

1. Linux Editors (Update 1)

i a o r c s x d y : / ? <esc> ! w q ZZ

Text Editors

Plaintext

- **emacs** Feature laden environment (See: [<gnu.org/software/emacs/>](http://gnu.org/software/emacs/))
- **vim** (**vi improved**) Standard (See: [<vim.org>](http://vim.org))
- **pico** (**Pine Composer**) and **nano** (**GNU clone replacement**) both for beginners (See: [<guckles.net/pico/>](http://guckles.net/pico/) and [<nano-editor.org/>](http://nano-editor.org/))
- **jed** Software Developer, TeX editor; GUI menu features. (See: [<jedsoft.org/jed/>](http://jedsoft.org/jed/))
- **Reference:** [<www.yolinux.com/TUTORIALS/LinuxTextEditors.html>](http://www.yolinux.com/TUTORIALS/LinuxTextEditors.html)

Text Editors (2)

Graphical

- **gedit** Default Gnome Editor (See: <wiki.gnome.org/Apps/Gedit>)
- **gvim** (graphical vi improved) Based on **vim** (See: <vim.org>)
- **NEdit** (Nirvana Editor) Intuitive, easy to use Motif Editor (See: <<https://en.wikipedia.org/wiki/NEdit>>)
- **tea** Qt based comprehensive editor (See: <tea-editor.sourceforge.net/>)
- **Reference:** <<http://www.yolinux.com/TUTORIALS/LinuxTextEditors.html>>

vi(m)/ex Editor 1

- **vi(m)/ex Modes & States** (See: <arkaye.com/VI1modes.gif>)
- **vi(m) Survival Sheet** (See: <arkaye.com/VI2SurvivalSheet.html>)
- **vi(m) Cursor Positioning on a line** (See: <arkaye.com/VI3CursorPositioning.gif>)
- **vi(m) Window Positioning in a file** (See: <arkaye.com/VI4WindowPositioning.gif>)
- **Problems Opening Files** (See: <arkaye.com/VI5ProblemsOpeningFiles.html>)
- **Problems Saving Files** (See: <arkaye.com/VI5ProblemsSavingFiles.html>)

vi(m)/ex Editor 2

- **Useful ex commands** (See: <arkaye.com/51notes.html>)
- **Keyboard Layout vim commands** (See: <[https://wiki.installgentoo.com/images/1/12/Vim cheat sheet.png](https://wiki.installgentoo.com/images/1/12/Vim_cheat_sheet.png)>)
- **vim tips** (See: <[vim.wikia.com/wiki/Vim Tips Wiki](http://vim.wikia.com/wiki/Vim_Tips_Wiki)>)
- **vim v7.3 FAQ** (See: <vimhelp.appspot.com/vim_faq.txt.html>)
- Run from commandline: **\$ vitutor**
- **Reference:** Neil, Drew (2012). Practical Vim. Pragmatic Bookshelf.

vim Exercises 1

1. Edit a new file, called vimfile, so that it contains the following 7 lines exactly as shown:

Vim stands for Vi IMproved. It used to be Vi IMitation, but there are so many improvements that a name change was appropriate. Vim is a text editor which includes almost all the commands from the Unix program "Vi" and a lot of new ones. All commands can be given with the keyboard. This has the advantage that you can keep your fingers on the keyboard and your eyes on the screen. For those who want it, there is mouse support and a GUI version with scrollbars and menus.

vim Exercises 2

2. How do you do the following? Consider all the reasonable ways to fix the second sentence:

```
Your lucky number is 72436529426013.  
Waacth for it everywhere.
```

There is one method that only requires 3 keystrokes, once the cursor is properly positioned. What is that method?

vim Exercises Answers

1. \$ vim vimfile

Type carefully and/or carelessly. Then press <esc> and move the cursor to each error and correct it.

Your lucky number is 72436529426013.
Waacth for it everywhere.

2. Place the cursor on the second 'a', and type xxp

2. Managing Files, Directories

/ hier mount umount cd pwd

File System Hierarchy Standard

- **Organization of directories under /** (See: https://en.wikipedia.org/wiki/Filesystem_Hierarchy_Standard)
- **Run Linux command: \$ man 7 hier**
(or see: linux.die.net/man/7/hier)
- **mount (8)**: mounts external filesystems to mount points on the root filesystem, /. (Done at boot time [using **/etc/fstab**] or by command) **\$ man 8 mount** (or see: [https://en.wikipedia.org/wiki/Mount_\(Unix\)](https://en.wikipedia.org/wiki/Mount_(Unix)))
- mount points should be empty directories. See also **automount** for automatic mounting on demand
- **umount (8)**: unmounts a mounted filesystem.

Automatic Mounting

- Mounts file systems when needed and unmounts them when no longer being used
- **automount** is the daemon (background system service) that mounts, unmounts file systems
- Default master map (configuration) file is **/etc/auto.master**
- Formats: See **auto.master (5)**; **autofs (5)**
\$ **/etc/init.d/autofs reload** #updated map

Directory Paths

- **Full Path:** filename or directory name starts with a /
- **Relative Path:** filename or directory name starts with anything else but a /
- **Special Characters:**
 - . current directory link
 - .. parent of current directory link
 - ~ your home directory
 - ~userid other user home directory
- Examples: \$ **ls /etc/passwd ./bash_profile ../KatzR ~**
- **cd [lp] {[- or <directory>]}** Make previous or different directory current **{bash builtin command}**
- **pwd [lp]** Show full path current directory name (l symbolic links OK, p symbolic links not shown) **{builtin}**

File Types

- **Symbol** **File Type (See ls -l output, Column 1 character or stat <filename>)**
 - **-** Ordinary File [create via: \$ **touch filename**]
 - **d** Directory [create via: \$ **mkdir dirname**]
 - **b** Block Special Device [e.g. \$ **mknod hdp b 89 64**]
 - **c** Character Special Device [or via \$ **mknod rhdp c 89 64**]
 - **l** Symbolic Link [create via: \$ **ln -s target linkname**]
 - **p** Named Pipe (Inter-process Communication)
[e.g. \$ **mkfifo fifofile p**]
 - **s** Socket (External Communication) [socket package install first]
[e.g. \$ **socket -v server <TCP or domain stream or port No.>**]

Command Syntax

- **Form:** **Cmd-name [wsp option(s)] [wsp parameter(s)]**
- **cmd-name** a special word that represents an executable file or builtin (to memory) program that the Shell will look for and launch
- **wsp** means at least 1 space or tab; **[]** means not required
- **option** a command flavor. short options written as $\{[\pm]\}\{\text{letter}(s)/\text{digit}(s)\}$
option argument via $\{[\pm]\text{letter}\ \text{wsp}\ \text{string}\}$
long options written as $\{--\}\{(\text{hyphenated})\ \text{word}(s)\}$
long option argument via $\{--\text{word}=\text{value}\}$
- **parameter** (or **argument**) is a word that gives information for the command to work with (or on) like a file name or value
- If line ends with a **<return>** or **<enter>** (symbolically **<Ctrl-M>**). This signals the shell to launch the command and execute it, otherwise **** to go on next line and continue the command specification.

Command Syntax Examples

- **good**

- **ls**

ls -a

ls -l

ls -la

ls f1

ls f1 f2 f3

ls -l f1 dir1

ls -l -a f1 dir1

no good

ls -lZ

ls f1 -la f2 [OK in Linux]

-l ls f3

f3 -al ls

ls f 1

Syntax Exercise

1. Write a commandline using tar {look up the man page for it} that has at least two options, one with an option argument and at least one parameter (argument) in your home directory and that executes successfully. How can you tell?

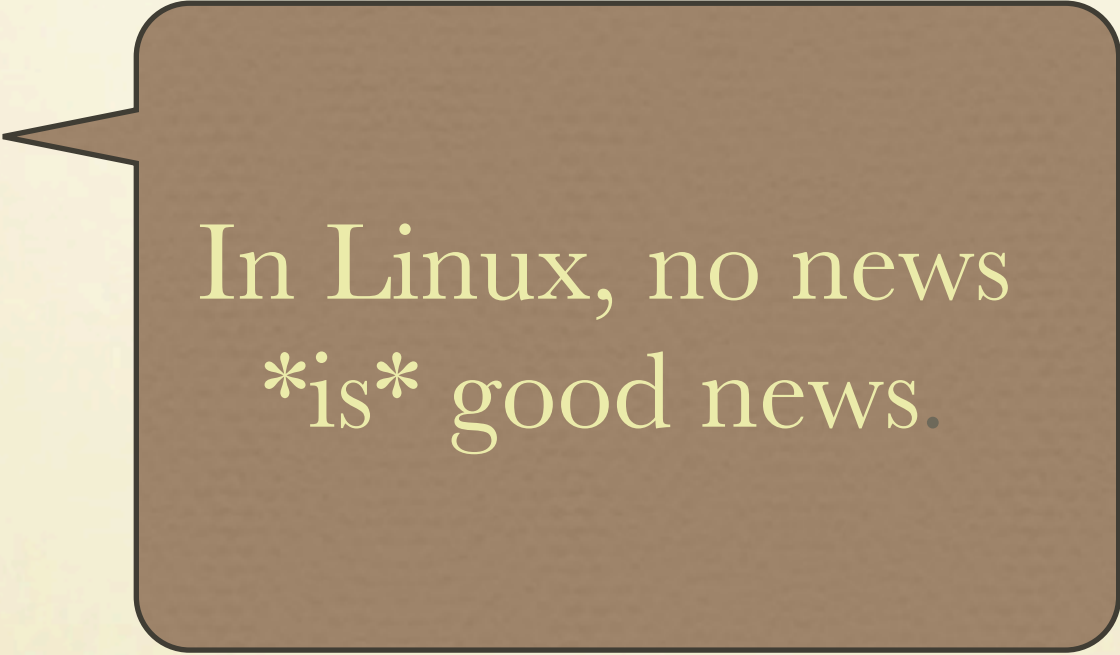
Syntax Exercise Answer

1. Write a commandline using **tar** {look up the man page for it} that has at least two options, one with an option argument and at least one parameter (argument) in your home directory and that executes successfully. How can you tell?

```
$ tar -cvf myfile.tar .bash_profile
```

```
$ tar -tvf myfile.tar
```

Observation



In Linux, no news
is good news.

3. Standard IO & Command Operators

- > 2> >> 2>> < << & | && || ;

Standard IO (STDIO)

- **Standard data streams facility defaults:**
- **Standard Input (STDIN):** Terminal Keyboard device
- **Standard Output (STDOUT):** Terminal Window pseudo device
- **Standard Error (STDERR):** Terminal Window pseudo device
- **Redirection:** Redefining, for the duration of a Linux command, the standard Input, Output and/or Error default destinations to be a logical device or a filename.

STDIO Redirection Examples

- **For STDIN:** `$ cat < ./bash_profile`
- **For STOUT:** `$ echo "A very primitive editor" > textfile`
- **For STDERR:** `$ cat -Z ./bash_profile 2> caterror`
- `$ echo one > a; echo two > b; echo three > c`
`$ cat a - c < b > d`
- **For Appending:** `$ cat b c >> a`
- `$ cat -Y d 2>> caterror`

STDIO Redirection

Example 2

- **For Appended STDIN:**

```
$ cat << EOF
```

```
> Hi
```

```
> there!
```

```
EOF
```

```
Hi
```

```
there!
```

```
$
```

Command Operators

- **Sequential Operator ;**
- **\$ cmd-1 ; cmd-2 ; ... ; cmd-n**
- The shell runs **cmd-1** first. when it is completely finished, **cmd-2** runs next. After it finishes, ..., finally **cmd-n** runs last and finishes.
- The **;** between two commands is identical to **<return>**, **<enter>** or **<Ctrl-M>** in behavior.
- Errors in any **cmd-i** do not abort commandline
- Grouping **()** of commands clarifies precedence

Command Operators 2

- Pipeline (parallel) Operator | [unnamed pipe]
- \$ **cmd-1 | cmd-2 | ... | cmd-n**
- The shell starts all commands, **cmd-i** at once. when **cmd-1** starts producing output, it becomes input to **cmd-2**. ... When **cmd-n-1** starts producing output, it becomes input to **cmd-n** finally **cmd-n** produces STDOUT and finishes last.
- No intermediate output occurs (except STDERR). Any **cmd-i** error aborts commandline. Use **tee (1)** command to snapshot internal output in pipeline.
- Sometimes considered **horizontal programming**

Command Operators 3

- **Background Operator** **&**
- **\$ cmd-1 & [cmd-2 [&]]**
- The shell starts **cmd-1** first, closes STDIN, and **cmd-1** is put in the background. **cmd-2** starts, closes STDIN and finishes in the foreground [or independently in background if 2nd **&** applied]. Either command may finish last.
- The **&** is a suffix of a command.
- Example:
\$ ls -l *.bash > bashscriptlist.txt & date & <return>

Command Operators 4

- Logical AND Operator **&&**
- **\$ cmd-1 && cmd-2 && ... && cmd-n**
- The shell runs **cmd-1** first. when it finishes successfully (true status), **cmd-2** runs next. if **cmd-2** shows a true status, **cmd-3** runs, and so on. If **cmd-i** shows non-success status the command-line finishes; commands to the right do not run.
- Use **\$ echo \$?** to display success status (=0) of the previous command. Any **cmd-i** error aborts commandline to that point.
- Examples: **\$ ping xyz.com && ssh userid@xyz.com**
\$ mkdir xyz && echo \$? && cd xyz

Command Operators 5

- Logical OR Operator `||`
- `$ cmd-1 || cmd-2 || ... || cmd-n`
- The shell runs **cmd-1** first. if it finishes unsuccessfully (non-zero status), **cmd-2** runs next. if **cmd-2** shows a non-zero status, **cmd-3** runs, and so on. The 1st **cmd-i** to show success status (true) stops the commandline; commands to the right do not run.
- Use `$ echo $?` to display previous command success status (**0**). Success status in any **cmd-i** aborts commandline to that point.
- Examples:
`$ (ping xyz.com && echo "Host OK") || echo "Host down"`
`$ ls -l xyz || (echo $? && ls -l)`

Command Operator Exercise

2. Describe (Predict) what the command below does.
Type in the command and execute it.
What is its exact output?
How does the output differ from your description?
Run it again. Does the output order change?
If so, why?

- **\$ (date; cal) & (sleep 15; whoami & uname -a) && echo \$?**

Command Operator Exercise Answers

2. Produces the current date and time and calendar month for July 2015, sleeps for 15 seconds, displays my userid and the Linux Server description and 0. Output order changes based on what else is running

- **\$ (date; cal) & (sleep 15; whoami & uname -a) && echo \$?**

```
[1] 8931
Sun Jul 12 13:30:29 PDT 2015
    July 2015
Su Mo Tu We Th Fr Sa
    1  2  3  4
  5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
u68732801
Linux icpu2520 3.2.65-grsec-infong-14351 #1 SMP Wed Dec 17 19:05:49 CET 2014
x86_64 GNU/Linux
[1]+  Done                  ( date; cal )
0
```

4. File Creation & Viewing

> touch vim

cat less more vim head tail

File Creation

- Use line editors: **ed, ex**
- Use screen editors: **vi, vim**
- Use Commands and Shell facilities
 - empty files: **\$ touch filename**
\$ > filename
\$ cat /dev/null > filename
 - non-empty: **\$ cat > file.txt**
Hi there! <Ctrl-D>
\$ echo "text info" > file.txt

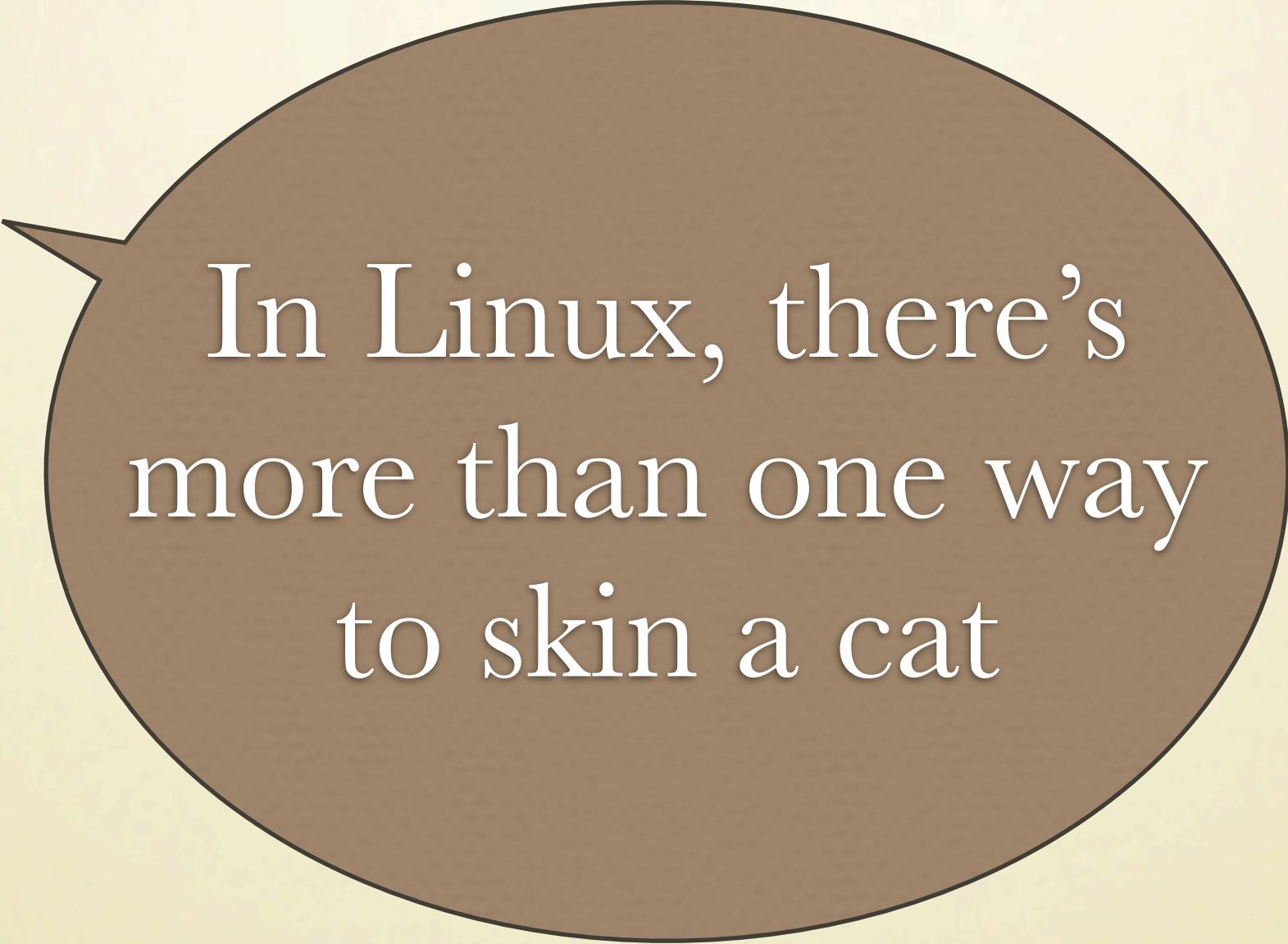
File Content Viewing

- Use screen editors: **vi, vim, view**
- Use pagers: **more, less**
- Use Commands and Shell facilities
 - \$ **cat file ; cat < file**
 - \$ **vim dirname**
 - \$ **head file; tail file; tail -f logfile**
 - \$ **cat -n file; nl -ba file; less -N file**
 - \$ **grep -n \^ file; awk '{print NR, \$0}' file**
 - \$ **sed = < file | sed 'N;s/\n/ /'**

File Links

- Hard Link = A multi-named file in a filesystem
 - Each name points to the File # (inode)
 - File gets removed only when all names deleted
 - Create via `$ ln <target> <nickname>`
 - Determine inode, count via `$ ls -il filename;`
`sudo find / -inum <nnn>`
- Soft Link = Symbolic Link = Shortcut
 - Points to full path file or directory (l type)
 - Can be broken if target moves/goes away
 - Create via `$ ln -s <target> <linkname>`

Observation



In Linux, there's
more than one way
to skin a cat

5. Searching The Filesystem

find xargs locate fuser lsof

find Command

- Searches any part of the file system according to starting directory (or file) and with search criteria in expressions.
- Form: **\$ find <dir | file> [expression(s)]**
- References: <https://en.wikipedia.org/wiki/Find>, <https://danielmiessler.com/study/find/>, <http://alvinalexander.com/unix/edu/examples/find.shtml>, <http://www.grymoire.com/Unix/Find.html>

find Command 2 U1

- **Problem:** If output produced from a (**find**) command overflows the shell's buffer, an error ensues.
Remedy: Pipe **find** (or **cmd**) to **xargs** command.
- Form:
xargs [options] [linux command | /bin/echo]
- Example: \$ **mkdir ~/backups**
\$ **find /fullpath -type f -name '*.*' -print0 |**
xargs -0 -I % cp -a % ~/backups # -print0 = no newline
- Reference: <<https://en.wikipedia.org/wiki/Xargs>>

find Command 3 U1

- Search for a file: `$ find <filename>`
- Files modified range between 6 and 8 days:
`$ find ~ -mtime +6 -mtime -8 -print # newline per result`
`$ touch -t 201507310000 /tmp/EOFjuly`
`$ find . -newer /tmp/EOFjuly`
- Running a command on each found file [slower]:
`$ find / -name '*.html' -exec ls -l {} \;`
`$ find / -name '*.html' -ok rm {} ;`
- Running one command on all found files:
`$ find / -name '*.bash' -print0 | xargs -0 ls -l`
`$ find / -name '*.bash' -print0 | xargs -0 rm -i`

locate Command

- **locate (1)** rapidly searches and outputs files by name (or wildcard pattern) by scanning one or more existing filename databases (created by [**locate.**]updatedb (8) or a **daemon**)
- Form: **\$ locate [option(s)] 'pattern(s)' # text -> *text***
- Examples: **\$ locate bin/zip ; locate zip | grep bin**
\$ sudo locate.updatedb; locate -d mydb: newfile
contents of database controlled by /etc/locate.rc
- Reference: **<www.freebsd.org/cgi/man.cgi?query=locate&sektion=1>**

fuser Command

- Does process searching in a specified file, filesystem, network/port, socket and outputs process numbers along with access and owner information.
- Used to diagnose why “resource busy” errors occur when unmounting filesystems.
- Has builtin terminate process option (-k)
- See Examples in Reference:
<[https://en.wikipedia.org/wiki/Fuser \(Unix\)](https://en.wikipedia.org/wiki/Fuser_(Unix))>

lsof Command

- **lsof (8)** lists open files and the processes that opened them. (Another remedy for “device busy” messages)
- Examples: `$ lsof; lsof /var`
`$ lsof -i -n -P | grep sshd`
`$ lsof -U # find sockets`
- References: Linux Man Page: <<http://linux.die.net/man/8/lsof>>, General Info: <<https://en.wikipedia.org/wiki/Lsof>>, Examples: <<http://www.thegeekstuff.com/2012/08/lsof-command-examples/>>

6. Searching inside files

regex grep awk sed perl

Regular Expression Patterns

- **Definition:** Special characters used to identify patterns in text.
- Special Characters: `^ $. * [] - \ { } ? + () |`
- Multi-Language Tables: www.regexegg.com/regex-quickstart.html#ref
- Descriptions: stackoverflow.com/questions/4736/learning-regular-expressions

Regex Patterns 2

- Character Classes: `[:alnum:]` = [A-Za-z0-9]
`[:word:]` = `[:alnum:]_` `[:alpha:]`=[A-Za-z]
`[:cntrl:]`=Ascii 0-31,127 `[:digit:]`=[0-9]
`[:blank:]` = [SpaceTab] `[:graph:]`=Ascii 33-126
`[:space:]`= `[:blank:]<\r><\n><\v><\f>`
`[:lower:]` = [a-z] `[:upper:]`=[A-Z]
`[:punct:]` = [-!"#\$%&'()*+,-./:;<=>?@[_`{|}~]
`[:print:]` = `[:graph:]<Space>`
`[:xdigit:]` = [A-Fa-f0-9]
- Regular Expression Tables: <http://www.arkaye.com/regextables.html>

Regex Exercises

- (1) What do the following regular expressions match?

`/^cat$/`

`/^$/`

`/^/`

- (2) `$ cat word.list`

Iraqi

Iraqian

miqra

qasida

qintar

qoph

zaqqum

Iraq

Qantas

Quinto

quote

What does `/q[^u]/` match within `word.list`?

How can you match `Iraq` too?

How can you match `Qantas` as well?

- (3) Write a regular expression that will match any date of the form:

`mm/dd/yy`, `mm-dd-yy`, `mm.dd.yy`

Regex Exercises Answers

- (1) What do the following regular expressions match?

```
/^cat$/      # Exactly the word cat on a line
/^$/        # Only empty lines with no characters
/^/         # Matches all lines
```

- (2) What does `/q[^u]/` match within `word.list`?

all but **Q**antas, **Q**uinto, **I**raq, **q**uote

How can you match `Iraq` too? Use `/q[^u]*/`

How can you match `Qantas` as well? Use `/[Qq][^u]*/`

- (3) regex for `mm/dd/yy` or `mm-dd-yy` or `mm.dd.yy`

```
#([ 012][:digit:]/[ 123][:digit:]/[:digit:]{2})|
([ 012][:digit:]-[ 123][:digit:]-[:digit:]{2})|
([ 012][:digit:]\.[ 123][:digit:]\.[:digit:]{2})#
```

grep Command

- Searches text file(s) [or STDIN] using a given pattern and displays matched lines.
- **grep** ported as OS Command from the ed/ex editor.
:g/re/p # global search for the “re” [or regexp] in the current file and **print** (display) matched lines.
- Form: **grep [Option(s)] ‘regexp pattern’ [File(s)]**
grep (1) man page: <linux.die.net/man/1/grep>
- Useful Options: **-v** non-matches, **-l** show filenames only, **-c** Count matches, **-h** no filenames, **-w** match whole word, **-x** match whole line
- Variants: **egrep** [or **-E** extended], **fgrep** [or **-F** fixed lines], **agrep** [approximate], **pgrep** [process id output], **zgrep** [compressed files]

grep Command 2

- Examples: `$ grep -h '.zip' dirlist.txt`
`$ grep -E -h '^ (zip | zip$ | ^zip)$' dirlist.txt`
`$ grep -h '[^bg]zip' dirlist.txt`
`$ grep -i '^..j.r$' /usr/share/dict/words`
`$ grep -h '^[:upper:]' dirlist.txt # [A-Z]`
`$ grep -h '^[-AZ]' dirlist.txt`
`$ echo "(555) 123-4567" |`
`grep -E '^ \((?[0-9]{3}\) ? [0-9]{3}-[0-9]{4}$'`
`$ echo -e "a b 9\na b d" | grep -E '^ ([[alpha:]]`
`+ ?)+$'`

grep Exercises

- 1. Suppose you have a file, passwordfile2:

```
$ cat passwordfile2
```

```
eineengae3 da7ahkae8e xeizoor5jo chaizaegh4 efeem8aba6 cee7aez1oh  
vureich8ay eeji8eisho shae0ap8sh mubeghoo6e igheeb7aiy eeb6ieba3m  
ua0queeroh neih8nahxa ahx8aeyahg mohghee5ta choo2liet7 ootei2faeg  
boore5tezu yaey3ohrai ees7aej2za xo3iezothu ur9ahteive pohquai3ri  
mubath0eiv aexool9thi ko9thahlah aikie5leir bei8oapohp lo6ishaesh  
xaech2eith cheiz5ohqu aa1aziusii ooth0luega noophuv7ih aiv5jude4w
```

- a) Use **grep** to match lines with the digits in sequence: **8** then **8**, **7** then **2**, **5** then **2**, in any order when letters are ignored.
- b) Use **grep** to match lines with the double letters: **ee**, **oo**, **ee** in that order when digits are ignored.

grep Exercises Answers

- `$ cat passwordfile2`

```
eineengae3 da7ahkae8e xeizoor5jo chaizaegh4 efeem8aba6 cee7aez1oh  
vureich8ay eeji8eisho shae0ap8sh mubeghoo6e igheeb7aiy eeb6ieba3m  
ua0queeroh neih8nahxa ahx8aeyahg mohghee5ta choo2liet7 ootei2faeg  
boore5tezu yaey3ohrai ees7aej2za xo3iezothu ur9ahteiwe pohquai3ri  
mubath0eiv aexool9thi ko9thahlah aikie5leir bei8oapohp lo6ishaesh  
xaech2eith cheiz5ohqu aa1aziusii ooth0luega noophuv7ih aiv5jude4w
```

- a) match lines with: **8** then **8**, **7** then **2**, **5** then **2**, in any order, ignoring letters.

```
$ grep '8[a-z ]*8' passwordfile2 | grep '7[a-z ]*2' | grep '5[a-z ]*2'
```

```
ua0queeroh neih8nahxa ahx8aeyahg mohghee5ta choo2liet7 ootei2faeg
```

- b) match lines with double letters: **ee**, **oo**, **ee** in that order ignoring digits.

```
$ grep 'ee.*oo.*ee' passwordfile2
```

```
eineengae3 da7ahkae8e xeizoor5jo chaizaegh4 efeem8aba6 cee7aez1oh  
vureich8ay eeji8eisho shae0ap8sh mubeghoo6e igheeb7aiy eeb6ieba3m
```

Observation

When it all
gets to you,
remember:
it's only ones
and zeroes.

awk Command

- Searches text file(s) [or STDIN] using a given pattern and produces output according to a prescribed program of actions.
- **awk** is considered a non-procedural programming language. Successive **'pattern { action(s); }'** pairs are written in any order.
- **awk** extends **grep**; is a pipeline filter; computes real numbers; processes numerical, text files, can break up its input into lines, fields.
- Form: **awk [Option(s)] '[pattern] [{ action(s); }]' [File(s)]**
awk (1) man page: <linux.die.net/man/1/awk>
- **awk tutorials:** <<http://www.tutorialspoint.com/awk/>>, <http://www.vectorsite.net/tsawk_1.html>
- **Reference: Robbins, Arnold (2015). Effective awk Programming, 4th ed. O'Reilly.**

awk Examples

```
• $ awk '/Albuquerque/' cities
Albuquerque, New Mexico
$ grep 'Albuquerque' cities
Albuquerque, New Mexico
$ awk '{print}' people
John,P.      Physics      20
Rick,L.      Mechanical  21
Jack,T.      electrical  23
Larry,M.     Chemical    22
Mary,Q.     Electrical  21
Betty,B.     Math        22
Mike,A.     Civil       23
Ruth,M.     Math        21
Tony,R.     English     22
Sarah,S.    Electrical  24
$ awk '$3 > 22 { print $1 }' people
Jack,T.
Mike,A.
Sarah,S.
$ awk '$2 ~ /[Ee]lectrical/ { print $1 }' people
Jack,T.
Mary,Q.
Sarah,S.
$ awk '$0 ~ /^J.*/ { print $0 }' people
John,P.      Physics      20
Jack,T.      electrical  23
```

awk Examples 2

```
• $ cat awkprog1
  { sum += $3
    ++no
  }
  END { printf "The average age is %.2f\n", sum/no }
$ awk -f awkprog1 people
The average age is 21.90
$ cat awkprog2
BEGIN { print "This is another variation" }
{ sum += $3
}
END { printf "The average age is %.2f\n", sum/NR }
$ awk -f awkprog2 people
This is another variation
The average age is 21.90
$ awk '{ if ($1 ~ /^J/) printf "%s\n", $0 > "Jfile"
> if ($2 ~ /^C/) printf "%s\n", $0 > "Cfile" }' people
$ cat Jfile
John,P.      Physics      20
Jack,T.      electrical   23
$ cat Cfile
Larry,M.     Chemical    22
Mike,A.     Civil       23
```

awk Examples 3

```
• $ cat awkprog3
{ pos = index($1,",");
  lname = substr($1,pos+1,2);
  fname = substr($1,1,1);
  major = substr($2,1,3);
  name = fname lname;
  printf "%s\t%s\t%s\n", name, major, $3 | "sort"
}
$ awk -f awkprog3 people
BB.  Mat  22
JP.  Phy  20
JT.  ele  23
LM.  Che  22
MA.  Civ  23
MQ.  Ele  21
RL.  Mec  21
RM.  Mat  21
SS.  Ele  24
TR.  Eng  22
$ ls -la | awk '$1 ~ /^d/ { print $9 "/" } $1 ~ /^-..x/ { print $9 "*" }'
```

sed Command

- A **s**tream **e**ditor that parses and transforms lines in text file (s) or STDIN. **sed** is like **awk**, but has more short, cryptic commands. “Command garbled” is only error message.
 - Forms: (See man page: <linux.die.net/man/1/sed>)
\$ **sed** [option(s)] -e "sed command" ... [file(s)]
\$ **sed** [option(s)] -f sedscriptfile [file(s)]
 - Editing Format: [address1[,address2]] command arguments
Delete Lines: [address1[,address2]] **d**
Add Lines: [address] **a** \ [address] **i** \
 text **text**
Substitution: [address1[,address2]] **s**/**pattern/new/[flag]**
- References: <<https://en.wikipedia.org/wiki/Sed>>, Tutorial:
<<http://www.grymoire.com/Unix/Sed.html>> or Book:
Dougherty, D., & Robbins, A. (1997). sed & awk, 2nd Ed. O'Reilly

sed Examples

```
• $ cat sedscript
/^d/d          # delete all lines starting with d
10d           # delete line 10
3,7d         # delete lines 3 through 7
$ ls -l | sed /^d/d      # list only files, no directories
$ cat file | sed -e1d -e13d # executes deletes of lines 1, 13
$ cat sed.example
This is line 1
This is line 2
This is line 3
This line is in Turkish when you are not looking at it
$ cat sed.example | sed 'a\
-----
This is line 1
-----
This is line 2
-----
This is line 3
-----
This line is in Turkish when you are not looking at it.
-----
$ sed -e '1,3d' < sed.example
This line is in Turkish when you are not looking at it.
```

sed Examples 2

- `$ sed -n '3,4p' sed.example`

```
This is line 3
```

```
This line is in Turkish when you are not looking at it.
```

```
$ sed -e 's/in Turkish/Greek/g' sed.example
```

```
This line is Greek when you are not looking at it.
```

```
$ sed -e 's/This is.*/& &/g' -e '3q' sed.example
```

```
This is line 1 This is line 1
```

```
This is line 2 This is line 2
```

```
This is line 3 This is line 3
```

```
$ cat numberfile
```

```
6 9 13 17 23 41
```

```
5 6 7 8 9 0 1 2 3 4
```

```
$ sed y/0123456789/5678901234/ numberfile
```

```
1 4 68 62 78 96
```

```
0 1 2 3 4 5 6 7 8 9
```

```
$ sed y/[0-9]/5678901234/ numberfile
```

```
command garbled: y/[0-9]/5678901234/
```

```
$ sed -nr 's/^PRETTY_NAME=(.*)/\1/p' /etc/os-release
```

```
"openSUSE 13.2 (Harlequin) (x86_64)"
```

perl Command

- **p**ractical **e**xtraction & **r**eport **l**anguage (Larry Wall)
 - a procedural, object oriented, interpretive language
 - adapts other language libraries (e.g. C, sql)
 - is network aware and favored by System Administrators
 - has no built in limits, unlike when awk, sed, bash fail
 - has many one-line programs for the command line.
- Form: (See man page: <<http://linux.die.net/man/1/perl>>)
`$ perl [option(s)] -e "perl command"...[file(s)]`
`$ [[perl] [option(s)]] script.pl [args] [file(s)]`
- Perl 1-liner examples: <<http://www.socher.org/index.php/Main/PerlScriptsAndOneLiners>>

References: <<https://en.wikipedia.org/wiki/Perl>>, Tutorial: <<http://learn.perl.org/tutorials/>> or Book:

7. Linux Commands, Utilities

Coreutils Util-Linux

Commands by Category 1

Category	Commands
Output file all contents	cat tac nl od shuf <i>rev</i>
Format file contents	fmt pr fold
Output file partially	head tail split csplit <i>less more</i>
Summarizing files	wc sum cksum md5sum sha1sum <i>file</i>
Operating on (sorted) files	sort uniq comm ptx <i>diff</i> <i>cmp</i> tsort
Operating on fields in a line	cut paste join
Operating on characters	tr expand unexpand
Listing Directory Contents	ls dir vdir

Commands by Category 2

Category	Commands
Basic Operations	cp mv rm dd install shred mktemp
Special File Types	ln mkdir rmdir mkfifo mknod
Search	find locate grep awk which whatis whereis
Changing file attributes	chown chgrp chmod touch umask
Disk Usage	df du stat sync
Outputting Text	echo printf yes
Conditionals	false true test [] expr [[]] (())
Redirection	tee < > >> << 2> 2>>

Commands by Category 3 ^{U1}

Category	Commands
File Name Manipulation	basename dirname pathchk
Environment Context	cd pwd stty tty printenv env
User Information	id logname whoami groups users who finger pinky w
System Context	date cal uname hostname hostid getlimits uptime nproc
Modified Command Invocation	chroot nice nohup sleep time set[ug]id timeout su sudo
Process Control	kill ps jobs bg fg
Numeric Operations	factor seq expr bc dc base64

Commands Exercises

- 1. Write a commandline that outputs the inode numbers of the root directory / and the directory of user accounts /home
- 2. Create a file in your home directory called tryit&. Rename it to be called triedit!
- 3. What happens and why does it happen when you type each of the following:
 - \$ **date > date**
 - \$ **date < date**
 - \$ **cat < date > nodate**
 - \$ **cat date < ~/.bashrc**

Commands Exercises Answers

- 1. Write a commandline that outputs the inode numbers of the root directory / and the directory of user accounts /home
\$ ls -ild / /home
- 2. Create a file in your home directory called tryit&. Rename it to be called triedit!
\$ touch 'tryit&' ; ls -l tr* ; mv 'tryit&' triedit! ; ls tr*
- 3. What happens and why does it happen?
\$ date > date # date and time written in date file
\$ date < date # current date and time output, STDIN ignored
\$ cat < date > nodate # contents of date written to nodate
\$ cat date < ~/.bashrc #old date, time output, STDIN ignored

Useful Commandlines

commandline	Result
<code>grep -A1 -B1 'pattern' files</code>	show line before & after each matched line
<code>grep -no 'pattern' files</code>	show line number and pattern only of every match
<code>ls -lahS; ls -lahSi</code>	Show sorted byte sizes in K,M,G units (base 2, base 10)
<code>ls -lRs directory sort -rn head</code>	show the top 10 largest files in directory
<code>strings binaryfile</code>	show strings of printable characters in binary files

rsync Command

- **rsync (1)** copies (synchronizes) files/directories very fast to/from a remote host, or locally on current host [or displays source]
- Form: `$ rsync source | remsource [remdestination | destination]`
- Features:
 - copies links, devices, owners, groups, permissions
 - offers exclusion options
 - can use bash, ssh or rsh
 - no superuser privileges needed
 - pipelining of file transfers to minimize latency
 - supports anonymous/authenticated rsync daemons for mirroring
- Examples:
 - `$ rsync -t *.c remotehost:src/`
 - `$ rsync -avz remotehost:src/bar/ /tmp/data`
 - `$ rsync -av /src/foo /dest #same as # $ rsync -av /src/fool /dest/`

tar Command

- **tar (1)**: a tape or disk archive utility used to transfer collections of files as a single entity to a (remote) destination.
- Form: `$ tar [ctxru][v][zjZpW][f device | file] source`
- Features:
 - Used for distribution or backup
 - Writes sequentially to sequential I/O devices or files
 - Local files transfer via:
`$ tar cf - srcdir | (cd destdir && tar xv)`
 - provides default `ls -l` format for archive contents
 - offers extended tar-format [pax-format] like arbitrary resolution, unlimited length and char set encoding for file attributes.
- Problems:
 - extraction can overwrite files
 - extraction of absolute path files can damage filesystem
 - sequential rather than random access, so slow
 - permits duplicates in archive (last file mentioned wins)

tar Command (2)

- Equivalent File Suffixes

Short	Long
.tgz	.tar.gz
.tbz, .tbz2, .tb2	.tar.bz2
.taz, .tz	.tar.Z
.tlz	.tar.lz, .tar.lzma
.txz	.tar.xz

tar Examples

- `$ tar cvf archive.tar source # create archive`
- `$ tar tvf archive.tar # show archive contents`
- `$ tar xvf archive.tar directory # extract archive`
- `$ tar xvf archive.tar directory/specialfile #get 1 \ file`
- `$ tar xvzf archive.tgz directory # uncompress and \ extract archive`
- `$ tar xvf archive.tar --wildcards 'dir/*.c' \ # selectively extract all C archived source files`
- `$ tar rvf archive.tar README.txt # append file`
- `$ tar tvWf archive.tar # verify uncompressed \ archive`

Compression Utilities

Command	Suffix	Compress Ratio
compress, uncompress. zcat	*.Z	fast, min.memory, lower CR
xz, unxz, xzcat	*.xz	lossless, higher memory, CR
gzip, gunzip, zcat	*.gz	fast, space efficient, improved CR
bzip2, bunzip2, bzcat	*.bz2	slower comp., faster decomp. high CR

8. Linux Permissions

`rxwxrwsrwt`

Linux Permissions

- 95% of Linux errors due to permissions problems (Anecdotal)
- Gives security for files and directories
- Letter (**r**, **w**, **x**, **s**, **t**) implies permitted
- Dash (-) implies not permitted
- User (owner), group, others categories
- `ls -l` shows permissions string: **rw-rw-r--**

Linux Permissions (2)

- To view permissions, run: `$ ls -l`
- To change permissions, run:
chmod (1) changes file modes (perm. string)
umask (1) get/set file mode creation mask
- To influence permissions, run:
chown (1) change file owner (and group)
chgrp (1) change file group owner
newgrp (1) make another group primary (login)

Linux Permissions (3)

Access	For Files	For Directories
r Read	File contents is readable, so these work: cat, more, cp, <, sort, grep	Directory contents is readable, so this works: ls
w Write	File contents can change, so these work: >>, vim, ed	Directory contents can change, so these work: cp, mv, ln, rm, >, mkdir, rmdir, touch

Linux Permissions (4)

Access	For Files	For Directories
x Execute/ Access	File contents executes, so scripts work (execute filename as a command)	Directory contents can be accessed, so this works: cd
s Set UserID/ GroupID	Execute the file as a command and assume the privileges of the owner/ group only while it runs	setgid means that new files in the directory will inherit the directory's group, not the file creator

Linux Permissions (5)

Access	For Files	For Directories
t Sticky Bit	A (large) file, when executed and then exited, would remain in memory in case it is run again (obsolete)	Directory name, files within or their existence can only be changed by the super user or file/directory owner

Web, ftp Permissions

- Web Client viewer “nobody” in other category
- Default Web directory is `~/public_html`
- Web pages (`*.htm`, `*.html`, `*.css`, `*.jpg`, `*.gif`) need `rw-r--r--` (644) permissions
- Executable Web Programs (`*.cgi`, `*.bash`, `*.pl`) need `rwxr-xr-x` (755) permissions
- Files transferred via **ftp** have default permissions at the destination based on `umask` value like `066`, giving `rw-----` (600) permissions

Setting Permissions

- Creation default: **umask 022: -rw-r--r--, drwxr-xr-x**
- **\$ chmod 644 file; chmod 755 dir; ls -ld file dir**
-rw-r--r-- 1 katz katz 0 Aug 1 06:42 file
drwxr-xr-x 1 katz katz 512 Aug 1 06:41 dir
- **\$ chmod u=rw,g=r,o=r file; ls -l file**
-rw-r--r-- 1 katz katz 0 Aug 1 06:42 file
- **\$ chmod u+rw,g+r,o-r file; ls -l file**
-rw-r----- 1 katz katz 0 Aug 1 06:42 file
- **\$ chmod u+rw,g+r,o+r,a+x dir; ls -ld dir**
drwxr-xr-x 1 katz katz 512 Aug 1 06:41 dir

Setting permissions Octally

User	3x3 octal digit strings	3 digit Octal
owner (u)	r=400, w=200, x=100 (s=4100)	700 (4700)
group (g)	r=040, w=020, x=010 (s=2010)	070 (2070)
other (o)	r=004, w=002, x=001 (t=1001)	007 (1007)

Octal 765 = (4+2+1)(4+2+0)(4+0+1) (4+4+2+1)(2+4+2+0)(1+4+0 +1)

r	w	x	r	w	-	r	-	x	r	w	s	r	w	S	r	-	t		
<u>1</u>	<u>1</u>	<u>1</u>	<u>1</u>	<u>1</u>	<u>0</u>	<u>1</u>	<u>0</u>	<u>1</u>	<u>1</u>	<u>1</u>	<u>1</u>	<u>1</u>	<u>1</u>	<u>1</u>	<u>0</u>	<u>1</u>	<u>1</u>	<u>0</u>	<u>1</u>
7			6			5			7	7		6			5				

Permission Exercise

- 1. Directory permissions for dir1 are: **drwxr-x--x**
File permissions for file1 in dir1 are: **-rw-rw-r--**
- What can each category of user do with these file objects?

owner: ___ modify ___ delete ___ read ___ execute

group : ___ modify ___ delete ___ read ___ execute

others: ___ modify ___ delete ___ read ___ execute

Permission Exercise (2)

- 2. Directory permissions for dir1 are: **drwxr-x--x**
File permissions for file2 in dir1 are: **-rwxr-xr-x**
- What can each category of user do with these file objects?

owner: ___ modify ___ delete ___ read ___ execute
group : ___ modify ___ delete ___ read ___ execute
others: ___ modify ___ delete ___ read ___ execute

Permission Exercise (3)

- 3. Directory permissions for dir2 are: **drwxrwxr-x**
File permissions for file3 in dir2 are: **-rwxr-xr-x**
- What can each category of user do with these file objects?

owner: ___ modify ___ delete ___ read ___ execute
group : ___ modify ___ delete ___ read ___ execute
others: ___ modify ___ delete ___ read ___ execute

Using ACLs (Access Control Lists)

- ACLs allow permissions to be given/denied to more than one user/group for a file/directory. (e.g. 1 group has full control, another has read only, a third has no access). See **acl (5)**
- ACLs allow for permission inheritance, when new files/directories are created in an ACL directory.
- ACLs are supplemental to Linux permissions.
- **getfacl (1)** and **setfacl (1)** are used to view current ACLs and to set them for files or directories.

ACL Examples

- `$ setfacl -R -m u:katz:rx dir # katz gets read.access for dir and recursively below`
- `$ setfacl -x u:katz:r file # undo the above permission`
- `$ setfacl -x g:users file # users group removed for file`
- `$ setfacl -m m::rx file # remove for all groups and all\n named users, write perm. for file`
- `$ getfacl file1 | setfacl --set-file=file2`
copy current ACL of file1 to that of file2

ACL Exercise

- 1. Type the command **setfacl -m g:www:rx ~/public_html**
and **setfacl -m g:games:rwX ~/.local**
this sets the ACL on directory names only.
- 2. Type the command **setfacl -m d:g:www:rx ~/public_html**
and **setfacl -m d:g:games:rwX ~/.local**
this sets the default ACL on directory contents.
- 3. Type the command **getfacl ~/.local** # Verify correctly set
- 4. What results using **touch ~/.local/somefile ; ls -la ~/.local/somefile**

File System Attributes

- Attributes restrict files/directories, regardless of userid (even root) outside of the file system.
- **lsattr (1)** and **chattr (1)** list and change file/directory attributes.
- 3 Useful Attributes to apply:
 - immutable (**i**) - can't change, delete or rename file
 - append only (**a**) - can modify but not delete file
 - undeletable (**u**) - can't delete file
- Uses: storing author of document; character encoding of plain text document, checksum, cryptographic hash, digital certificate
- Example:

```
$ var="~/katz/.local/somefile"; ls -al $var; $ lsattr $var  
$ chattr -V +i +u $var; echo hi >> $var  
$ chattr -V -i -u $var; lsattr $var # undoes previous line
```

9. Linux Processes

`^C ^D ^Z ^\ ; & () { } bg fg fuser jobs (p)kill
nice nohup ps sleep pstree pgrep pmap top`

Process Scheduling

- The Linux Kernel combines “First Come First Serve” and “Round-Robin” Scheduling
 - individual process tasks
 - groups of tasks owned by particular users
 - Kernel control groups.
- The method used is known as the Completely Fair Scheduler (CFS) to optimize scheduling.

Linux Process States

- See: arkaye.com/unix/PSESD60807/LinuxProcessStates.gif
- **process group**: a collection of processes that are all affected by a single signal. Process groups are themselves grouped into **sessions**
- Examples: `$ (pwd; cd / ; date; ls)`
`$ cat file | tee file2 | wc -l | sleep 60 &`

Process Attributes

- PID Process ID, a unique No. based on boot uptime
- PPID Process' parent process or creator
- UID User ID, owner number of the process
- EUID Effective User ID
- GID Group ID, group number of owner
- EGID Effective Group ID
- Nice Priority: (+19 to -19) [low-high]
- Control Terminal: sets defaults for STDIN, STDOUT, STDERR

ps, top Commands

- **ps (1)** displays the process status of tasks, jobs associated with your userid, your terminal, those of others, and/or all processes using snapshot monitoring.
\$ **ps -aef** # show all processes
\$ **ps** # show your processes
- **top (1)** recurrently monitors CPU activity in real time, showing the status of the 15 most CPU intensive processes.
\$ **top** shows summary information followed by a table of PID, User, Priority, Nice, and %CPU utilization
[type q to quit **top**]

Job Control

- **jobs** shows the status of jobs (processes) started in the current shell environment
- In **jobs** output: { [N]{+|-} Status Command line }
Current job marked with +
Previous job marked with -
- **bg [%"jobid"]** puts foreground job in background while it is running
- **fg [%"jobid"]** puts background job in foreground while it is running
- **<Ctrl-Z>** suspends current foreground job; **fg** resumes job

Abnormal Termination

- **kill (1)** Sends a stop signal to running process(es)
- **\$ kill [-[s]{ SignalName | SignalNumber}] PID | %jobno** or
\$ kill [{-SignalName | -SignalNumber}] PID | %jobno
- To list and match Signal numbers with names, use **\$ kill -l**
\$ kill -9 PID # an irresistable signal, which can't be caught
When PID = **0**, all processes in current process group are signaled
When PID = **-1**, all processes with a pid more than **1** are signaled
When PID = **-n**, all processes in the process group **n** are signaled
- **<Ctrl-C>**, **<Ctrl-\<>** stops a foreground process
- **<Ctrl-D>**, **exit** stops (exits from) your login or (sub)shell session
- **\$ nohup command & #** runs command in the background and ignores the hangup signal (signal **1**)

Managing jobs Exercise

- 1. From a graphical user interface [gnome], open a terminal and from it, start the **gedit** program. This prevents the terminal from starting any additional programs.
- 2. Click in the terminal where you started **gedit** and type **<Ctrl-Z>**. Make sure you get a prompt in the terminal.
- 3. Use the **bg** command to make **gedit** a background job (use **\$ bg %1**)
- 4. In the terminal window, type **jobs** on the commandline. You should see the **gedit** program in the list and its job number. (If you only have a single job, its number is always 1.)
- 5 To put a background job in the foreground, use the **fg** command. **fg** without arguments uses the most recent background job, otherwise use **fg %n**

/proc (procfs) Filesystem-1

- **proc (5)** A direct reflection of the system kept in memory and represented hierarchically. See also **procinfo (8)**
- An interface to Kernel data structures ([/dev/kmem](#))
- Used to get information about the system and to change certain kernel parameters at runtime via **sysctl**

/proc (procfs) Filesystem-2

- Used for system related tasks as:
 - viewing statistical information
 - finding out about hardware
 - modifying runtime kernel parameters
 - Viewing, modifying network & host parameters
 - Viewing memory and performance information
- **\$ procinfo** or
\$ procinfo -n15 # gives 15 second updates

/proc (procfs) Filesystem-3 U1

- `$ ls -la /proc | less # selected results`
total 16
dr-xr-xr-x 292 root root 0 Jul 28 09:21 .
dr-xr-xr-x 9 root root 0 Jul 28 09:21 1
dr-xr-xr-x 9 root root 0 Jul 28 hh:mm **pids**
dr-xr-xr-x 9 katz users 0 Jul 29 09:26 8508
dr-xr-xr-x n root root 0 Jul 28 hh:mm **hwmodules**
-r-xr-xr-x 1 root root 0 Jul 28 hh:mm **modules**
lr-xr-xr-x 1 root root 0 Jul 29 09:26 self -> 8508

/proc (procfs) Filesystem-4

- `$ ls /proc/8508` # short list; red=symlinks, blue=directories

<code>attr</code>	<code>fd</code>	<code>mountstats</code>	<code>schedstat</code>
<code>autogroup</code>	<code>fdinfo</code>	<code>net</code>	<code>sessionid</code>
<code>auxv</code>	<code>gid_map</code>	<code>ns</code>	<code>smaps</code>
<code>cgroup</code>	<code>io</code>	<code>numa_maps</code>	<code>stack</code>
<code>clear_refs</code>	<code>latency</code>	<code>oom_adj</code>	<code>stat</code>
<code>cmdline</code>	<code>limits</code>	<code>oom_score</code>	<code>statm</code>
<code>comm</code>	<code>loginuid</code>	<code>oom_score-adj</code>	<code>status</code>
<code>coredump_filter</code>	<code>map_files</code>	<code>pagemap</code>	<code>syscall</code>
<code>cpuset</code>	<code>maps</code>	<code>personality</code>	<code>task</code>
<code>cwd</code>	<code>mem</code>	<code>projid_map</code>	<code>timers</code>
<code>environ</code>	<code>mountinfo</code>	<code>root</code>	<code>uid_map</code>
<code>exe</code>	<code>mounts</code>	<code>sched</code>	<code>wchan</code>

- `$ cat /proc/8508/limits | fmt` # shows Limit name, Soft, Hard Limit
- `$ cat /proc/8508/mountinfo | less` # shows all mounted filesystems

Processes Exercises

- 1. Use **ps** to display the status of processes you own that are running in your login session. Which one is your login shell? How can you tell, if more than one shell running?
- 2. Run:
\$ (sleep 60; nohup find / -name '*sh' -print > findout 2> /dev/null)&
Use **ps** to verify that it is running. Logout. Login again. Use **ps -a** to see if the **find** command is still running.
- 3. Give an example of a Linux process that doesn't terminate when **<Ctrl-C>** is pressed.

Processes Exercises Answers

- 1. `ps -eaf | grep '^katz' | tail -6` # First bash is parent and login shell launched from gnome-terminal-server process

```
katz 1786 1 0 Jul28 ? 00:00:38 /usr/lib/gnome-terminal-server
katz 1789 1986 0 Jul28 ? 00:00:00 gnome-pty-helper
katz 1790 1786 0 Jul28 pts/0 00:00:00 bash
katz 8241 1790 0 09:29 pts/0 00:00:00 bash
katz 10068 8241 0 12:27 pts/0 00:00:00 ps -eaf
katz 10069 8241 0 12:27 pts/0 00:00:00 grep --color=auto ^katz
```
- 2. It's still running due to `nohup find` command prefix being part of a background process group
- 3. The `bash` login shell and subshells are immune to `<Ctrl-C>` signals, due to its builtin `trap` handler

process Priority

- **nice (1)** runs a command or shell with a lower priority (if a user) or changed priority (if root) as the command starts (modifiable in </etc/security/limits.conf>)
- **renice (1)** runs a process PID with a lower priority (if a user) or changed priority (if root) **while** the associated command runs
- **nice** means you are being “nice” to other users by reducing your job’s priority (+5 means lowering priority by 5)

process Priority Example

- `$ nice -n 15 sleep 50 &`
[1] 13343
`$ ps -eo pid,ni,pri | grep -A1 -B1 13343`
13322 0 19
13343 15 4
13349 0 19
- `$ sleep 60 &`
[1] 13279
`$ renice +5 13279`
13279 (process ID) old priority 0, new priority 5
`$ ps -eo pid, ni, pri | grep -A1 -B1 13279`
13278 0 19
13279 5 14
13297 0 19

Process Priority Exercise

- 1. Open a terminal window and use **su -** to become root.
- 2. At the # prompt, type **dd if=/dev/zero of=/dev/null & #** repeat 4 times
- 3. Type **top** at the # prompt. You should see 4 dd commands listed highest. In the **PR** column the priority of all of these processes is set to **20**. The **NI** column shows the actual niceness at **0**. The **%CPU** column shows the utilization of the CPU, which is similar for these 4 highest processes.
- 4. From within top, press **r** (renice). It suggests a PID. Select it or type one of the others <enter>; type **5** as the (lowered) renice value <enter>. Observe the effect on the **%CPU**.
- 5 Repeat the procedure with one of the other **dd** processes. Use -10 as the (raised) renice value. Observe the effect on the **%CPU**.
- 6. From within top, press **k** (kill). It suggests a PID. Select it or type one of the others associated with **dd** <enter>. Terminate all the other **dd** processes in the same way. Then exit from the root shell.