk footprint.
- `/usr` - user installed things generally.... comes with a lot in it these days. it is like an alternative root where u generally would modify things for system wide access. has the same directory structure as `/`
- `/proc` - process information emulated as block storage devices and stuff like this. can get info about some hardware from drivers, and access some other kernel level information pertaining to active system processes
- `/sys` - kernel emulated filesystem tree allowing information and interaction of various kernel level functionality and hardware devices. This includes the ability to read parameters from live kernel modules and set them by writing to said file as well, for example.
- `/run` - contains filesystem socket devices and other quasi-file dynamic objects written by userland software (as opposed to kernel level features in `proc` and `sys`)
- `/dev` - devices, filesystem emulation of actual hardware. all disks are here, your sound devices, usb devices, all accessed from here if you want to do it directly. it is a virtualized/emulated filesystem integrated representation of a group of non-file objects(very cool) like `proc`. these are not actual files,

but dynamic emulated files that make access to devices like accessing a file. reading and writing to them is the same as a file

- `/opt` - not sure what it is supposed to be but it is often used to store globally accessed proprietary software that doesn't have facility to install in the typical global directory structure (where things are in `/bin` and `/lib` and `prstuff`)
- `/bin` - binaries, these are where the commands are stored for the base system. most of the higher level stuff is in `/usr/bin` and `/usr/local/bin`
- `/home` - home directories for each user here. all user settings and information and data are in their home folder. copy it to a new system and it will all be there
- `/root` - home directory for admin/root user
- `/boot` - contains the kernel and initial root disk, boot loader stuff IE GRUB. is more commonly a separate partition still
- `/cdrom` - vestigial artifact of a time when people used cdrom
- `/mnt` - this was originally where you would mount drives, IE, any drive that was not hosting system critical contents, like removable media, was mounted here. you added these to be automounted using `/etc/fstab`, and mounting had to be done by root
- `/media` - this is where things are mounted now, in a path like `/media/<username>/<uuid serial thing>`, this is now handled by some daemon that will do it for you as a `setuid-to-root` binary or something, to streamline the process of using removable media since the proliferation of USB storage devices (previously portable storage media didn't carry its hardware interface with it, so the system wouldn't see new media as a new device entirely, but a change in state of a known device)

notable filesystem objects, global

- `/proc/cpuinfo` - cpu core info, pretty great
- `/dev/random` - random data from hardware. cat this and u get a dump of real physical entropy
- `/dev/urandom` - output of a prng using above as seed. cat this and get infinite 'random' data generated from finite entropy harvested from hardware
- `/etc/passwd` - old school place where some user info is stored, originally included encrypted passwords. now it is where you go to look up info like groups and home directories and shells quickly. each line is a user and all of their `chsh` / `usermod` related properties
- `/etc/shadow` - where they moved the encrypted passwords and put them as only `r/w` by root and `r` group shadow from `passwd` to hide them from users when it was realized they could be cracked
- `/etc/hosts` - list of hosts that are basically added to DNS, can put some of your servers here so u don't type ip
- `/etc/hostname` - your hostname, for some reason i feel i usually must edit this and use the `hostname` command at the same time/session
- `/etc/rc.local` - old school place to put commands to have them run on boot, on many linux systems.
- `/etc/resolv.conf` - old way of keeping global nameservers. depends on the system now. In theory you can just add lines to add hosts but generally there is some crackpot software stack hiding behind a local service that this file points to. way to make something overcomplicated.
- `/etc/motd` - text displayed at login. put stuff here if you have users, info about the system, advertisements, cuss them out, etc

notable filesystem objects, local

- `~` - alias to your homefolder `/home/username`
- `~/.ssh/authorized_keys` - put in a copy of someones `id_rsa.pub` file as a line, and it allows anyone with the corresponding private key to log into said account to whom `~` belongs.
- `~/.ssh/config` - its a preconfig defaults for various servers and things, pivotal when using `scp` and `git` regularly. `man ssh_config` exists and shows syntax
- `~/.ssh/id_rsa.pub` - default place for public ssh key, without the `.ssh/id_rsa` is default for private, which, should be `chmod 600` for the perms
- `~/.bashrc` - if you use `bash`, this is a place you can add commands that run on login. such as adding things to your `$PATH`
- `~/.bash_history` - history of commands in `bash`, some cap length by default, `grep` this to find stuff you did and need the command for
- `.profile` - this is like `.bashrc` but not specific to `bash`. on many systems, `mac OSX` and i believe other `BSD`. definitely check if you are not using `bash`
- `~/local/` - has a root filesystem mirror structure that user installed things (like `pip` packages) can sit in. like a personal `/usr/local`. `pip` user installed stuff goes here
- `~/config/` - it is now considered best practice for packages to put their user config files in here rather than randomly as a hidden file or folder in `~`

some good config file lines

`~/.ssh/config` This is an import config file, sometimes it is absolutely necessary if you are using `scp` and other `ssh` based utilities like `git` that sometimes do not have the ability to take the more advanced arguments you may need to give them, in the case of having multiple users at the same host with multiple keys and things like this

```
>>>
Host bob
  HostName bob.com
  User userb
  IdentityFile ~/.ssh/id_rsa_bob
```

this enables you to simply `ssh bob`, and `tab` to complete works on this alias for the host. `HostName` is a misleading label, as it is the actual network address, `dns` or `ip`, and the alias you are giving it which will follow this setup every time is the first line in each entry `Host`. these aliases carry over to `git` commands and `scp`, etc

host a git, barebones

simple and dirty instructions always use passwordless `SSH` for this make `git` user on server. NO PASSWORD ON IT. no way to log in with password, furthermore, use `git-shell`

```
>>>
sudo useradd -s `which git-shell` git
sudo su -s /bin/bash git
mkdir <package-name>
cd <package-name>
git init .
git config receive.denyCurrentBranch ignore #lets you push to bare repo
```

put public keys in `/home/git/.ssh/authorized_keys` as a line, on the host `n`

on clients: `git clone ssh://git@server:/home/git/package`

then make an initial commit to master to make sure it works

pull requests are a social media feature tied to the web interface and dont really exist in this setting.

git client side

process of creating branch and merge:

```
>>>
git checkout master
git pull # make sure its up to date
git branch mybranchname # make a branch
git checkout mybranchname #- now you are on it, it is forekd off main
#do stuff, write code
git add stuff
git commit -m"new stuff"
git push #- upload it to the remove server
#keep doing stuff, eventually ready to merge
git checkout master
git pull #-make sure its up todate
git merge mybranchname
#now if theres conflicts, you make sure it works, correct them.
#you can checkout a file from master by "git checkout <branch> <file>" to overwrite your version with one from another branch
git push
git branch -d mybranchname #delete the branch that you merged in, keep it from cluttering repo
```

git is very user friendly for a command line interface, gives useful messages but remember to push after you merge, push and pull and clone are remote commands. commit, checkout, merge, etc, are local manipulation and interfacing with the underlying repo datastructure that is entirely local. Git can be useful without a remote, just to track progress and allow you to undo things if you mess up your code. noobs and people in the past that didnt have version control used to keep many copies of their code.

docker

docker is super helpful, especially if youre a noob. It allows you to do things as root but not destroy your baremetal system.

It was originally to make back end services scaleable, reproducible, and sandboxed while avoiding the use of a VM. apps in docker run on your kernel but network and disk is sandboxed and communicates through whatever avenues you specify(shared folders and port forwards). you can run things in docker seamlessly, including graphical interfaces. its a good way to silo sketchy ass commercial spyware-riddled-packages. good way to keep reproducible devleopment environments to remove variation between peoples systems on a dev team. it has a built in management system for images shared by project teams and the community.

if you dont use it youre basically failing at life. It is not something that requires a ton of knowledge or practice to benefit from.

to get started you need to add user to docker group `usermod -aG docker <user>`, and then make a empty directory and put a file in it called Dockerfile, in which you list a series of commands building your custom system, generally starting with something from the docker repo. example including most of what you need:

```
>>>
FROM ubuntu:latest #start with the baseline latest image
RUN apt-get update
RUN apt-get upgrade -y --force-yes
RUN apt-get install -y --no-install-recommends <packages> #only install the requirements and avoid any extra dependencies
RUN groupadd -g 1000 ubuntu
RUN useradd -d /home/ubuntu -s /bin/bash -m ubuntu -u 1000 -g 1000
USER ubuntu #rest of lines are as this user, as is runtime(default is root)
ENV HOME /home/ubuntu #set environment variable $HOME
RUN apt-get clean
#clean up, rm -rf basically anything you dont need to run the entrypoint
WORKDIR /workspace #in this file after this command and at runtime launch we are in /workspace
CMD <command> #whatever you put in for <command> will be the default entrypoint
```

then build with `docker build` and run with `docker run` with appropriate settings for network exposure and volume sharing etc.

- `docker-compose` - utility for launching a few different docker containers of different services, allowing you to easily config them to be interconnected in one file. simply put `docker-compose.yml` in an empty folder and edit/generate/write it to your specs. editing yaml can be kind of annoying due to autistic standards with whitespace and stuff. so work off of a cypypaste
- `docker` - the normal interface to docker to run one container
- `docker stats` shows current running containers with resource use. important for noobs because people forget and leave them running
- `docker <obj> prune- <obj>` may be container, image, volume, network and maybe others i forget. this deletes the unused objects of said type, freeing up space.

`-it --name box0 --device /dev/snd -v /etc/file:/etc/file:ro -v ~/stuff:/etc/stuff.d --net host image`
 left to right: run, remove when done, interactive session(dont run in background like nohup), name box0 on the running container, share /etc/file respectively in container, mount folder ~/stuff to /etc/stuff.d , share same network as host, run latest version of imagename, use

DONT

- *DONT* store data in a docker container. you store that in volumes or shared/mounted directories on host filesystem
- *DONT* try to keep persistent systems in docker, it is better to always `docker run --rm` to auto remove the container when you are done, and any changes that were needed should go to the Dockerfile. any config files and things should be in shared directories, safely stored on the host. containers should always be reproducible by automated build process defined in the Dockerfile
- *DONT* not run `apt-get clean` in Dockerfile. look for other things to delete too. ideally you make a second container from a lighter cleaner image and copy over the things you set up, leaving behind everything else
- *DONT* not use `apt-get --no-install-recommends`
- *DONT* forget `DEBIAN_FRONTEND=noninteractive apt-get -y <pkgs>`