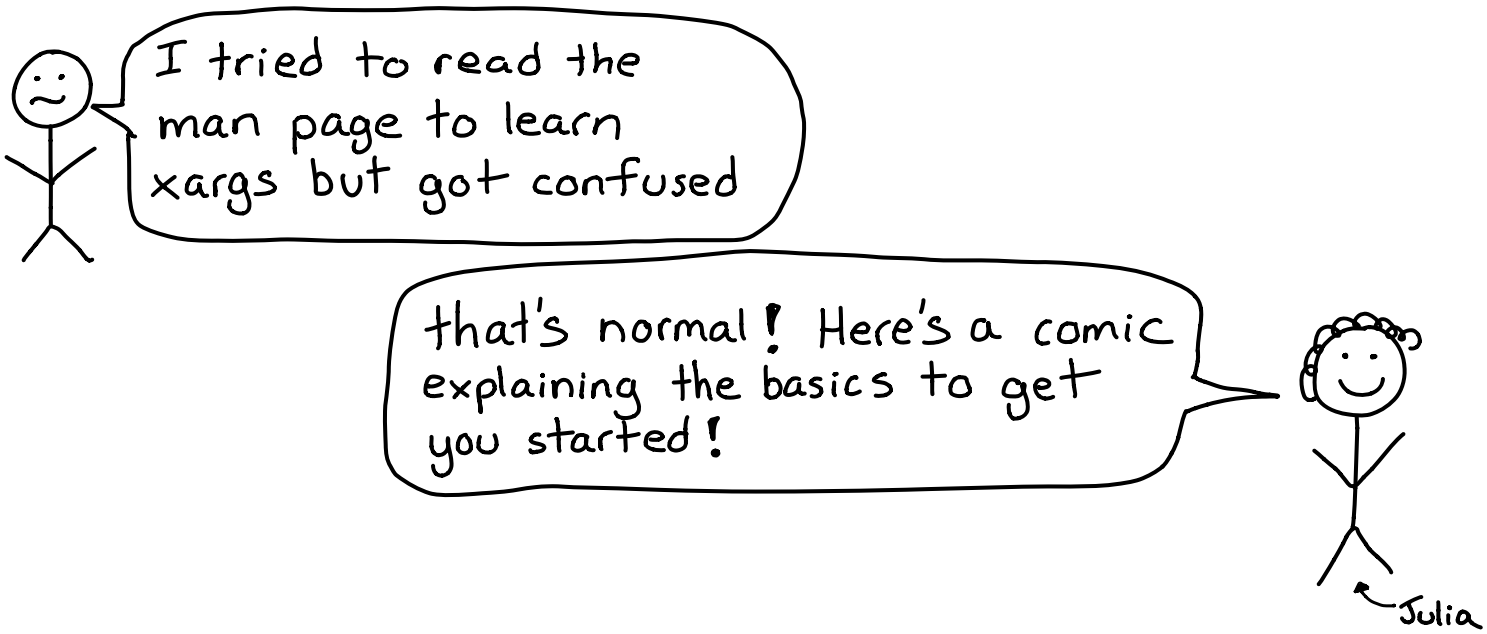
Bite Size Command Line

By Julia Evans

love this?
find more awesome zines at
→ wizardzines.com ←

This zine explains some of the most useful Unix command line tools in 1 page each.



Even if you've used the tool before, I might have a new trick or two for you ♥

more useful tools

- make
- jq
- nohup
- disown
- cut/paste
- sponge
- xxd
- hexdump
- objdump
- strings
- screen
- tmux
- date
- entr
- seq
- join

parallel:

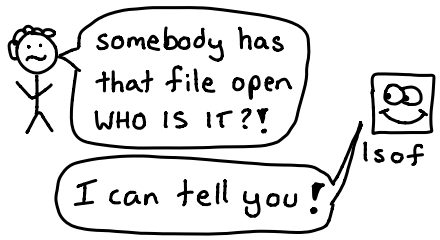
- GNU parallel
- pigz/pixz
- sort --parallel

- diff -U
- vipe
- imagemagick
- fish
- ranger
- chronic

lsof

lsof

stands for list open files



what lsof tells you

for each open file:

- pid
- file type (regular? directory? FIFO? socket?)
- file descriptor (FD column)
- user
- filename / socket address

-p PID

list the files PID has open

lsof /some/dir

list just the open files in /some/dir

-i

list open network sockets (sockets are files!)

examples:

```
-i -n -P   <- -n & -P mean "don't resolve host names/ports" (also -Pni)
-i :8080
-i TCP
-i -s TCP:LISTEN
```

find deleted files

```
$ lsof | grep deleted
```

will show you deleted files!

You can recover open deleted files from

```
/proc/PID/fd/FD
```

process that opened the file

netstat

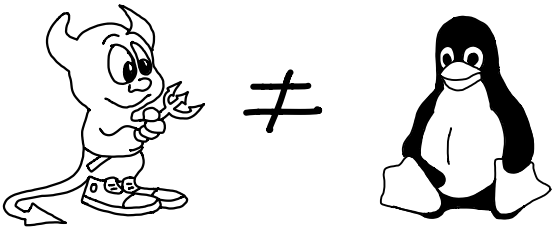
another way to list open sockets on Linux is:

```
netstat -tunapl
```

tuna, please!

On Mac, netstat has different args.

♥ Table of contents ♥



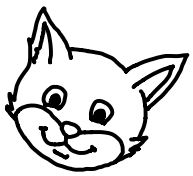
For almost all of these tools, there are at least 2 versions:

① The BSD version (on BSDs & Mac OS)

② The GNU version (on Linux)

All of the examples in this zine were tested on Linux. Some things (like sed -i) are different on Mac. Be careful when writing cross-platform scripts!

You can install the GNU versions on Mac with brew install coreutils.



cat & friends

cat concatenates files

```
$ cat myfile.txt
```

prints contents of myfile.txt

```
$ cat *.txt
```

prints all .txt files put together!

you can use cat as an EXTREMELY BASIC text editor:

① Run `$ cat > file.txt`

② type the contents (don't make mistakes ☺)

③ press ctrl+d to finish

cat -n

prints out the file with line numbers!

```
1 Once upon a midnight..
2 Over many a quaint.
3 While I nodded, nearly
```

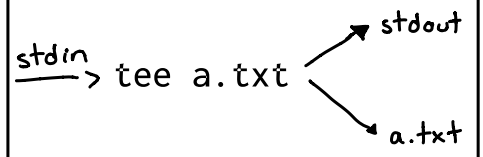
zcat

cats a gzipped file!

Actually just a 1-line shell script that runs gzip -cd, but easier to remember.

tee

'tee file.txt' will write its stdin to both stdout and file.txt



how to redirect to a file owned by root

```
$ sudo echo "hi" >> x.txt
```

this will open x.txt as your user, not as root, so it fails!

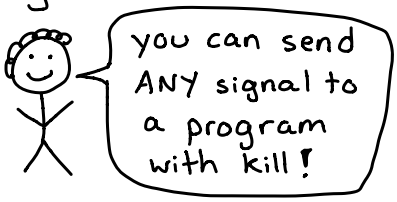
```
$ echo "hi" | sudo tee -a x.txt
```

will open x.txt as root ☺

kill

20

kill doesn't just kill programs




```
$ kill -SIGNAL PID
```

name or number

which signal kill sends

		name	num
kill	=>	SIGTERM	15
kill -9 kill -KILL	=>	SIGKILL	9
kill -HUP	=>	SIGHUP	1
kill -STOP	=>	SIGSTOP	19

kill -l

lists all signals

1 HUP	2 INT	3 QUIT	4 ILL
5 TRAP	6 ABRT	7 BUS	8 FPE
9 KILL	10 USR1	11 SEGV	12 USR2
13 PIPE	14 ALRM	15 TERM	16 STKFLT
17 CHLD	18 CONT	19 STOP	20 TSTP
21 TTIN	22 TTOU	23 URG	24 XCPU
25 XFSZ	26 VTALRM	27 PROF	28 WINCH
29 POLL	30 PWR	31 SYS	

killall -SIGNAL NAME

signals all processes called NAME

for example:

```
$ killall firefox
```

useful flags:

- `-w` wait for all signaled processes to die
- `-i` ask before signalling

pgrep

prints PIDs of matching running programs

pgrep fire matches firefox, firebird

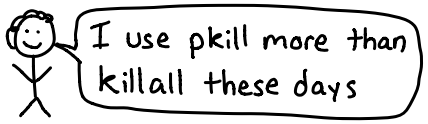
NOT bash firefox.sh

To search the whole command line (eg bash firefox.sh), use `pgrep -f`

pkill

same as pgrep, but signals PIDs found. Example:

```
$ pkill -f firefox
```



grep

5

grep lets you search files for text

```
$ grep cats file.txt
```

Here are some of my favourite grep command line arguments!

- `-i` case insensitive
- `-A` `-B` `-C` Show context for your search. For example:

```
$ grep -A 3 cats
```

will show 3 lines of context after a match

- `-E` aka egrep — Use if you want regexps like ".+" to work. otherwise you need to use ".\+"
- `-V` invert match: find all lines that don't match
- `-l` only show the filenames of the files that matched
- `-F` aka fgrep — don't treat the match string as a regex
- `-r` recursive! Search all the files in a directory.
- `-o` only print the matching part of the line instead of the whole line
- `-a` search binaries: treat binary data like it's text instead of ignoring it!

grep alternatives

ack ag ripgrep

(better for searching code!)

I ♥ ripgrep

find

6

find searches a directory for files

```
$ find /tmp -type d -print
```

- /tmp: directory to search
- -type d: which files
- -print: action to do with the files

-name/-iname

case insensitive

the filename! Example:

```
-name '*.txt'
```

-type TYPE

f: regular file
d: directory
l: symlink
+ more!

-path/-ipath

search the full path!

```
-path '/home/*/*.go'
```

-maxdepth NUM

only descend NUM levels when searching a directory

-size 0

find empty files!
Useful to find files you created by accident

-print0

print null-separated filenames.
Use with `xargs -0`!

locate

The locate command searches a database of every file on your system.

good: faster than find
bad: can get out of date

```
$ sudo updatedb
```

updates the database

-exec COMMAND

action: run COMMAND on every file found

-delete

action: delete all files found

less

19

less is a pager

that means it lets you view (not edit) text files or piped-in text

man uses your pager (usually less) to display man pages

many vim shortcuts work in less

/	search
n/N	next/prev match
j/k	down/up a line
m/'	mark/return to line
g/G	beginning/end of file

(gg in vim)

less -r

displays bash escape codes as colours

try `$ ls --color | less -r`

with -r	without -r
a.txt	a.txt
a.txt.gz (red, bold)	ESC[0m ESC [01;31m a.txt.gz ESC[0m (ugh)

q

quit

v ← lowercase

edit file in your $EDITOR

arrow keys, Home/End, PgUp, PgDn work in less

F

press F to keep reading from the file as it's updated (like tail -f)

press Ctrl+C to stop reading updates

+

+ runs a command when less starts

```
less +F    : follow updates
less +G    : start at end of file
less +20%  : start 20% into file
less +/foo : search for 'foo' right away
```

head & tail

head

shows you the first 10 lines of a file

if you pipe a program's output to head, the program will stop after printing 10 lines (it gets sent SIGPIPE)

tail

tail shows the last 10 lines!

`tail -f FILE` will follow:

print any new lines added to the end of FILE. Super useful for log files!

-n NUM

-n NUM (either head or tail) will change the # lines shown

NUM can also be negative. Example:

```
$ head -n -5 file.txt
```

will print all lines except the last 5

-c NUM

show the first/last NUM bytes of the file

```
$ head -c 1k
```

will show the first 1024 bytes

tail --retry

keep trying to open file if it's inaccesible

tail --pid PID

stop when process PID stops running (with -f)

tail --follow=name

Usually tail -f will follow a file descriptor.

```
tail --follow=name FILENAME
```

will keep following the same filename, even if the file descriptor changes

xargs

xargs takes whitespace separated strings from stdin and converts them into command-line arguments

```
echo /home/tmp | xargs ls
```

will run:

```
ls /home/tmp
```

this is useful when you want to run the same command on a list of files!

- → delete (xargs rm)
- → combine (xargs cat)
- → search (xargs grep)
- → replace (xargs sed)

how to replace "foo" with "bar" in all .txt files:

```
find . -name '*.txt' |
  xargs sed -i 's/foo/bar/g'
```

how to lint every Python file in your Git repo:

```
git ls-files | grep .py |
  xargs pep8
```

if there are spaces in your filenames "my day.txt" xargs will think it's 2 files "my" and "day.txt"

fix it like this:

```
find . -print0 |
   xargs -0 COMMAND
```

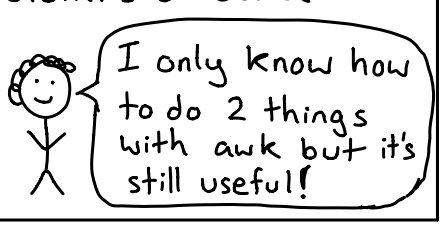
more useful xargs options:

`-n 1` (max-args) makes xargs run a separate process for each input

`-P` (capital P, max-procs) is the max number of parallel processes xargs will start

awk

awk is a tiny programming language for manipulating columns of data



basic awk program structure:

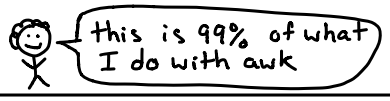
```
BEGIN { ... }
CONDITION { ACTION }
CONDITION { ACTION }
END { ... }
```

do ACTION on lines matching CONDITION

extract a column of text with awk

```
awk -F, '{print $5}'
```

column separator; single quotes!; print the 5th column



SO MANY unix commands print columns of text (ps! ls!)

so being able to get the column you want with awk is GREAT!

awk program example:
sum the numbers in the 3rd column

```
{s += $3};
END {print s}
```

action

at the end, print the sum!

awk program example:
print every line over 80 characters

```
length($0) > 80
```

condition

(there's an implicit {print} as the action)

misc commands ♥

rlwrap

adds history & ctrl support to REPLs that don't already have them

(rl stands for readline)

```
$ rlwrap python
```

watch

rerun a command every 2 seconds

pv

"pipe viewer", gives you stats on data going through a pipe

file

figures out what kind of file (png? pdf?) a file is

cal

a tiny calendar ♡

ts

add a timestamp in front of every input line

ncdu

figure out what's using all your disk space

diff

diff 2 files. Run with -U 8 for context.

comm

find lines 2 sorted files have in common

column

format input into columns

xsel / xclip

copy/paste from system clipboard.
(pbcopy/pbpaste on Mac)

sort & uniq

16

sort sorts its input

```
$ sort names.txt
```

the default sort is alphabetical.

sort -n

numeric sort

'sort' order	'sort -n' order
12	12
15000	48
48	96
6020	6020
96	15000

sort -h: human sort

'sort -n' order	'sort -h' order
15G	45K
30M	30M
45K	15G
200G	200G

useful example:

```
$ du -sh * | sort -h
```

uniq removes duplicates

a
b
b
a
c
c

=>

a
b
a
c

notice there are still 2 'a's! uniq only uniquifies adjacent matching lines

sort + uniq = ♥

Pipe something to 'sort | uniq' and you'll get a deduplicated list of lines! `sort -u` does the same thing.

b
a
b
a

`| sort -u =>`

a
b

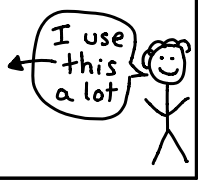
or sort | uniq

uniq -c

counts each line it saw.

Recipe: get the top 10 most common lines in a file:

```
$ sort cats.txt
  | uniq -c
  | sort -n
  | tail -n 10
```



sed

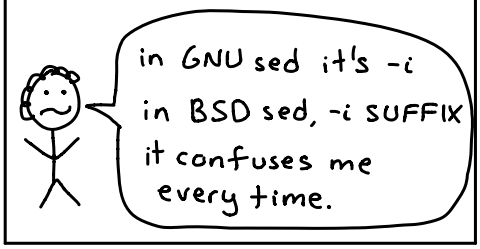
9

sed is most often used for replacing text in a file

```
$ sed s/cat/dog/g file.txt
```

can be a regular expression

change a file in place with -i



some more sed incantations...

sed -n 12 p

print 12th line

-n suppresses output so only what you print with 'p' gets printed

sed 5d

delete 5th line

sed s+cat/+dog/+

can be any character

use + as a regex delimeter

sed G

double space a file (good for long error lines)

sed /cat/d

delete lines matching /cat/

sed '/cat/a dog'

append 'dog' after lines containing 'cat'

sed -n 5,30 p

print lines 5-30

sed -n s/cat/dog/p

only print changed lines

sed 'i17 panda'

insert "panda" on line 17

bash tricks

10

★ ctrl + r ★

search your history!

I use this ♥ constantly ♥ to rerun commands

★ magical braces ★

```
$ convert file.{jpg,png}
```

expands to

```
$ convert file.jpg file.png
```

{1..5} expands to 1 2 3 4 5

!!

expands to the last command run

```
$ sudo !!
```

commands that start with a space don't go in your history. good if there's a 🔒 password 🔒

loops

```
for i in *.png
do
  convert $i $i.jpg
done
```

$( )

gives the output of a command. Example:

```
$ touch file-$(date -I)
```

creates a file named file-2018-05-25

more keyboard shortcuts

ctrl+a	beginning of line
ctrl+e	end of line
ctrl+l	clear the screen

+ lots more emacs shortcuts too!

top

15

top

a live-updating summary of the top users of your system's resources

let's explain some numbers in top!

load average

3 numbers that roughly reflect demand for your CPUs on the system in the last 1, 5, and 15 minutes

if it's higher than the # of CPUs you have, that's often bad!

memory

4 numbers:

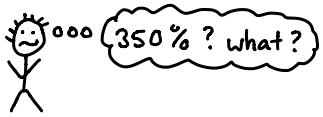
total / free / used / cached

One perhaps unexpected thing: total is not free + used!

total = free + used + cached

filesystem cache

% CPU



this column is given as the % of a single core. If you have 4 cores, this can go up to 400%!

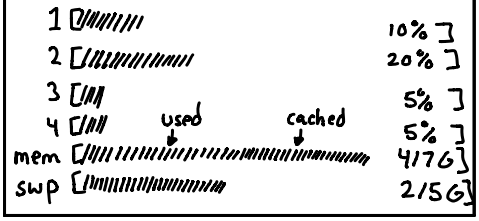
RES

this column is the "resident set size", aka how much RAM your process is using.

SHR is how much of the RES is shared with other processes

htop

a prettier & more interactive version of top ★



ps

14

ps

ps shows which processes are running

I usually run ps like this:

```
$ ps aux
```

u means include username column

a+x together show all process

(ps -ef works too)

w

is for wide. ps auxwww will show all the command line args for each process

e

is for environment. ps auxe will show the environment vars!

wchan

you can choose which columns to show with ps (ps -eo ...)

One cool column is 'wchan', which tells you the name of the kernel function if the process is sleeping.

try it:

```
$ ps -eo user,pid,wchan,cmd
```

★ process state ★

Here's what the letters in ps's STATE column mean:

- R: running
- S/D: asleep
- Z: zombie
- l: multithreaded
- +: in the foreground

f

is for "forest" 🌲. ps auxf will show you an ASCII art process tree!

pstree can display a process tree, too.

ps has 3 different sets of command line arguments 💔

1. UNIX (1 dash)
2. BSD (no dash)
3. GNU (2 dashes)

you can write monstrosities like:

```
$ ps f -f
```

f: forest (BSD)

-f: full format (UNIX)

more bash tricks

11

cd -

changes to the directory you were last in

pushd & popd let you keep a stack

ctrl+z

suspends (SIGTSTP) the running program

bg

starts suspended program & backgrounds it (use after ctrl+z)

fg

brings backgrounded/suspended program to the foreground

♡ shellcheck ♡

shell script linter! helps spot common mistakes.

<( )

process substitution

treat process output like a file (no more temp files!)

Example:

```
$ diff <(ls) <(ls -a)
```

fc

"fix command"

open the last command you ran in an editor

and then run the edited version

type

tells you if something is a builtin, program, or alias

try running:

```
$ type time
$ type ping
$ type pushd
```

(they're all different types!)

disk usage

du

tells you how much disk space files / directories take up

-s summary: total size of all files in a directory

-h human readable sizes

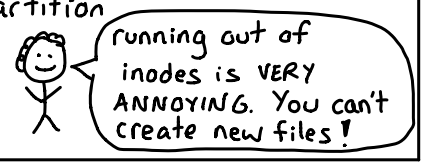
df

tells you how much free space each partition has. -h for human-readable sizes

```
Filesystem     Size  Used Avail Use% Mounted on
/dev/sda3       18G     G  2.5G  86% /
udev           483M  4.0K  483M   1% /dev
tmpfs           99M  1.4M   97M   2% /run
/dev/sda4      167G  157G  9.9G  95% /home
```

df -i

instead of % disk free, report how many inodes are used/free on each partition



ncdu

see what's using disk space in an interactive way

```
17.5 GiB [#####] /music
3.2  GiB [##    ] /photos
5.7  MiB [      ] /code
2.0  MiB [      ] file.pdf
```

iostat

get statistics about disk reads / writes

interval to report at

```
# iostat 5
Device:  kB_read/s  kB_wrtn/s
sda        2190.21     652.87
sdb           6.00       0.00
```

tar

The .tar file format combines many files into one file.

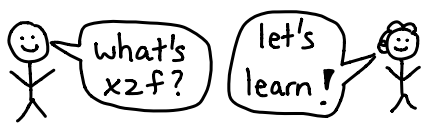
a.txt dir/c.txt b.txt

.tar files aren't compressed by themselves. Usually you gzip them: .tar.gz or .tgz!

Usually when you use the 'tar' command, you'll run some incantation

To unpack a tar.gz, use:

```
tar -xzf file.tar.gz
```



-x is for extract

into the current directory by default (change with -C capital C)

-c is for create

-c lowercase

makes a new tar file!

-t is for list

lists the contents of a tar archive

-f is for file

which tar file to create or unpack

tar can compress / decompress

-z gzip format (.gz)

-j bzip2 format (.bz2)

-J xz format (.xz)

& more! see the man page ☺

putting it together

list contents of a .tar.bz2:

```
$ tar -tjf file.tar.bz2
```

create a .tar.gz

```
$ tar -czf file.tar.gz dir/
```

files to go in the archive