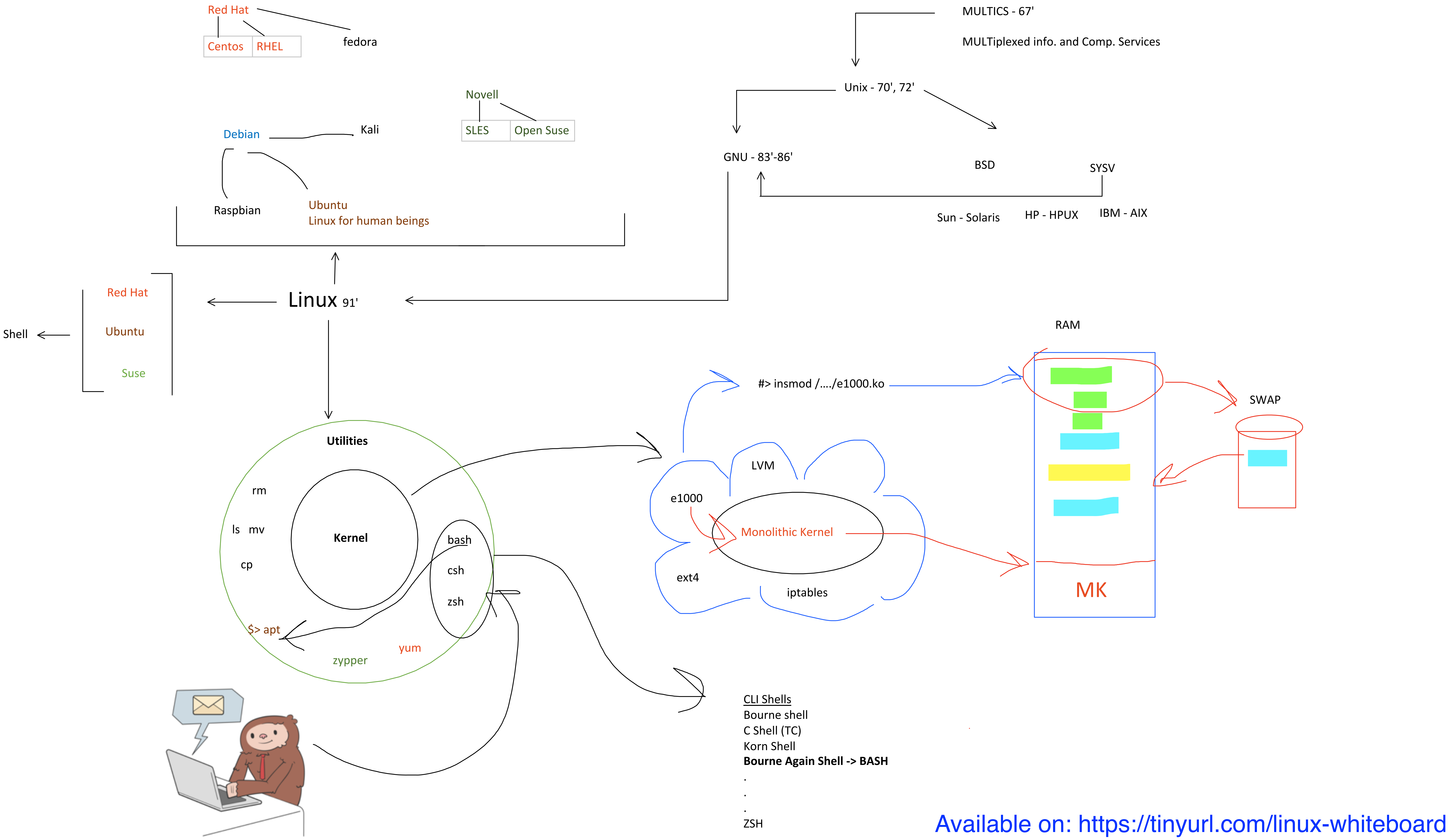


Red Hat

Centos | RHEL

fedora

Novell

SLES | Open Suse

Debian

Kali

Raspbian

Ubuntu
Linux for human beings

MULTICS - 67'

MULTiplexed info. and Comp. Services

Unix - 70', 72'

GNU - 83'-86'

BSD

SYSV

Sun - Solaris

HP - HPUX

IBM - AIX

Linux 91'

Red Hat

Ubuntu

Suse

Shell

Utilities

rm

ls mv

cp

Kernel

bash

csh

zsh

$> apt

zypper

yum

LVM

e1000

Monolithic Kernel

ext4

iptables

#> insmod /..../e1000.ko

RAM

MK

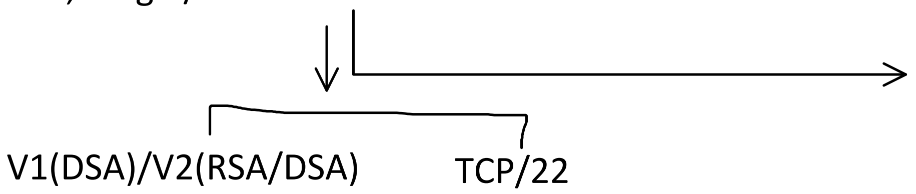
SWAP

CLI Shells
Bourne shell
C Shell (TC)
Korn Shell
Bourne Again Shell -> BASH
.
.
.
ZSH

Available on: https://tinyurl.com/linux-whiteboard

Connecting to a Linux machine - The Login Process

1. Method of connection:
    a. Console --> No network option (No network module [e.g., smoke detector], maintenance mode)
    b. Network --> Telnet, Rlogin/RSH… SSH

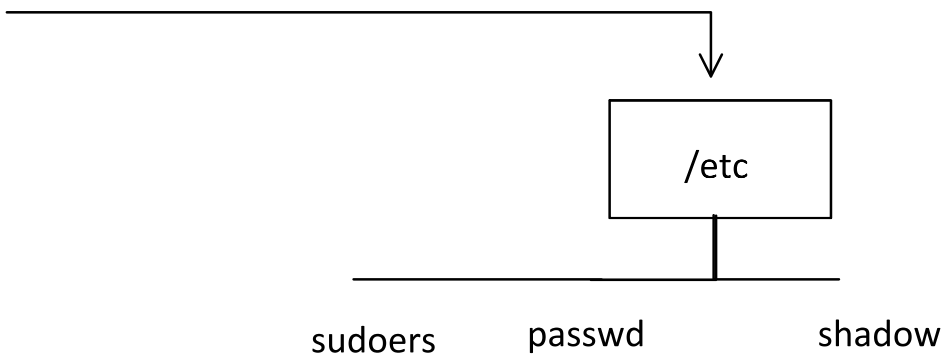


V1(DSA)/V2(RSA/DSA) TCP/22

Server: openssh-server

Client:
    Windows: PuTTY, MobaXTerm, Gitbash, openssh-client…
    Linux: openssh-client…

2. Credentials --> User and Password
    a. User types --> root, all other users…
    b. User configuration



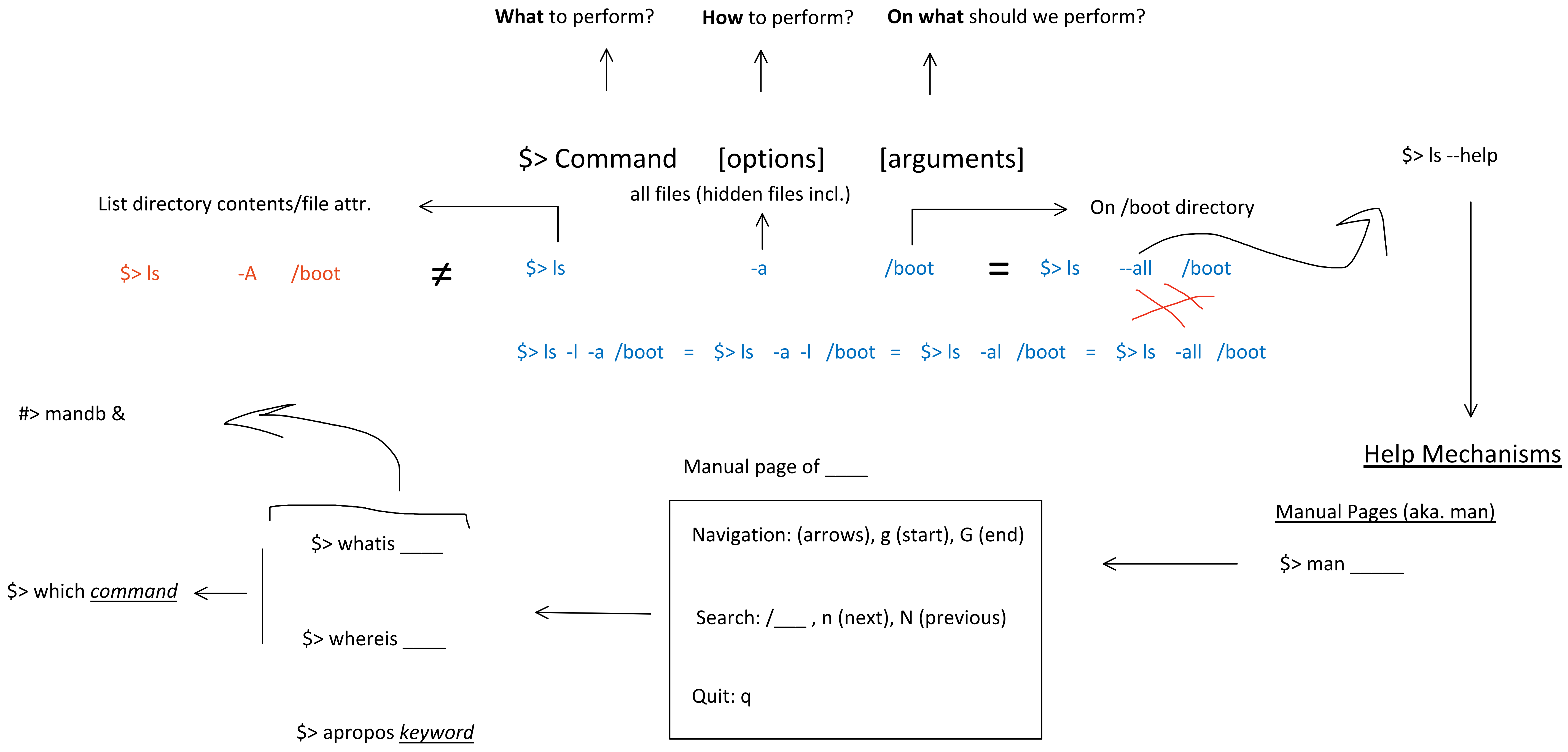
/etc

sudoers passwd shadow

3. Complementary actions:
    a. Execute user's default shell
    b. Enter user's home directory

4. Using commands…

Basic command structure



What to perform?

How to perform?

On what should we perform?

$> Command [options] [arguments]

List directory contents/file attr.

all files (hidden files incl.)

On /boot directory

$> ls -A /boot ≠ $> ls -a /boot = $> ls --all /boot

$> ls -l -a /boot = $> ls -a -l /boot = $> ls -al /boot = $> ls -all /boot

$> ls --help

Help Mechanisms

Manual Pages (aka. man)

$> man _____

Manual page of ____

Navigation: (arrows), g (start), G (end)

Search: /___ , n (next), N (previous)

Quit: q

$> whatis ____

$> whereis ____

$> which *command*

#> mandb &

$> apropos *keyword*

Basic command structure - http://tinyurl.com/y99pnsef

A - ls exercise

--

1. View /boot directory's content
2. View a detailed list of both regular and hidden files
3. View the list of all files sorted by modification time
4. View content of all directories and sub-directories (and so forth..)

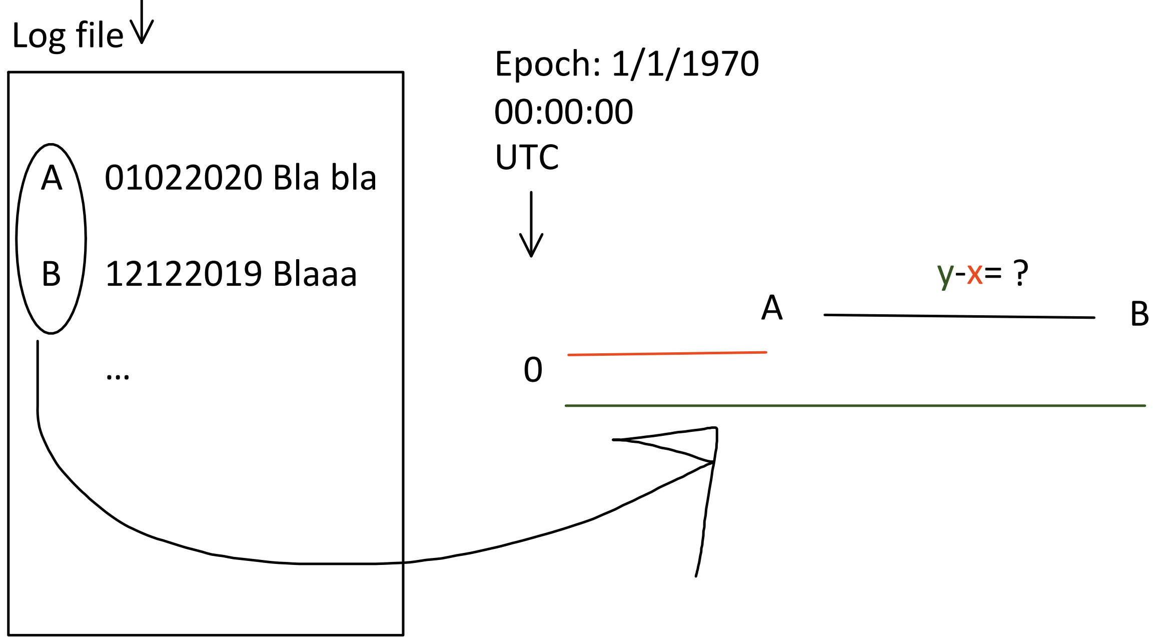
B - date exercise

--

1. Display the current time in the format HH_MM_SS (e.g. 15_50_17)
2. Display the current date in the format ddmmyyyy (e.g. 03052013)
3. Display the date of last Friday in the format of ddmmyyyy
4. Display the full date and time as it were 127 hours ago.
5. Display the number of seconds passed since 1.1.1970 until two days ago.

Solution →

```
cd
ls /boot
ls -al /boot
ls -alt /boot
ls -R /boot

date +%H_%M_%S
date +%d%m%Y
date -d "last friday" +%d%m%Y
date -d "127 hours ago"
date -d "2 days ago" +%s
```



Log file

A 01022020 Bla bla

B 12122019 Blaaa

...

Epoch: 1/1/1970
00:00:00
UTC

Basic filesystem commands

File types:

Common:

-	file
d	directory
l	soft link (symbolic link) --> Shortcut

Devices:

b	Block device
c	Character device

IPC (inter-process communication)

p	pipe (FIFO)
s	socket

File (of file type file) Classes:

.mp3
.txt
.log
.sh
...

file.txt

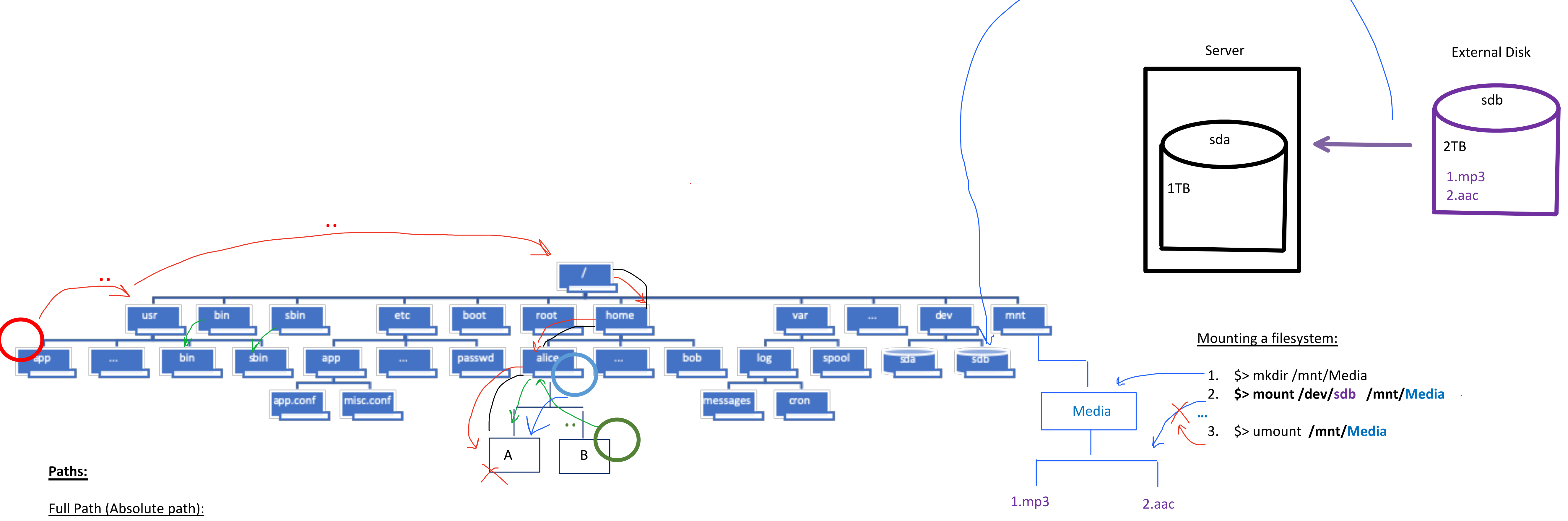
```
date
echo "Hi"
Happy Birthday
```

$> ls file.txt   => file.txt

$> cat file.txt =>	date
	echo "Hi"
	Happy Birthday

$> bash file.txt =>	11052020
	Hi
	bash: Happy: command not found...

The Linux file tree (directory tree - aka. file system)



Mounting a filesystem:

1. $> mkdir /mnt/Media
2. **$> mount /dev/sdb /mnt/Media**

...

3. $> umount **/mnt/Media**

Paths:

Full Path (Absolute path):

```
$> ls /home/alice/A
$> ls ~/A
$> ls ~alice/A
```

Relative Path:

```
$> ls A
$> ls ../A
$> ls ../../home/alice/A
```

Basic filesystem commands:

Filesystem navigation:
pwd
ls [-altRdh...] [path...]
cd [path... | - | ~-]

Copy, Move (rename), Remove (delete):
cp [-ifr...] source1 ... destination
mv [-if...] source1 ... destination
rm [-ifr...] files to delete...

-i interactive response (ask before removing)
-f force (never ask, never print errors)
-r recursive (apply to subdirectories and so fourth...)

Empty directories:
mkdir [-p...] newdirs...
rmdir empty-dirs...

Reading files:
cat file1...
less file1... (an advanced *more* tool)
head [-*number*] file1...
tail [-f -*number*] file1...

Creating text files:
touch file1...

CMD > file

e.g. echo A > file

ls >> file

> file

file

A

A
<ls output>

Filesystem commands - http://tinyurl.com/y99pnsef
----
1. Create the directory Dir in your home directory, and in it the following: (the word dir instructs you to create a directory)
    file1 file2.txt File3.c .file4 my_file. my_dir dir1
2. Create the directory Folder in your home directory and in it the following: (Without entering the directory)
    My_file .file.hidden file5
3. Create the directory scripts under the directory Folder and within create the file script1.temp with the following lines:
    a. '#!/bin/bash' (notice the single quote and not ")
    b. "#This file is soon to be a bash script"
4. Create under scripts the file script2.temp with the following lines:
    a. date
    b. ls -l
5. Print both scripts' content to the screen
6. Concatenate both scripts to a single file named script
7. Under scripts directory create a new directory called temp and move all temp scripts to it.
8. Rename the file script to script.sh
9. Create a backup file to script.sh called script.bak
10. Copy the scripts directory to your home directory
11. Remove the scripts directory from the Folder directory
12. Execute script.sh using its full path (i.e. /home/<username>/scripts/script.sh) - (notice it should fail you)
13. Execute script.sh using the command bash before its full path (i.e. bash /home/<username>/scripts/script.sh)
14. Explore du command through the manual pages and view the disk usage of your home directory and its subdirectories (only 1 level beneath it)
15. Create the file no-underscore under the directory my_dir (which is under ~/Dir)

```
mkdir -p Dir/{dir1,my_dir}
touch Dir/{file1,file2.txt,File3.c,.file4,my_file.}
mkdir -p Folder
touch Folder/{My_file,.file.hidden,file5}
mkdir -p Folder/scripts
cd ~/Folder/scripts
echo '#!/bin/bash' > script1.temp
echo "#This script is soon to be a bash script" >>
script1.temp
echo date > script2.temp
echo ls -l >> script2.temp
cat script1.temp script2.temp
cat script1.temp script2.temp > script
mkdir temp
mv script1.temp script2.temp temp
mv script script.sh
cp script.sh script.bak
cd ~
cp -r Folder/scripts ~
rm -r Folder/scripts
~/scripts/script.sh
bash ~/scripts/script.sh
du -h --max-depth=1
touch Dir/my_dir/no-underscore
```

Wildcards (FNG - Filename Generation)

a
b
c

$> rm *

1. BASH (*) = a b c
2. rm a b c

Wildcard symbol	Meaning
*	All strings (including empty string)
?	All characters (only one)
[Ax5@*],[a-zA-Z0-9]	All characters from the list (only one)
[!a-p] ~ [^a-p]$^{regex\ syntax}$	All characters not in the list (only one)

$> rm *.?

$> rm [a-z].*

$> rm [!a-z].[!0-9]*

.a			
a			
a.		V	
a.c	V	V	
a.log		V	
az.log			
1.			
1.h	V		V
1.3gp			
1.txt			V

[a-z] ≠? a,b,c,...z → [[:lower:]]

a,A,b,B,c,C...z,Z

[A-Z] ≠? A,B,C...Z → [[:upper:]]

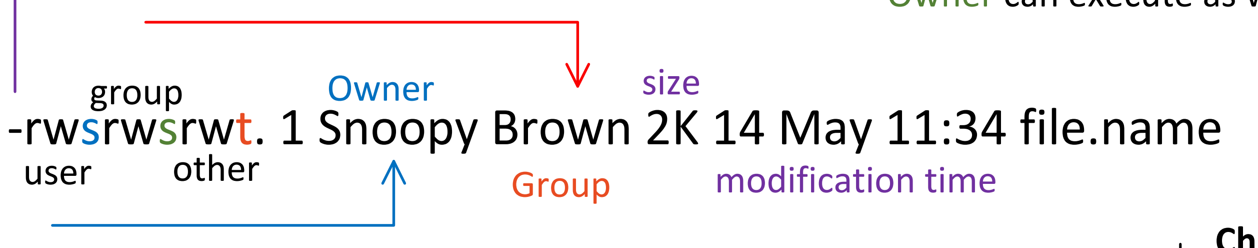
→ $> man 7 glob

[a-zA-Z] ~ [[:alpha:]] = [[:lower:][:upper:]]

Wildcards
----
1. Create the directory "files" under your home folder, and copy to it all files under "Dir" that starts with a letter
2. View using ls the files (or directories) whose name answers the following:

Permissions

Type:
d directory
- file
l soft-link
...



```
-rwsrwsrwt. 1 Snoopy Brown 2K 14 May 11:34 file.name
```

(r) Read - Read the content of the file/directory

(w) Write - Edit the content of the file/directory

(x) eXecute - Execute the file / Traverse (enter) into directory

Changing ownership:

Only root can exec.

```
$> chown user files...
$> chown user:group files...
$> chown :group files...  ~  $> chgrp group files...
```

Owner can execute as well

Default permissions:

```
       777
     -
$> umask 27
    ------
       750
```

Maximum allowed default permissions

Changing permissions --> Changing mode (chmod)

chmod -> Symbolic syntax **/ Numeric Syntax**

s s t
4 2 1
r w x

```
$> chmod [ugoa] {+|-|=} [_rwxs ] files...  /  $> chmod [0-7][0-7][0-7]0-7 files...
```

special, user, group, other

```
$> chmod u=rwx,g=rx,o=r file.name   ->   $> chmod 754 file.name
$> chmod g+w,o-r file.name          ->   -
$> chmod = file.name                ->   $> chmod 0 file.name
```

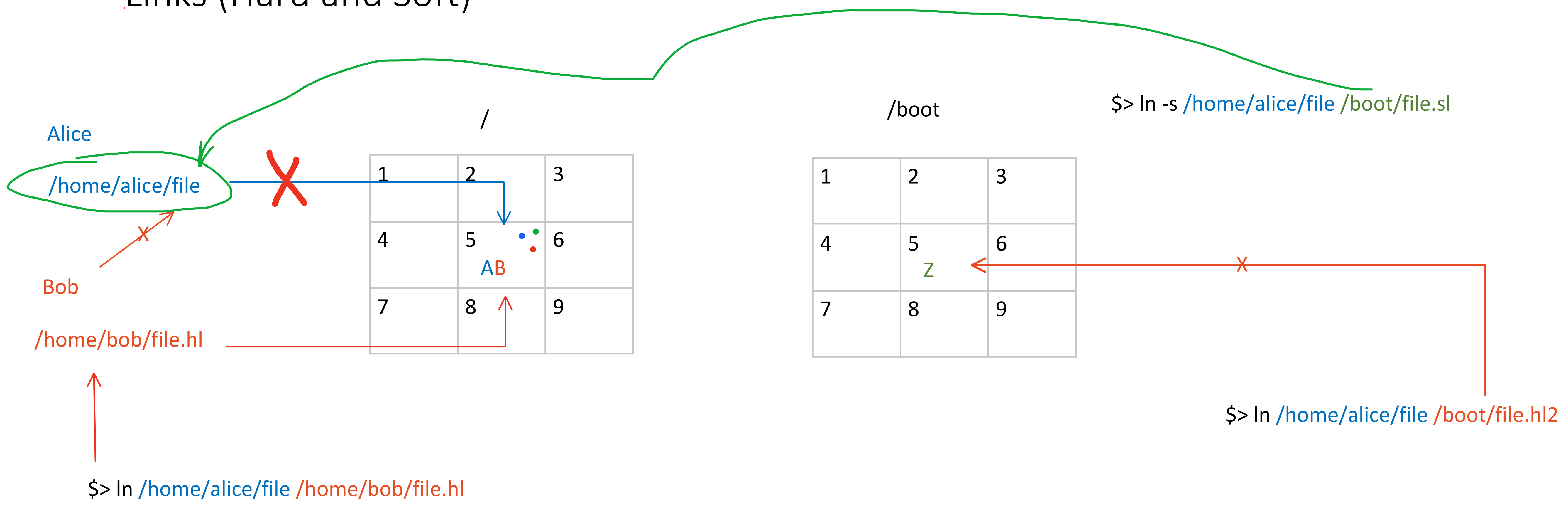
Special Permissions:

Permission	Files	Directories
SUID (u+s)	Executes the file with the UID of file's owner	---
SGID (g+s)	Executes the file with the GID of file's group	New files in the directory will be owned by directory's group
Sticky bit (o+t)	---	Only owner can delete his own files

Permissions
----
1. View the permissions of "Dir"'s files and directories. is there a difference between files and directories? why?
2. Add an execution permission to all users on all of "Dir"'s files and directories.
3. Remove write permission to everyone except the files' owners to all of "Dir"'s files and directories.
4. Set the permissions of "Folder"'s files and directories to the same permissions of "Dir"'s files using the symbolic syntax.
5. Set the permissions of "files"'s files and directories to the same permissions of "Dir"'s files using the numeric syntax.
6. Create 2 script files under the "scripts" directory. what are their permission? why?
7. Set the current session, so that every new directory will have full permission to its owner, and read and exec permission to all other users.
8. Create a directory named python under "scripts" folder and within create the file script.py. what are the permissions of the directory? what are the permissions of the file?
9. Grant write permission to everyone on "python" directory. and set the state so that each new file in this directory will be owned by the directory's group.
10. Using the user "root" create a file under "python". check its group and owner. now try to delete it. did you have the permission? why?
11. Using the user "root" create a file under "/tmp".
12. Check if your standard user has write permissions to /tmp
13. Try to delete "root"'s file created in "/tmp". did it let you? why?
14. Grant the permissions 4755 on "script.py". what are their meaning?

```
ls -l Dir
chmod a+x Dir/*
chmod go-w Dir/*
chmod u=rwx,g=rx,o=rx Folder/*
chmod 755 files/*
cd ~/scripts
touch script{1,2}.sh
umask 22
mkdir python; ls -l python
touch python/script.py; ls -l python/script.python
chmod a+w,g+s python
su -c "touch python/root.file" ; rm python/root.file
su -c "touch /tmp/nobody_can_delete"
ls -ld /tmp
rm /tmp/no_body_can_delete
chmod 4755 ~/scripts/python/script.py
```

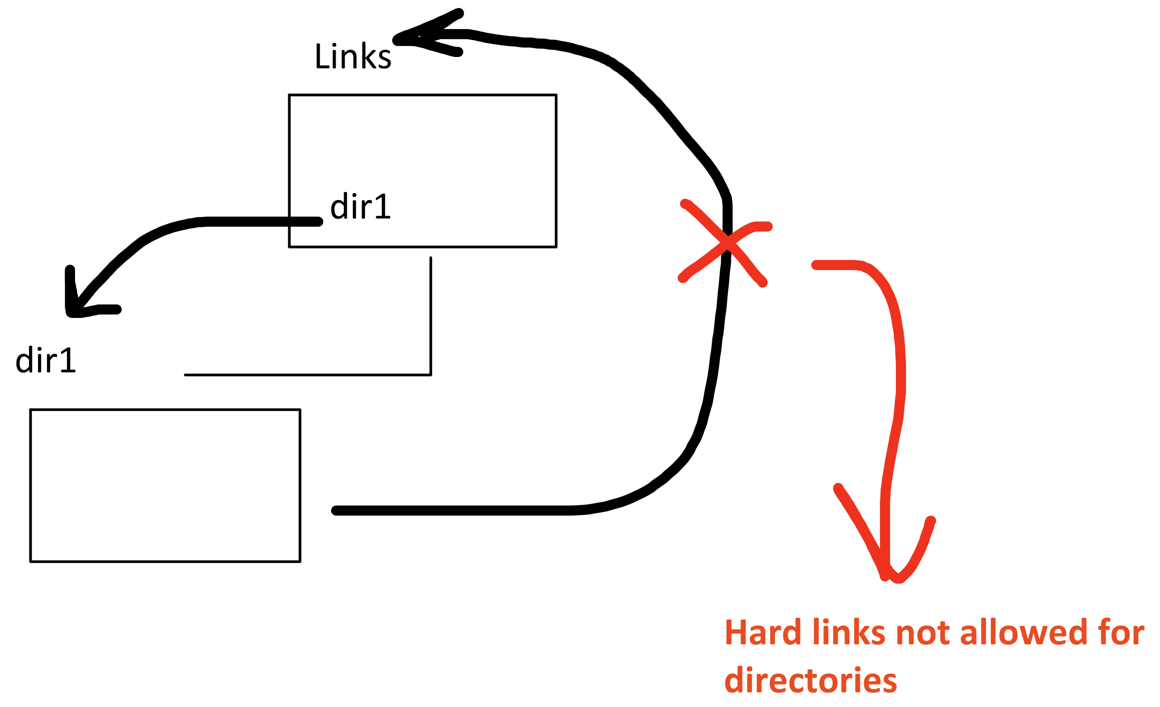
Links (Hard and Soft)



$> ln -s /home/alice/file /boot/file.sl

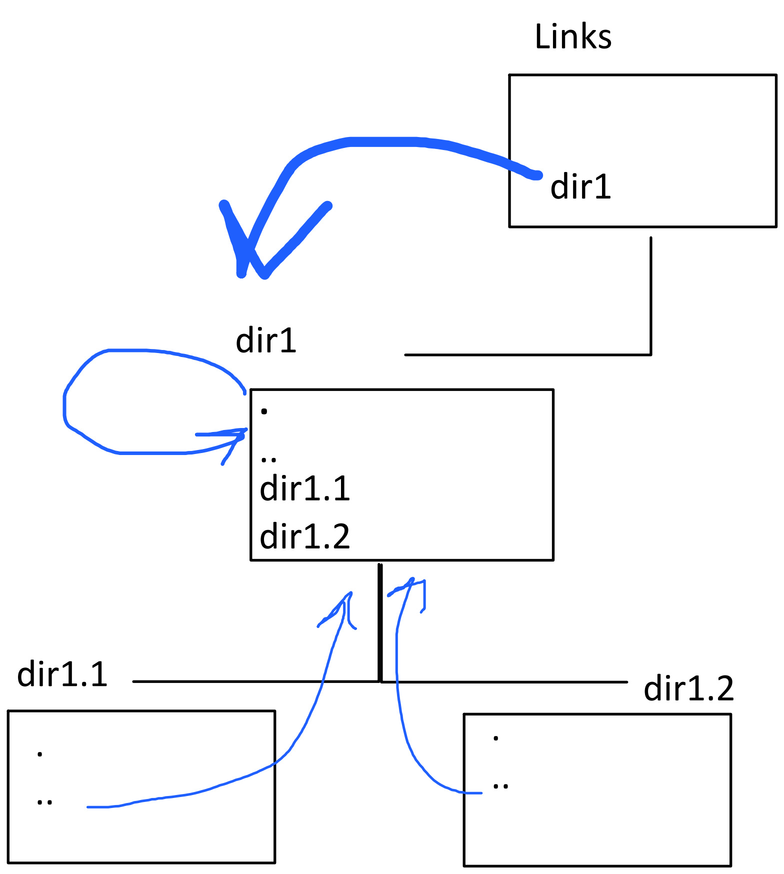
$> ln /home/alice/file /boot/file.hl2

$> ln /home/alice/file /home/bob/file.hl



Hard links not allowed for directories

Tell number of subdirectories according to link count:



Links

-----

1. Create the directory links under your home folder.
2. Create in "links" the file "original" with the text "This is the original file"
3. Create a Hard link to "original" named "pseudo-original" that will be place in "links"
4. Enter your home folder and from this working directory create a Soft link of the file "original" named "link2original" that will be placed in "links"
5. Get back to "links" folder. review all files. is the soft link ok? - if not. re-create it so it would.
6. Remove the file "original"
7. Display the content of remaining files. which of them worked?
8. Restore the file "original" (Hint: you could use pseudo-original to do it).
9. Display once more the file "link2original" - now it should work.
10. Change the permissions of "link2original" to full permission to the user and none to all other users.
11. Review the files' properties again. which of them changed?
12. Again, remove the file "original"
13. Create a new file named "original" with the text "This is not the original file"
14. Ensure that "pseudo-original" is not a Hard link of "original".
15. Set 640 permissions on "link2original". on what files did it affect?

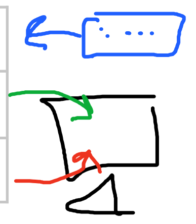
→

```
mkdir links; cd links
echo "This is the original file" > original
ln original pseudo-original; ls -li
cd ~; ln -s original links/link2original
cd links; cat *
rm original
cat *
ln pseudo-original original
cat link2original
chmod 700 link2original
ls -l
rm original
echo "This is not the original file" > original
ls -li
chmod 640 link2original; ls -l
```

Input / Output redirection

Bash

0	STDIN
1	STDOUT
2	STDERR



$> echo A > file

$> ls file > list

$> ls file *NE* 1> list 2>errors → CSH → $> (ls file *NE* > list) >& errors

(non-existing file)

/dev/null

$> ls file *NE* &>both.txt → CSH → $> ls file *NE* >&both.txt

Clobbering files: (how to prevent accidental file overwrite)

```
[noam@localhost ~]$ cat file
A
[noam@localhost ~]$
[noam@localhost ~]$ echo Hi > file
[noam@localhost ~]$ set -o noclobber
[noam@localhost ~]$ echo Bye > file
-bash: file: cannot overwrite existing file
[noam@localhost ~]$
[noam@localhost ~]$ echo Bye >> file
[noam@localhost ~]$ cat file
Hi
Bye
[noam@localhost ~]$ echo Cya >| file
[noam@localhost ~]$ cat file
Cya
[noam@localhost ~]$ set +o noclobber
[noam@localhost ~]$ echo "Hello again" > file
[noam@localhost ~]$ cat file
Hello again
```

Input redirection (how to create and use an answer file to an interactive process:

```
[noam@localhost ~]$ mkdir inputred
[noam@localhost ~]$ cd !$
cd inputred
[noam@localhost inputred]$
[noam@localhost inputred]$ cat > ans
y
y
y
n
n
n
y
y
y
[noam@localhost inputred]$
[noam@localhost inputred]$ touch {1..9}
[noam@localhost inputred]$ ll
total 4
-rw-r--r--. 1 noam noam  0 May 17 20:11 1
-rw-r--r--. 1 noam noam  0 May 17 20:11 2
-rw-r--r--. 1 noam noam  0 May 17 20:11 3
-rw-r--r--. 1 noam noam  0 May 17 20:11 4
-rw-r--r--. 1 noam noam  0 May 17 20:11 5
-rw-r--r--. 1 noam noam  0 May 17 20:11 6
-rw-r--r--. 1 noam noam  0 May 17 20:11 7
-rw-r--r--. 1 noam noam  0 May 17 20:11 8
-rw-r--r--. 1 noam noam  0 May 17 20:11 9
-rw-r--r--. 1 noam noam 18 May 17 20:11 ans
[noam@localhost inputred]$ rm -i *
rm: remove regular empty file '1'?
rm: remove regular empty file '2'?
rm: remove regular empty file '3'?
rm: remove regular empty file '4'?
rm: remove regular empty file '5'?
rm: remove regular empty file '6'?
rm: remove regular empty file '7'?
rm: remove regular empty file '8'?
rm: remove regular empty file '9'?
rm: remove regular file 'ans'?
[noam@localhost inputred]$ rm -i * 2>/dev/null
```

```
[noam@localhost inputred]$ rm -i * < ans 2>/dev/null
[noam@localhost inputred]$ ll
total 4
-rw-r--r--. 1 noam noam  0 May 17 20:11 4
-rw-r--r--. 1 noam noam  0 May 17 20:11 5
-rw-r--r--. 1 noam noam  0 May 17 20:11 6
-rw-r--r--. 1 noam noam 18 May 17 20:11 ans
```

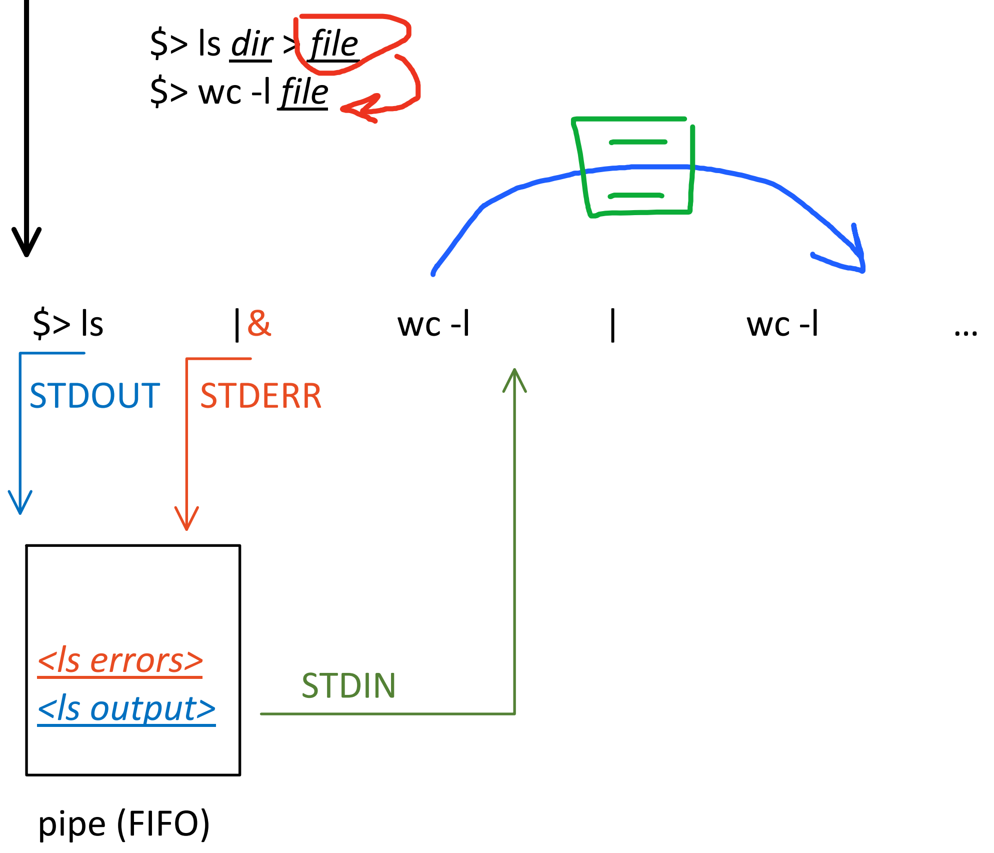
Input/Output redirection
----

0. Create a directory named "iored" and enter it.
1. Redirect the output of the command 'ls -l ~/Dir NOTHING' to the file "output.only"
2. Redirect the errors of the command 'ls -l ~/Dir NOTHING' to the file "errors.only"
3. Redirect the both the output and errors of the command 'ls -l ~/Dir NOTHING' to the file "err-n-out.both"
4. Execute the command 'echo oops > err-n-out.both'
5. Oh NO! - you've clobbered the "err-n-out.both" file
    a. Restore it using both files created in section 1 and 2
    b. Set the session so that clobbering files will not be an option.
6. Again execute the command in section 4
7. Append the text "Well done" to "err-n-out.both". is that consistent with the no clobbering policy?
8. Again, run command in section 4 with a correction that will overwrite "err-n-out.both".
9. Cancel the no clobbering affect, and re-run section 5-a
10. Create the files a.temp b.temp c.temp d.temp
11. Create a file named "file.answers" and remove with it (and with 'rm -i' command) the 1st, 3rd and 5th file in the directory.

→

```
mkdir iored; cd iored
ls -l ~/Dir Nothing > output.only
ls -l ~/Dir Nothing 2> errors.only
ls -l ~/Dir Nothing &> err-n-out.both
echo Oops > err-n-out.both
cat output.only errors.only > err-n-out.both
set -o noclobber
echo Oops > err-n-out.both
echo "Well done" >> err-n-out.both
echo Oops >| err-n-out.both
set +o noclobber
cat output.only errors.only > err-n-out.both
touch {a..d}.temp
echo -e "y\nn\ny\nn\ny" > file.answers ; rm -i * < file.answers
```

Pipes and filters



Line filters:

head [-*number*]
tail [-*number*]
grep [-iwvceRl…] *pattern*

Misc. filters:

tr *original char translated char*
tr -d *char to delete*

sort [-rgk *no. of ordering field* -t *delimiter between fields* …]

Column filters:

cut -c *1-3,8,…*   (cut -d *delimiter* -f *field numbers to print*)

awk [-F *field separator expression*] '{print "Text…" , $1}'

Default field separator is *white space*

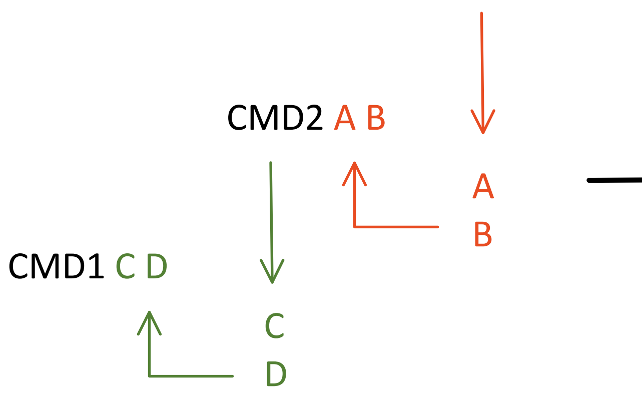
$1, $2, …, $(NF-1) ,$NF

Number of Fields (number of last field)

Command substitution --> $(…) / `…`

BASH   BASH
              CSH

$> CMD1 $(CMD2…$(CMD3…)…)



```
[noam@localhost ~]$ ll | grep -v noam | tail -n +2 | awk '{print $NF}'
root.file1
root.file2
[noam@localhost ~]$ whoami
noam
[noam@localhost ~]$
[noam@localhost ~]$
[noam@localhost ~]$ ll | grep -v $(whoami) | tail -n +2 | awk '{print $NF}'
root.file1
root.file2
[noam@localhost ~]$ rm $(ll | grep -v $(whoami) | tail -n +2 | awk '{print $NF}')
rm: remove write-protected regular empty file ‘root.file1’?
rm: remove write-protected regular empty file ‘root.file2’?
```

CMD3

CMD1   CMD2

Pipes and Filters
----
1. Print the User name and UID of the 3 first users in /etc/passwd
2. Print the User name and UID of the 2nd user in /etc/passwd
3. Print the line of your current user in /etc/passwd
4. Print the list of files under your home folder and their permissions
5. Print the number of directories in your home folder (including sub-directories)
6. Print the number of files (of file type 'file') that have write permission to 'other'
7. Print the number of files changed today. (Should be relevant to each day you execute the command.)
Hints:
    a. Run ls -l and look at the date field
    b. Run the date command, with the relevant format (as used in ls -l)
    c. Now filter lines from the output of ls -l command using the date command and the mechanism of 'Command substitution'
8. Print using df the usage of the / filesystem (Without the % sign)
9. Print the 3 smallest files in your home folder
10. Print the last 10 users created in the system (last 10 lines in /etc/passwd) sorted by their UID
11. Print the number of files in your home folder whose names end with a digit and have 'read' permission to all users (sub-directories don't have to be included)

```
Pipes and Filters
------------------
head -3 /etc/passwd | awk -F: '{print $1,$3}'
head -2 /etc/passwd |tail -1 | awk -F: '{print $1,$3}'
grep $(whoami) /etc/passwd
ls -l | awk '{print $1, $NF}'
ls -lR | cut -c1 | grep d -c
ls -l | cut -c1,9 | grep -ce "-w"
ls -l | awk '{print $6$7}' | grep -c $(date | awk '{print $2$3}')
df | grep -w / | awk '{print $(NF-1)}' | tr -d %
ls -l | tail -n +2 | sort -nk5 -r | tail -3
tail /etc/passwd | sort -nk3 -t:
ls -l *[0-9] | cut -c2,5,8 | grep rrr -c
```

Regular Expressions

Line filters:

eg/re/p ... "regex"

Regular Expressions (\ is required when using grep instead of egrep)

Characters	Multipliers	Position
[a-zA-Z0-9 @*#...]	* - 0-infinity	^ - Beginning of line
[^a]	\+ - 1-infinity	$ - End of line
.	\? - 0,1	
	\{2\} - 2 times	
	\{3,10\} - between 3 and 10 times	
	\{6,\} - 6-inifnity	

All chars (without \n)

^a....z$

MAC address:

[0-9A-Fa-f]

12:34:56:78:9A:BC
12-34-56-78-9A-BC
123456789ABC
1234.5678.9abc

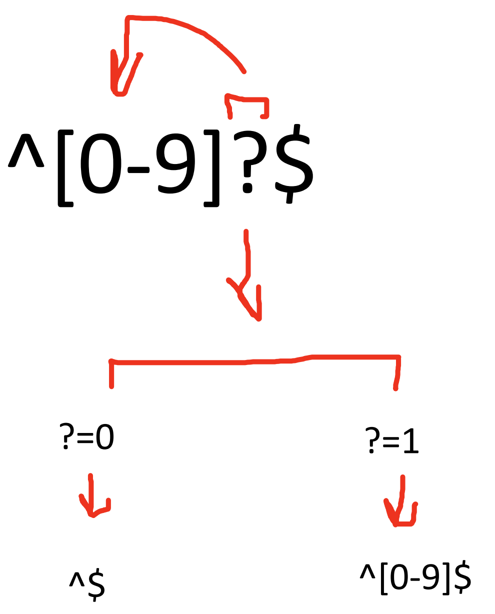
Abstract syntax

(([0-9A-Fa-f])*2[:-.])*6

Concrete RegEx syntax

\([0-9a-fA-F]\{2\}[-:.]\?\)\{6\}

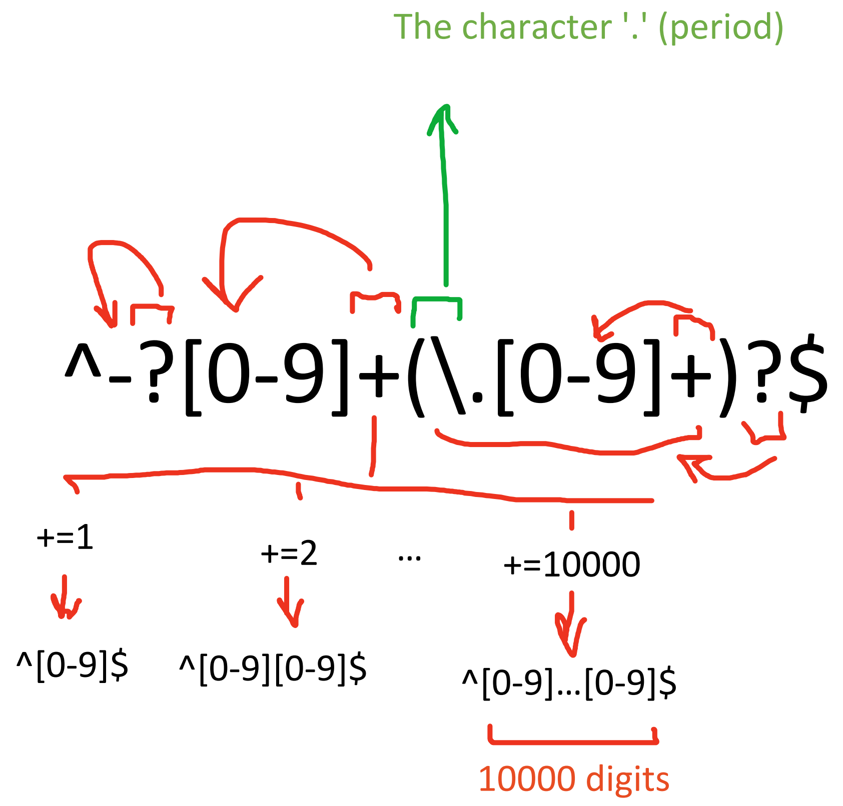
12:34-56.789a-BC:



^[0-9]?$

?=0 → ^$

?=1 → ^[0-9]$



The character '.' (period)

^-?[0-9]+(\.[0-9]+)?$

+=1 → ^[0-9]$

+=2 → ^[0-9][0-9]$

...

+=10000 → ^[0-9]...[0-9]$

10000 digits

Examples:

```
[noam@localhost ~]$ cat > file
Justice
Justice4All
Jstuice4All2
2
[noam@localhost ~]$ grep "^[a-zA-Z]*[0-9]$" file
2
[noam@localhost ~]$
[noam@localhost ~]$
[noam@localhost ~]$ cat > file
vi
viable
viableable
viabl
[noam@localhost ~]$ egrep "^vi(able)*$" file
vi
viable
viableable
[noam@localhost ~]$
[noam@localhost ~]$ egrep "^viable*$" file
viable
viabl
```

Regular Expressions
----

1. List all the directories under your home folder (including sub-directories)
2. List all files whose name ends with "c"
3. Out of the output of 'ls -lR ~' list lines that starts with "-" and contains the letter "f"
4. List all directories whose name ends with a digit (under all directories of your home folder)
5. List all files whose permission are full for everyone (777) without using the expression "rwx"
6. Print the number of empty lines on the output of 'ls -lR ~'
7. Print the number of non-empty lines on the output of 'ls -lR ~' (in 2 different ways)
8. Print lines that contains between 1 and 10 characters out of the output of 'ls -lR ~'
9. Print lines that contains exactly 7 characters that none of them is '.' (dot) out of the output of 'ls -lR ~'
10. List all files under your home folder whose name ends with a digit, and contains a line that starts with "t" and ends with a digit. (No need to search under sub-directories of your home folder)

find command (and locate command)

```
$> locate [options] <pattern>
```

Locate operates on an index - and thus really fast!
if index not updated --> #> updatedb &

cp
mv
rm
grep
find
gzip
cat
....

2 -> 1 + subdir's of level one content. without subdirs of level 2
1 -> 0 + dir's content. without content of subdirs.
0 -> specified dir's properties. without dir's content

```
...
-perm   -->  777, -1002
-mtime -->  0, -7, +60 (days) ~~ -mmin --> same in minutes
-size     -->  2M,-4k, +3G
-type     -->  f,d,l,s,p,b,c
-user     -->  owner
-name   -->  wildcard
```

Logic "OR"

-ok

$> find [dir1 dir2 ...] [-maxdepth *depth*] *search-parameter value-of-parameter* [[-o] *search-parameter value-of-parameter* ...] [-exec *shell-command* {} \;]

(A | B)& C

```
$> find /home /tmp -maxdepth 2  \( -user $(whoami) -o -user root \) -name "*.log" -exec rm -f {} \;
```

≈

```
rm -f  $(find /home /tmp -maxdepth 2  \( -user $(whoami) -o -user root \) -name "*.log")
```

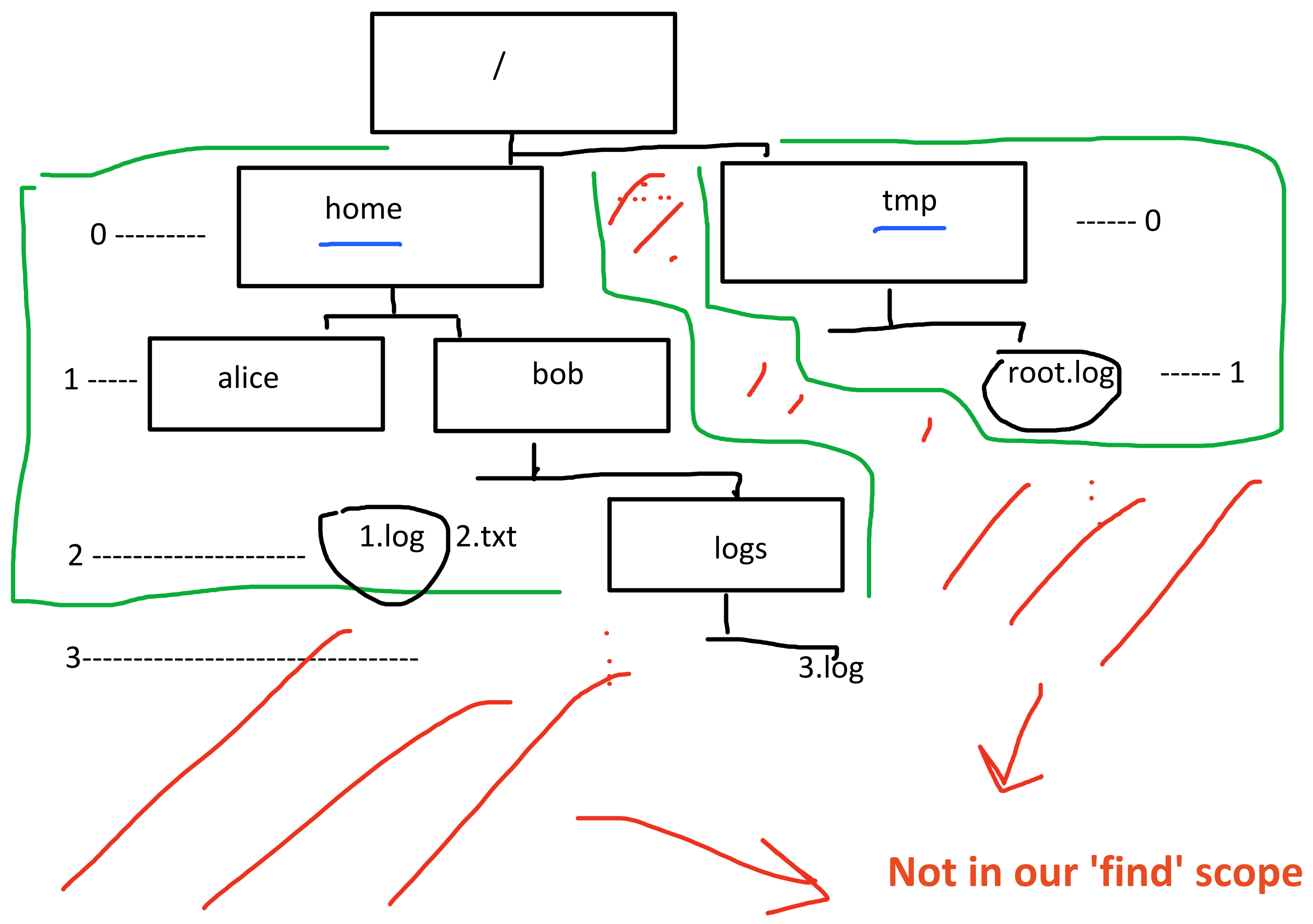
```
rm -f  /home/bob/1.log /tmp/root.log
```

For N results 1 process of *rm -f* will be executed

```
/home/bob/1.log
/tmp/root.log
```

```
rm -f /home/bob/1.log ; rm -f /tmp/root.log ;
```

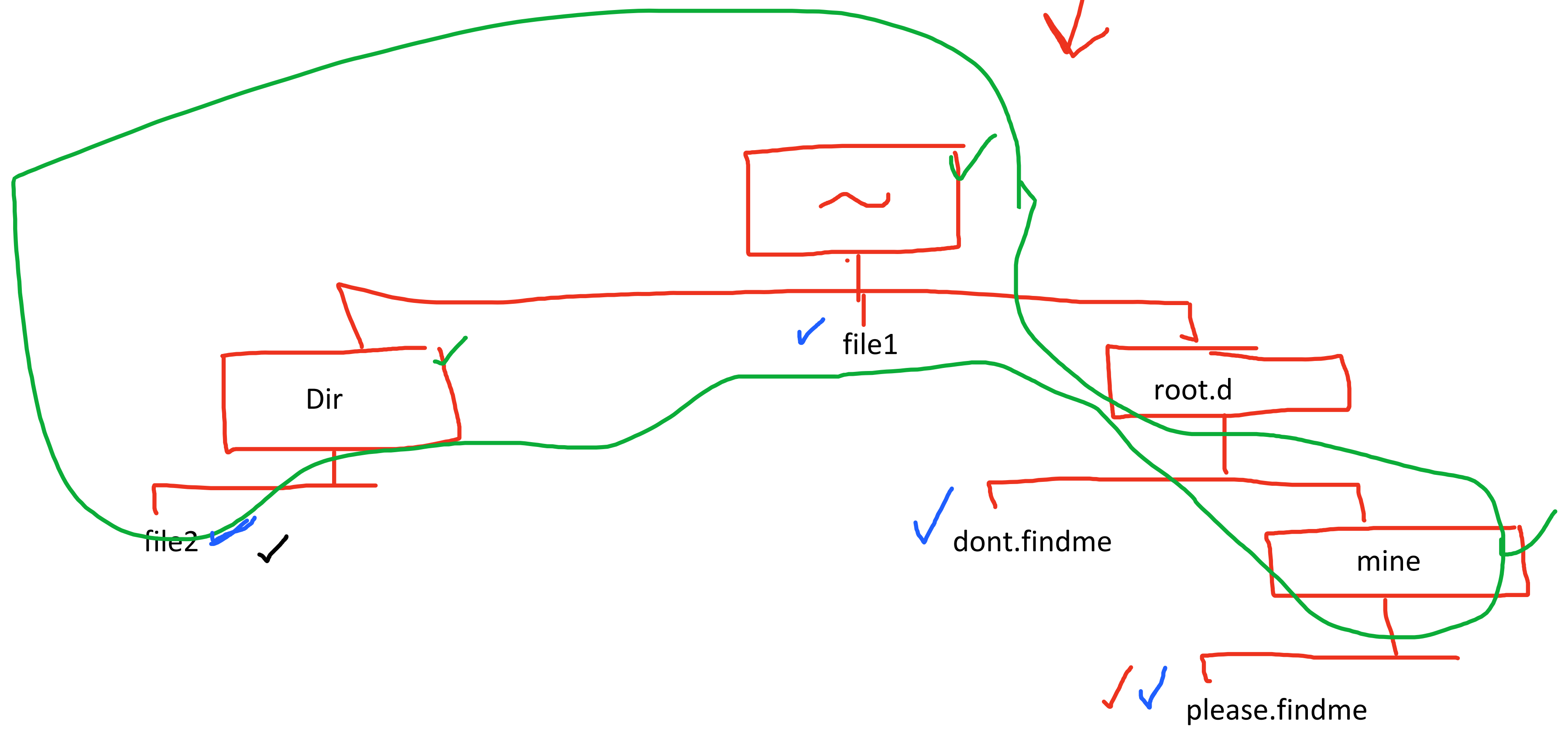
For N results N processes of *rm -f* will be executed



Not in our 'find' scope

Find
-----
0. NOTE - Redirect the errors so they won't be displayed
1. What is the number of directories owned by the current user under /tmp
2. What is the number of files (file type 'file') under your home folder
3. How many files have changed in the last 8 hours under your home folder
4. Display the name and size (in kB MB GB etc.) of files larger than 1M under your home folder
5. Find all files whose name ends with '.c'
6. Remove all files (not directories) whose names end with '.temp' under your home folder
7. Execute the command -> dd if=/dev/zero of=~/large.log bs=1M count=32
8. Using gzip compress all files larger than 20MB whose file extension is 'log'
9. Find under folders owned by you (under your home folder), files that contain the word 'total' in its content
    A. The owner of the file isn't relevant
    B. Create using root a folder under your home folder and give everyone full permissions
    C. With your user create a file under root's folder containing the word 'total'
    D. Make sure that this file doesn't come up in your search.
10. Find under folders owned by you (under your home folder), files larger than 2MB and display its size.
    A. The owner of the file isn't relevant
    B. Create using root a folder under your home folder and give everyone full permissions
    C. With your user create a file under root's folder of size 3MB (using the command in section 7)
    D. Make sure that this file doesn't come up in your search.

```
find /tmp -user $(whoami) -type d 2>/dev/null| wc -l
find ~ -type f | wc -l
find ~ -mmin -480 | wc -l
find ~ -size +1M -exec du -h {} \;
find -name "*.c"
find -name "*.temp" -exec rm -f {} \;
dd if=/dev/zero of=~/large.log bs=1M count=32
find ~ -size +20M -name "*.log" -exec gzip {} \;
grep -l total $(find -type d -user $(whoami) -exec find {} -maxdepth 1 -type f \;)  2>/dev/null
du -h $(find -type d -user $(whoami) -exec find {} -maxdepth 1 -size +2M \;)  2>/dev/null
```

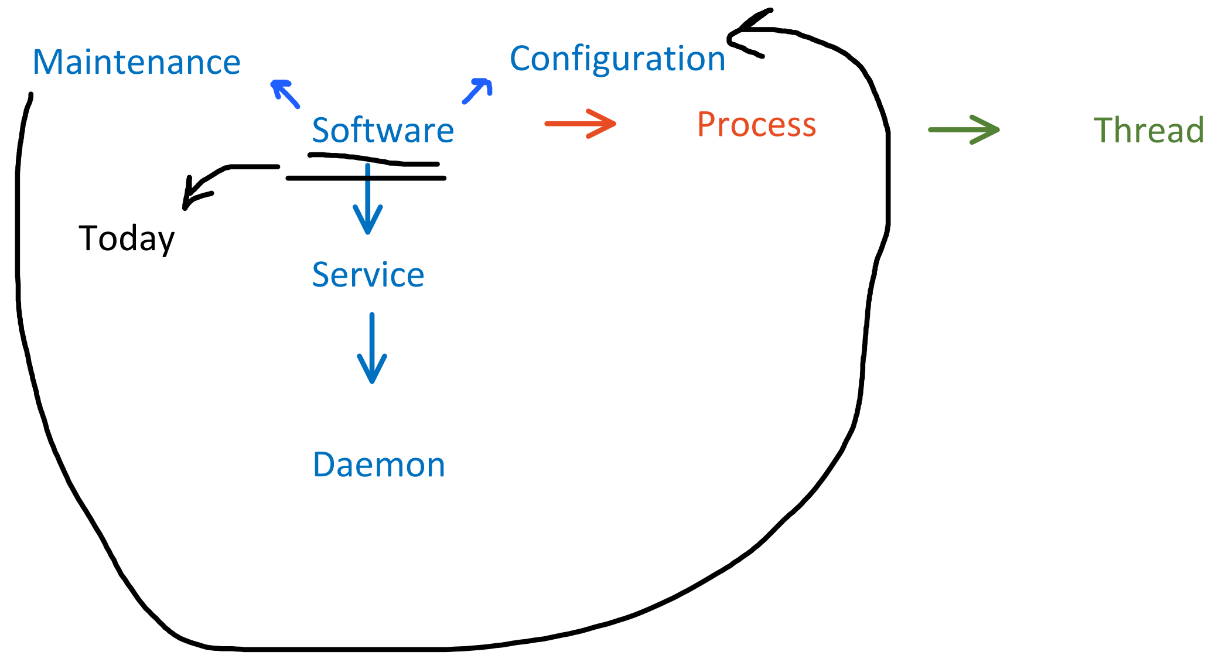


```
find -type d -user $(whoami) -exec grep -R "snoopy" {} \;
```

```
~
~/Dir
~/root.d/mine
```

```
grep -R "snoopy" ~ ; grep -R "snoopy" ~/Dir ; grep -R ~/root.d/mine ;
```

Software management



Software management - 3 technics

Compilation

app.tar.gz

0. Extract --> Extract installation files from archive to directory

   $> tar -zxvf app.tar.gz

1. Setup --> a. Compatibility check of our system to the package
   b. Custom install

   $> ./configure [OPTOINS] → Makefile: Build, Install, Cleanup

2. Build --> Compile the source files into binary files

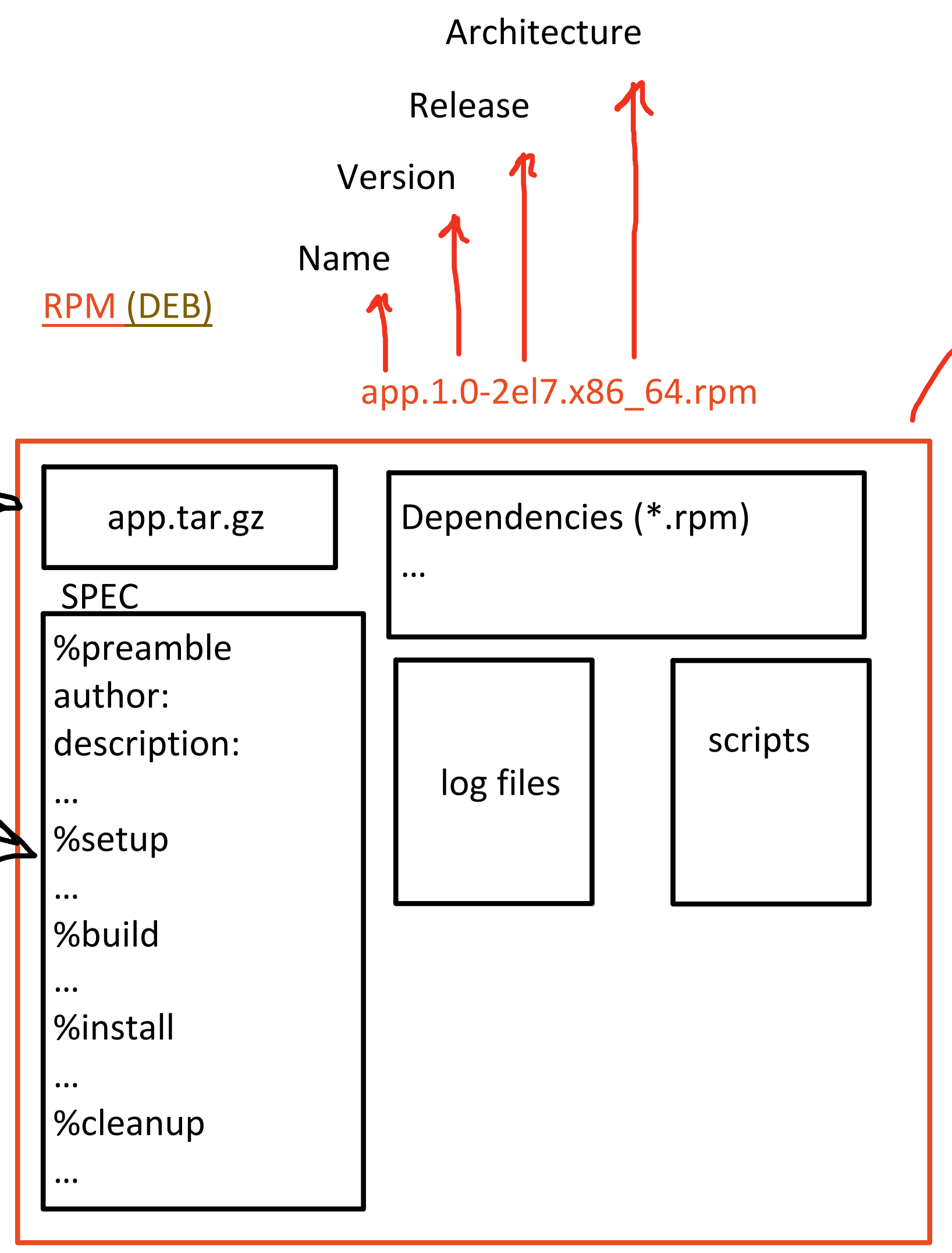
   $> make → Application **files** ready for execution

3. Install --> a. Distribution of application file to various dirs on our system (exec, conf, service, manuals etc.)
   b. Complimentary actions

   $> make install → Installed application

4. Cleanup --> Remove temp. installation files

   $> make clean



Install and Upgrade RPM:

$> rpm [-iUvh...] *app.rpm*

Removing RPM (Erasing):

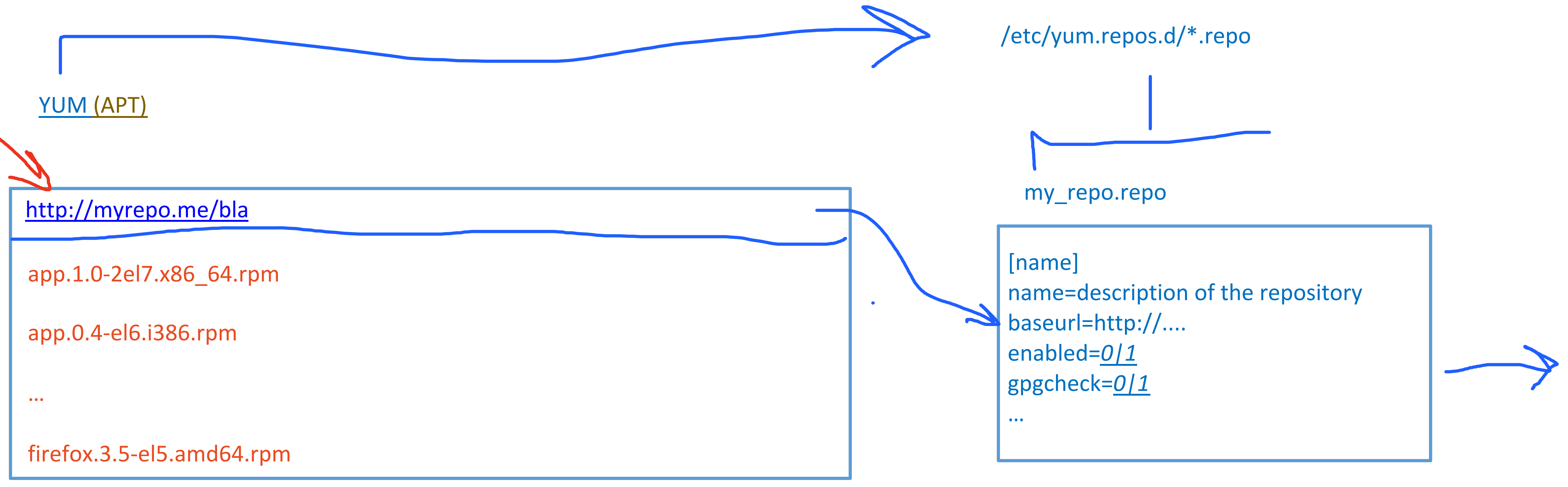
$> rpm -e [--nodeps] *app-name*

Querying the RPM Database:

$> rpm -q
- a -> all installed pckgs
- i -> information (shows the preamble)
- l -> list of files
- c -> list of conf. files
- f -> shows the original rpm of *file*
- p -> for pre-installed packages

Example: showing the conf. files of the packge that holds /etc/ssh/sshd_config)

```
rpm -qc $(rpm -qf /etc/ssh/sshd_config)
```



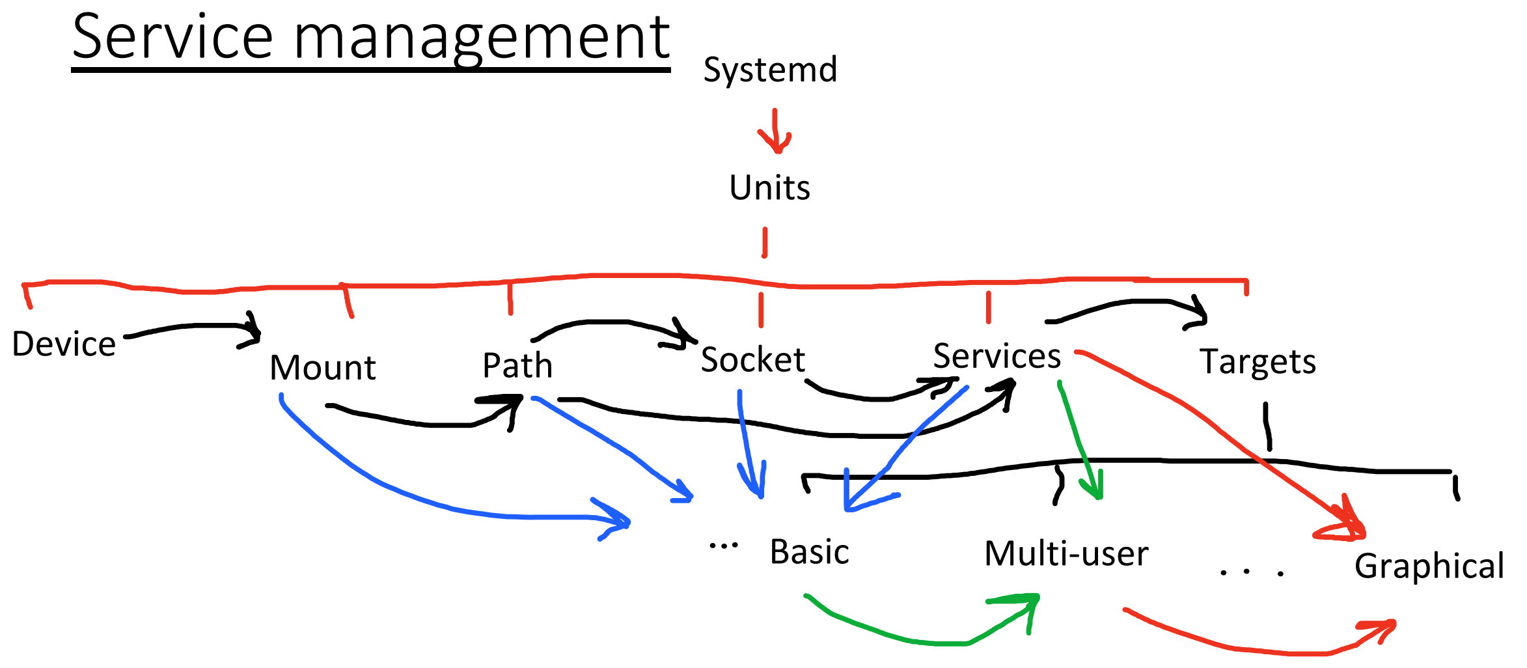
```
[name]
name=description of the repository
baseurl=http://....
enabled=0|1
gpgcheck=0|1
...
```

1. Manually
2. Download RPM of repo
3. #> yum-config-manager --add-repo *base-url*

$> yum	list
	search[-all] *keyword*
	info *pkg-name*
	install *pkg-name*
	update [*pkg-name*]
	upgrade [*pkg-name*]
	downgrade *pkg-name*
	remove *pkg-name*
	provides *pkg-name*
	localinstall *file.rpm*

Service and Process management

Service management



Systemd

Units

Device

Mount

Path

Socket

Services

Targets

...

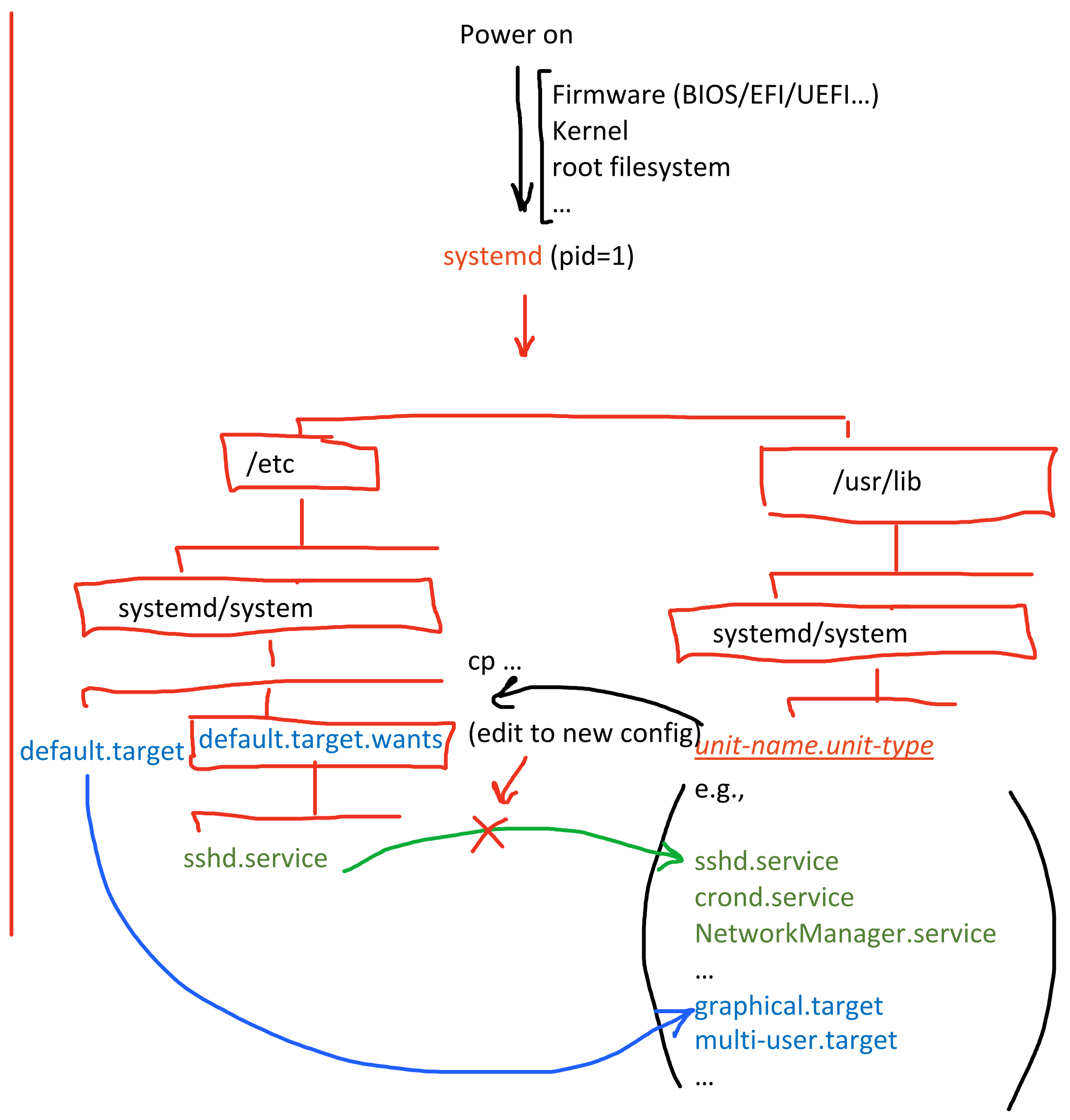
Basic

Multi-user

. . .

Graphical

Boot process - systemd driven (RH>7, Ubuntu>~15.6, SUSE>12…)



Power on

Firmware (BIOS/EFI/UEFI…)
Kernel
root filesystem
…

systemd (pid=1)

/etc

systemd/system

default.target

default.target.wants

sshd.service

cp …

(edit to new config)

/usr/lib

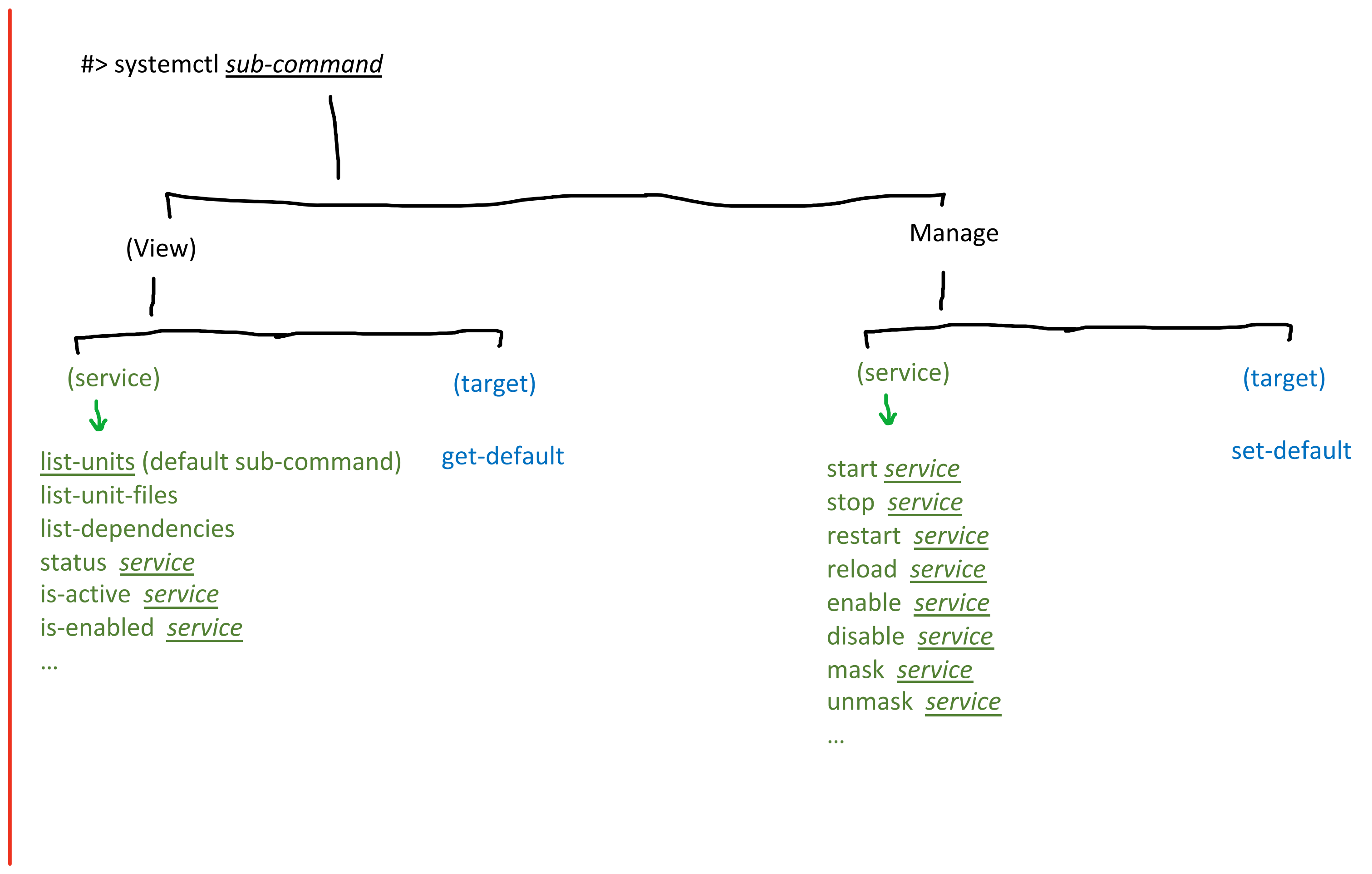
systemd/system

unit-name.unit-type

e.g.,

sshd.service
crond.service
NetworkManager.service
…
graphical.target
multi-user.target
…

Managing systemd with *systemctl*



#> systemctl *sub-command*

(View)

(service)

list-units (default sub-command)
list-unit-files
list-dependencies
status *service*
is-active *service*
is-enabled *service*
…

(target)

get-default

Manage

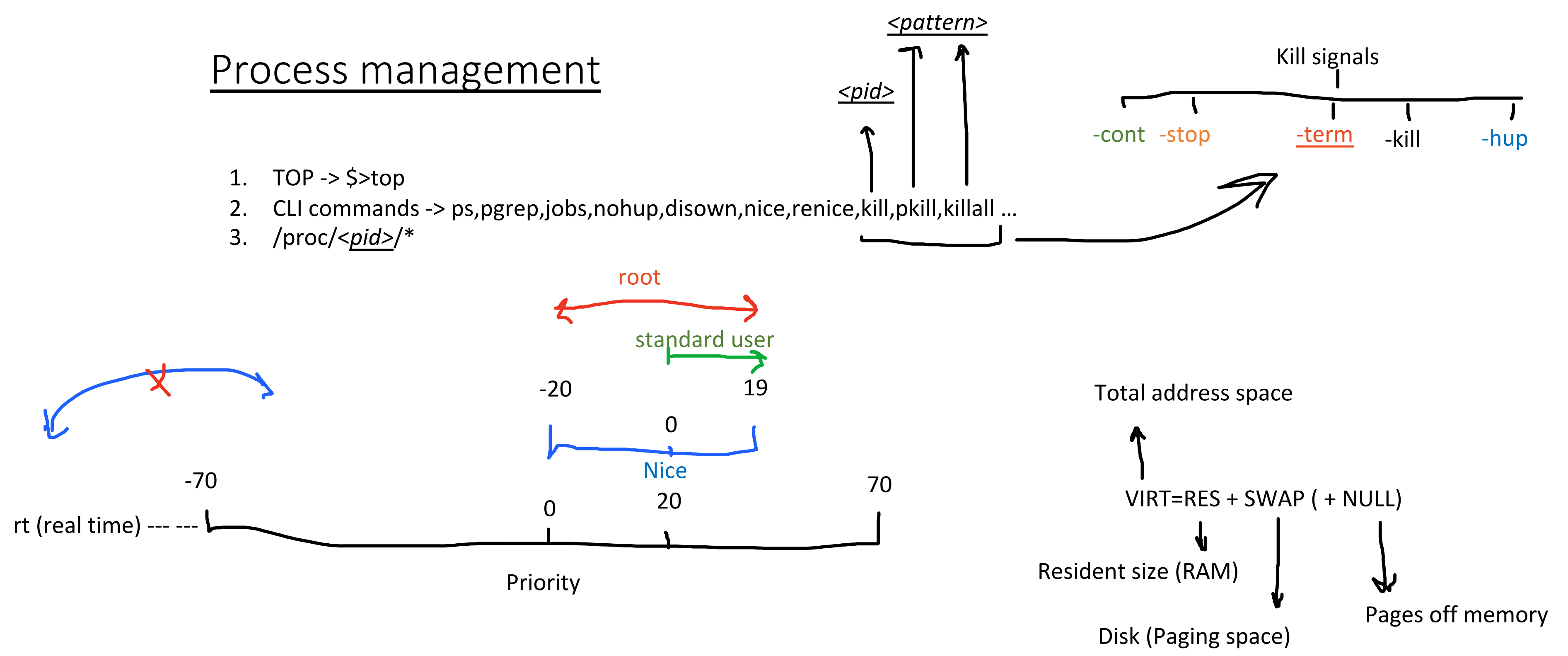
(service)

start *service*
stop *service*
restart *service*
reload *service*
enable *service*
disable *service*
mask *service*
unmask *service*
…

(target)

set-default

Process management



<pattern>

<pid>

1. TOP -> $>top
2. CLI commands -> ps,pgrep,jobs,nohup,disown,nice,renice,kill,pkill,killall …
3. /proc/<*pid*>/*

Kill signals

-cont -stop -term -kill -hup

root

standard user

-20

19

0

Nice

-70

70

0

20

rt (real time) --- ---

Priority

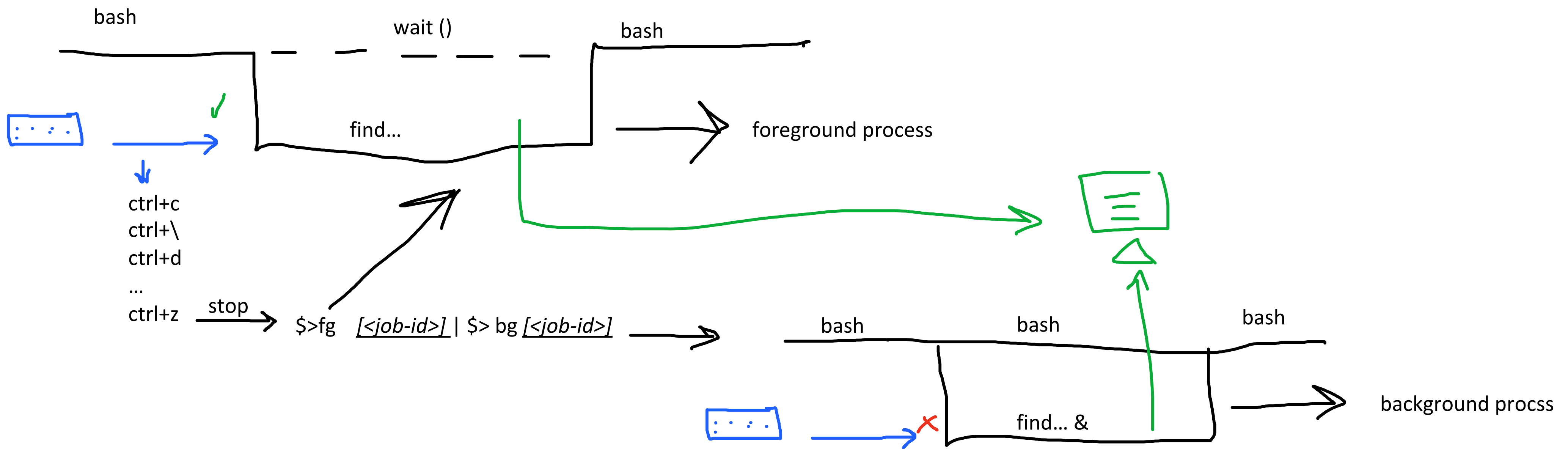
Total address space

VIRT=RES + SWAP (+ NULL)

Resident size (RAM)

Disk (Paging space)

Pages off memory



bash

wait ()

bash

find…

foreground process

ctrl+c
ctrl+\
ctrl+d
…
ctrl+z

stop

$>fg *[<job-id>]* | $> bg *[<job-id>]*

bash

bash

bash

find… &

background procss

Services and processes
----
1. Switch user to root
2. Execute the command: sed -i /^id/s/5/6/ /etc/inittab
3. Restart the server - notice that you're in a loop of reboots
4. Fix it.

Boot management

---
5. Check on what runlevels 'crond' is running by default.
6. Configure 'crond' to start on runlevels 3 and 5 alone.
7. Make sure in the relevant directory that 'crond' is off on runlevel 2.
8. Switch to runlevel 2, make sure that 'crond' is off.
9. Go back to runlevel 5, make sure that 'crond' is on. if so, stop it.
10. Execute the command: 'ls -lR /' and pause it immediately. look at its job number.
11. Now, transfer the job to the background.
12. Execute 'sleep 1001' and 'sleep 1002' in the background.
13. Create the 'sleep 1003' process with a nice number '3'.
14. Change the nice value of 'sleep 1001' and 'sleep 1002' to 1 and 2 respectively.
15. View the priority of all processes above.
16. Kill all sleep processes on one command.

Service/Process management

Advanced bash mechanisms

Up to now:

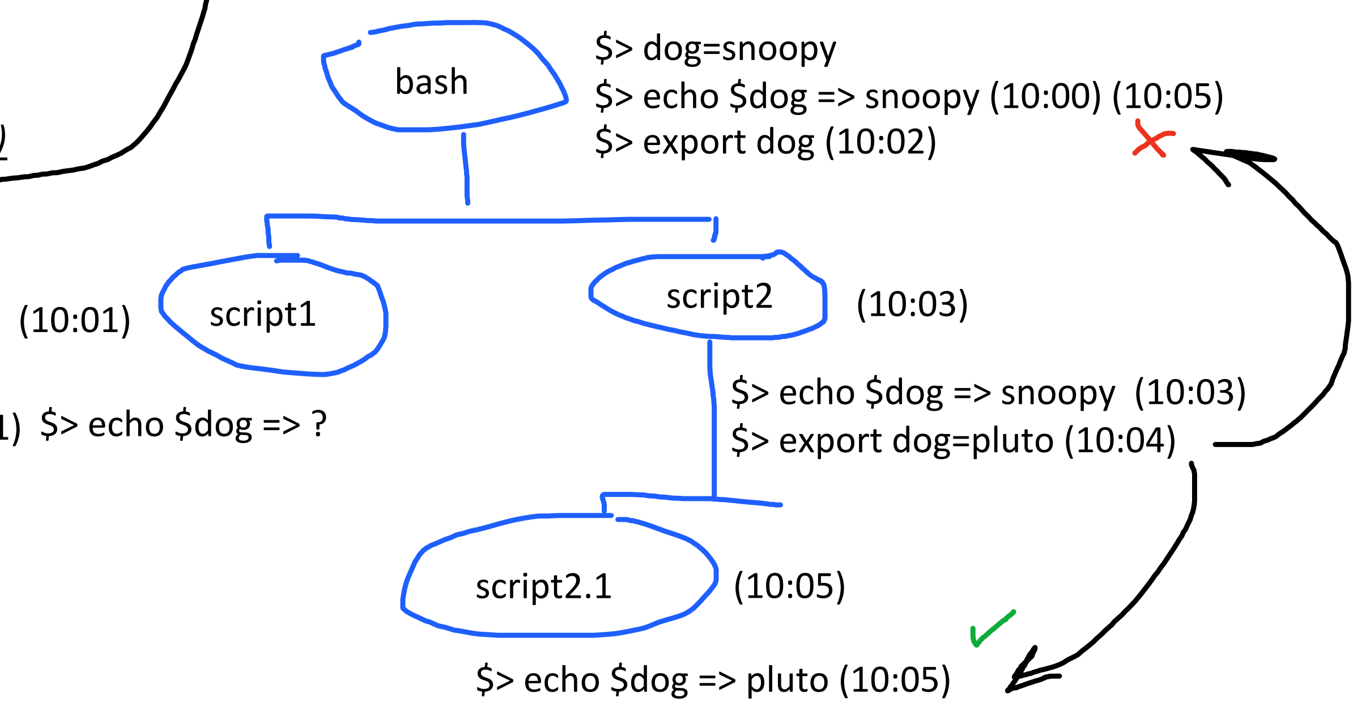
- ☑ Wildcards -> * ? [a-z] ...
- ☑ Pipes -> *CMD1* | *CMD2* | ...
- ☑ Command substitution -> *CMD1 $(CMD2...)*

Today:

- ☑ Variables
- ☑ History
- ☑ Quoting
- ☑ Alias
- ☑ Environment files

$> set => outputs all defined bash vars

bash vars.



$> dog=snoopy
$> echo $dog => snoopy (10:00) (10:05)
$> export dog (10:02)

$> echo $dog => snoopy (10:03)
$> export dog=pluto (10:04)

(10:03) (10:01) $> echo $dog => ?

$> echo $dog => pluto (10:05)

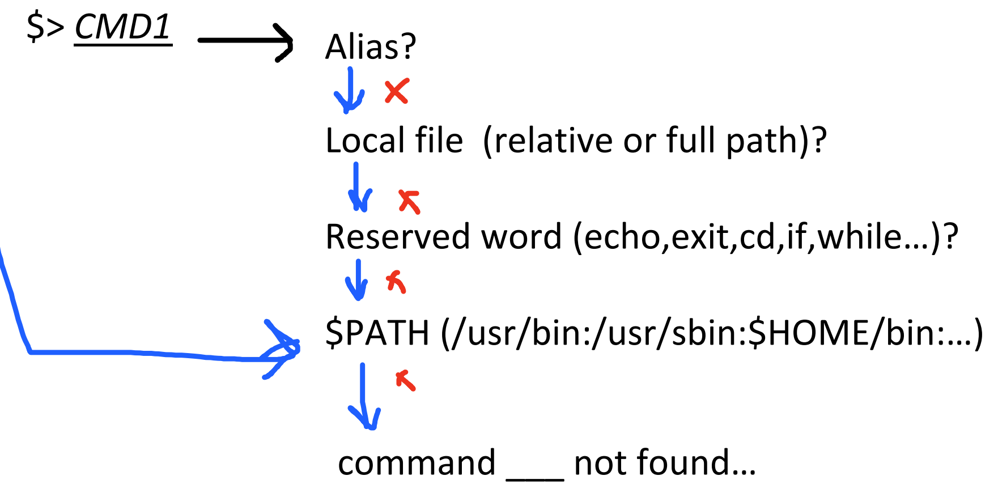
$> env

Env. vars. (exported vars.)

PATH

0 -> Current shell
$ -> Current PID
? -> exit status of last command
...

command inter. by bash



$> *CMD1*

Alias?

Local file (relative or full path)?

Reserved word (echo,exit,cd,if,while...)?

$PATH (/usr/bin:/usr/sbin:$HOME/bin:...)

command ___ not found...

Environment files

Purpose	Global	Personal
Login	/etc/profile (/etc/profile.d/*)	~/.bash_profile (ubuntu: ~/.profile)
Shell	/etc/bashrc	~/.bashrc
Logout	-	~/.bash_logout

Login process env files order:

1 /etc/profile
2,6 ~/.bash_profile
3,5 ~/.bashrc
4 /etc/bashrc

```
> History
> -------
> $HISTFILE => ~/.bash_history
> $HISTSIZE => 1000
>
> History mechanisms - Interactive
> -------------------------------
> $> history -> outputs command history (numbered)
> ^C
[noam@localhost ~]$ history | grep find
  305  find / -name "*.c"
  306  find / -name "*.c" 2>/dev/null
  576  echo snoopy > root.d/mine/please.findme
  577  echo snoopy > root.d/dont.findme
  578  find ~ -type d -user $(whoami) -exec grep "snoopy" {} \;
  579  find ~ -type d -user $(whoami) -exec grep -R "snoopy" {} \;
  580  grep "snoopy" $(find ~ -type d -user $(whoami) -exec find {} -maxdepth 1 -type f)
  581  grep "snoopy" $(find ~ -type d -user $(whoami) -exec find {} -maxdepth 1 -type f \;)
  582  grep -l "snoopy" $(find ~ -type d -user $(whoami) -exec find {} -maxdepth 1 -type f \;)
  587  history | grep find
[noam@localhost ~]$
[noam@localhost ~]$ '
>
> (ctrl+r) -> reverse search on command history
> $> (ctrl+r) expression
> ^C
(reverse-i-search)`grep': ^Cep "snoopy" $(find ~ -type d -user $(whoami) -exec find {} -
maxdepth 1 -type f \;)
[noam@localhost ~]$ '
>
> History mechanisms - Automatic
> -----------------------------
> !! -> exec. last command (e.g. head -1 /etc/shadow ; sudo !! => sudo head -1 /etc/shadow)
> !350 -> exec. cmd. no. 350 (as appeared on $> history)
> !-3 -> exec. the cmd we execed. 3 cmds. ago
> !str -> exec. last cmd. started with 'str'
>
> !$ -> Holds last argument of last cmd. (e.g. touch file1 file2 file3; ls -l !$ => ls -l file3)
> !* -> Holds all arguments of last cmd.
```

```
> Quoting - ignore special chars.
> -------
> ^C
[noam@localhost ~]$ echo You      owe      me        $5
You owe me
[noam@localhost ~]$
[noam@localhost ~]$ echo "You      owe      me        $5"
You      owe      me
[noam@localhost ~]$
[noam@localhost ~]$ " " -> Weak quoting: $ ! \ "    ^C
[noam@localhost ~]$
[noam@localhost ~]$ echo "You      owe      me        \$5"
You      owe      me        $5
[noam@localhost ~]$
[noam@localhost ~]$ \ -> Escape: \n \b \r \v ... (for full list see man echo -> -e)     ^C
[noam@localhost ~]$
[noam@localhost ~]$ echo 'You      owe      me        $5'
You      owe      me        $5
[noam@localhost ~]$
[noam@localhost ~]$ ' ' -> Strong quoting: '   ^C
[noam@localhost ~]$
[noam@localhost ~]$ echo 'I'm Happy'
> '
Im Happy

[noam@localhost ~]$
```

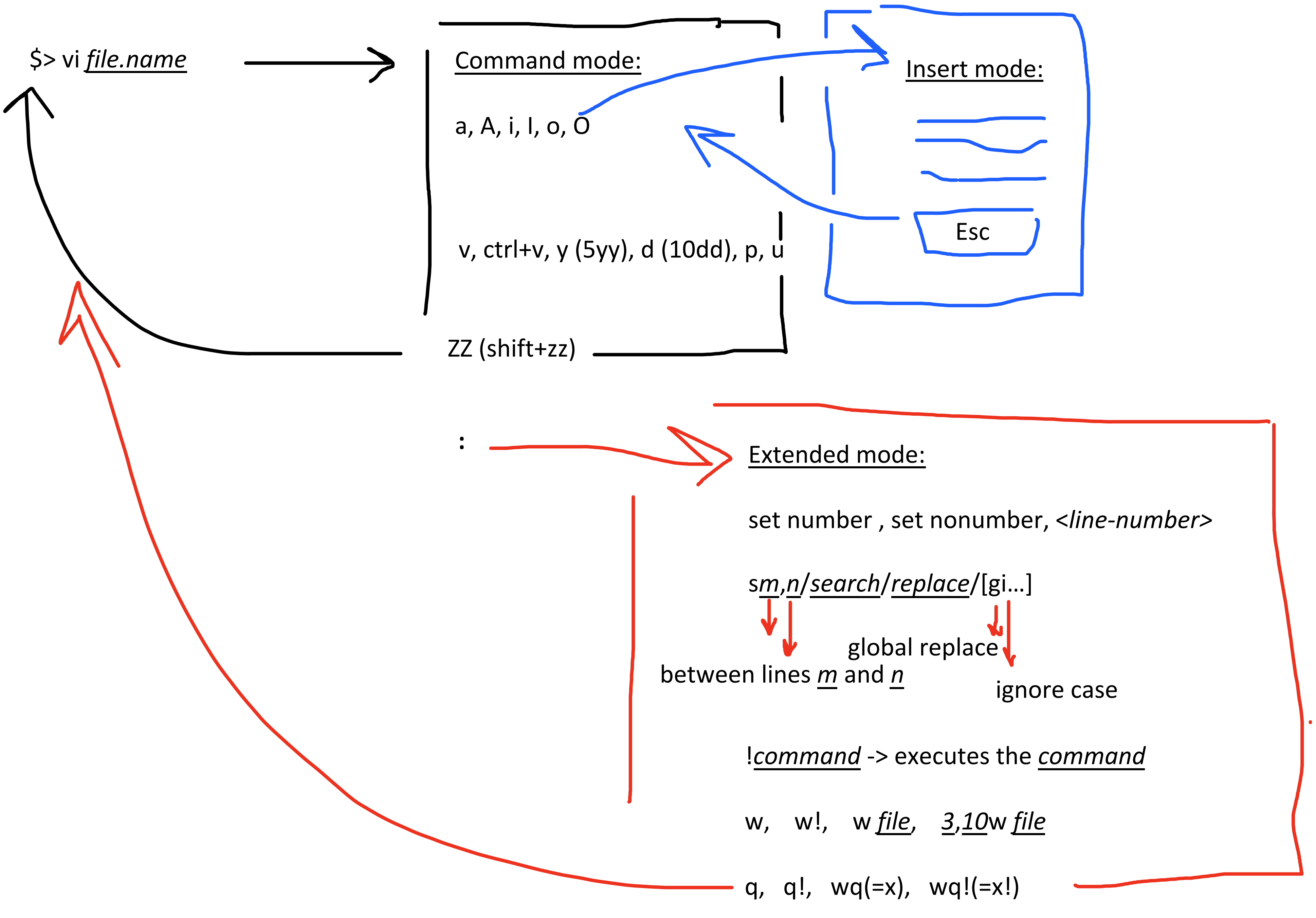
```
[noam@localhost ~]$ alias
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l.='ls -d .* --color=auto'
alias ll='ls -l --color=auto'
alias ls='ls --color=auto'
alias vi='vim'
alias which='alias | /usr/bin/which --tty-only --read-alias --show-dot --show-tilde'
[noam@localhost ~]$
[noam@localhost ~]$ alias hi="echo Hi"
[noam@localhost ~]$ hi
Hi
[noam@localhost ~]$ ls
Dir  file  file2  files  inputred  links  mylink  perms  root.file1  scripts
err  file1  file3  Folder  iored    list  numbers  root.d  root.file2
[noam@localhost ~]$ \ls
Dir  file  file2  files  inputred  links  mylink  perms  root.file1  scripts
err  file1  file3  Folder  iored    list  numbers  root.d  root.file2
[noam@localhost ~]$
[noam@localhost ~]$ unalias hi
[noam@localhost ~]$ hi
bash: hi: command not found...
```

Bash mechanisms
----
1. Create a directory called 'Clint Eastwood' with a file called "Million$Baby" in it.
2. Change the prompt string so it would display the current username and current working directory.
3. Create the following aliases
    A. 'run' -> adds execute permission to a file
    B. 'files' -> Displays all files (file type 'file') under the current directory
    C. 'links' -> Displays all soft links under the current directory
    D. 'large' -> Displays all files larger than 2MB under current directory
    E. 'del' -> moves all "deleted" files into a hidden directory called '.trash' under your home folder
    (Hint: look at mv --help)
    F. 'disk' -> Displays the %USE of the / filesystem
4. Make sure the aliases will be consistent over different bash processes.
5. Open a new terminal and execute one of the aliases you made, just to make sure.
6. Display the path of your history file
7. Read the last 10 lines of the history file without using its name or its variable.
8. Change the number of saved commands in the history file from 1000 to 2000
9. Make that change consistent over different logins

```
mkdir "Clint Eastwood" ; touch "Clint Eastwood"\'Million$Baby'
PS1='[$(whoami)@$(pwd)]$ '
alias run='chmod a+x'
alias files='ls -lR | grep ^-'
alias links='ls -lR | grep ^l'
alias large='find -size +2M -exec du -h {} \;'
mkdir ~/.recycle && alias del='mv -t ~/.recycle'
alias disk='df | grep /$ | awk "{print \$(NF-1)}" | tr -d %'
alias >> ~/.bashrc
echo $HISTFILE
cat !$
HISTSIZE=2000
echo  "export HISTSIZE=2000" >> ~/.bash_profile
```

vi (cheat sheet)

What is **vi** -> **vi**sual editor (as oopose to *sed* which is **stream editor**) | vim is VI Improved --> $> vimtutor



$> vi *file.name*

Command mode:

a, A, i, I, o, O

v, ctrl+v, y (5yy), d (10dd), p, u

ZZ (shift+zz)

Insert mode:

Esc

:

Extended mode:

set number , set nonumber, <*line-number*>

s*m*,*n*/*search*/*replace*/[gi…]

between lines *m* and *n*

global replace

ignore case

!*command* -> executes the *command*

w, w!, w *file*, *3*,*10*w *file*

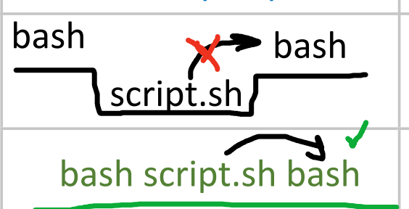
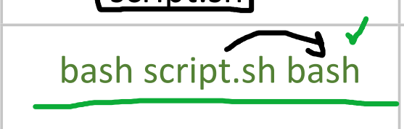
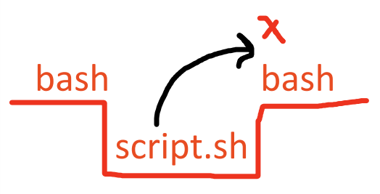
q, q!, wq(=x), wq!(=x!)

Bash scripts

script1.sh

```
#!/bin/bash
#Remarks…

echo $2 $1

echo $#

echo $@

echo "$*"
```

Executing bash script:

Execution command	Level of script's process	Permissions required for execution
Full path: /path/to/script.sh Relative path (current dir): (./script.sh)	 bash bash script.sh	rx (Read and Execute)
. /path/to/script.sh (. script.sh)	 bash script.sh bash	r (Read)
bash /path/to/script.sh bash script.sh -v --> Verbose -x --> Debug (not step-by-step)	 bash bash script.sh	r (Read)

Sending data into scripts:

1) Command line arguments/parameters

```
        $0      $1  $2  $3
$> ./script1.sh  A   B   C
```

```
B A
3
A B C
"A B C"
```

2) Reading STDIN into variables

$> read [-p "Please enter …"] *var1 var2* …

Conditions:

```
if condition
then
        commands to execute
elif condition2
then
        commands to execute
…
else
        commands to execute
fi
```

1) Test command: ($> man test)

syntax1: $> test *predicate value* (e.g. $> test -d dir.example --> Tests if dir.example is a directory type file)

syntax2: $> [*predicate value*] (e.g. $> [-d dir.example] --> same as before)

[*! predicate value*] Negative test

Binary test (on 2 values) [*value1 comparison predicate value2*]

Tests on files:

exists [-e *file*]
-f, -d, -L… , -r, -w, -x…

newer than… (by mod. time) [*file1* -nt *file2*]
-ot, -ef

equivalent file --> same inumber (hard links)

Tests of strings:

[-z *string*] ! -z = -n (not of length zero => not empty string)

[*string1* = *string2*]
> , >=, < , <=, != …

Tests on numbers:

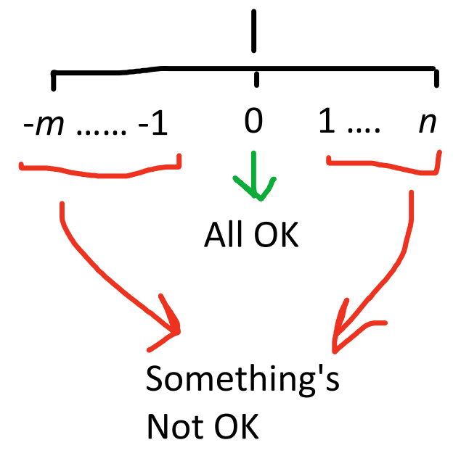
[*number1* -eq *number2*]
-gt, -ge, -lt, -le, -ne

[7 < 10] ✗

[7 -lt 10] ✓

2) Exit status:

$> *command* --> $?



Uses:

A) $> *command* ; stat=$?

```
if [ $stat -eq 0 ]
then
        blablabla…
…
elif [ $stat -eq 5 ]
then
        blabla…
…
fi
```

B) if *command* (Test command is one example)

```
then
        bla..
else
        blabla…
fi
```

if [$x -gt 0] && [$x -lt 10]

C) Conditional execution:

$> *CMD1* && *CMD2* --> CMD2 will execute only if CMD1 succeeded ($?=0)
$> *CMD1* || *CMD2* --> CMD2 will execute only if CMD1 failed($?≠0)

Loops:

```
while condition
do
        commands-to-execute
done[<file]
```

```
for var [ in list-of-values ]
do
        commands-to-execute
done
```

Scripts

Parameters:
----
1. The script should get parameters from the command line. and print its name, last parameter and all parameter values.
2. The script should get a file as a parameter and print "The size of ___ is ___ bytes".
3. The script should get a folder as a parameter and change all its contents' ownership to the owner of the dir itself.

Conditions
----
1. The script should get a process name as a parameter and check how many instances it has.
    A. If the process doesn't run the script will notice.
    B. If the process runs between 1 and 10 times, the script will print the number of instances of the process.
    C. If the process runs more than 10 times, the script will ask the user if it should kill the process.
        i. If the user enters 'y' the script will kill the processes.
        ii. If not, the script will print a message saying that this process is funky.

Loops
-----
1. The script gets a file containing a list of users.
    A. The script will check for all users on the list if they exist in the system
    B. If the user is absent, the script will add its name and the date of the check to 'absent_users.log'
    C. If the user exists, the script will copy '~/file' to the user's home folder.
2. The script get a folder containing gzip files, and will add .gz extension to all gzip files that lack it.

```
#--Parameters--

#!/bin/bash
#--1--Prints details on the scripts and its parameters
last=$(echo $* | awk '{print $NF}')
echo "We sent $# parameters into $0. the last parameter is: $last. total values are: $@"

#!/bin/bash
#--2--Prints the size of a specified file
size=$(ls -l $1 | awk '{print $5}')
echo "The size of $1 is $size bytes"

#!/bin/bash
#--3--Set ownership on all of folder's files according to folder owner
owner=$(ls -ld "$1" | awk '{print $3}')
chown $owner "$1"/*


#--ifs--

#!/bin/bash
#--1--Checks the number of instances for a specified process
if [ -n "$1" ]
then
        num=$(ps -e | grep -w "$1" -c)
        if [ $num -eq 0 ]
        then
                echo "$1 isn't running"
        elif [ $num -ge 1 ] && [ $num -le 10 ]
        then
                echo "$1 is running $num times"
        else
                read -p "$1 is running too many time. do you want to terminate? [y/n]:" ans
                if [ "$ans" = "y" ]
                then
                        killall "$1"
                else
                        echo "Over 10 instances of $1 are running, please check."
                fi
        fi
else
echo "No process mentioned"
fi

#--loops--

#!/bin/bash
#--1--Checks if a user exists. if so, copies, if not, logs.
cat "$1" | while read user
do
        if awk -F: '{print $1}' /etc/passwd | grep $user &>/dev/null
        then
                cp "$2" $(grep ^"$user" /etc/passwd | awk -F: '{print $6}')
        else
                echo "$(date):$user doesn't exist" >> absent_users.log
        fi
done

#!/bin/bash
#--2--Rename gzip file to have the extension of .gz
for file in "$1"/*
do
        if [ -f "$file" ] && file $file | grep gzip &>/dev/null
        then
                echo "$file" | grep ".gz" &>/dev/null || mv -T "$file" "${file}.gz"
        fi
done
```