GitHub This repository Search Explore Features Enterprise Blog Sign up Sign in

jlevy / **the-art-of-command-line**

Watch 390 Star 9,446 Fork 521

Master the command line, in one page

86 commits 1 branch 0 releases 18 contributors

branch: master the-art-of-command-line / +

Missed in last commit.

jlevy authored a day ago latest commit 8e14e43ef2

README.md	Missed in last commit.	a day ago
cowsay.png	Add tmux.	11 days ago

Code

Issues 14

Pull requests 20

Pulse

Graphs

HTTPS clone URL

https://github.com/jlevy/

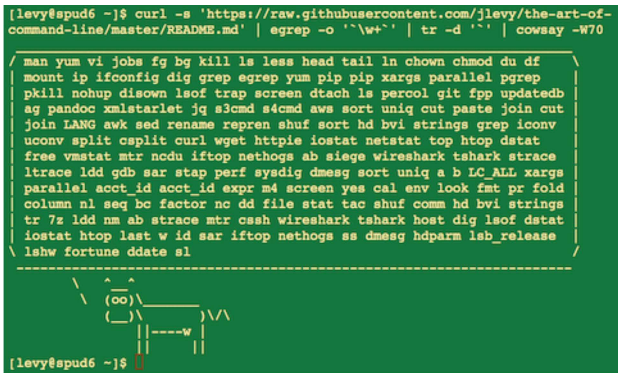
You can clone with HTTPS or Subversion.

Clone in Desktop

Download ZIP

README.md

The Art of Command Line



```
[levy@spud6 ~]$ curl -s 'https://raw.githubusercontent.com/jlevy/the-art-of-
command-line/master/README.md' | egrep -o '`\w+`' | tr -d '`' | cowsay -W70
 _____________________________________________________________________
/ man yum vi jobs fg bg kill ls less head tail ln chown chmod du df    \
| mount ip ifconfig dig grep egrep yum pip pip xargs parallel pgrep   |
| pkill nohup disown lsof trap screen dtach ls percol git fpp updatedb |
| ag pandoc xmlstarlet jq s3cmd s4cmd aws sort uniq cut paste join cut |
| join LANG awk sed rename repren shuf sort hd bvi strings grep iconv  |
| uconv split csplit curl wget httpie iostat netstat top htop dstat    |
| free vmstat mtr ncdu iftop nethogs ab siege wireshark tshark strace  |
| ltrace ldd gdb sar stap perf sysdig dmesg sort uniq a b LC_ALL xargs |
| parallel acct_id acct_id expr m4 screen yes cal env look fmt pr fold |
| column nl seq bc factor nc dd file stat tac shuf comm hd bvi strings |
| tr 7z ldd nm ab strace mtr cssh wireshark tshark host dig lsof dstat |
| iostat htop last w id sar iftop nethogs ss dmesg hdparm lsb_release  |
\ lshw fortune ddate sl                                                /
 ---------------------------------------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
[levy@spud6 ~]$
```

Fluency on the command line is a skill often neglected or considered arcane, but it improves your flexibility and productivity as an engineer in both obvious and subtle ways. This is a selection of notes and tips on using the command-line that I've found useful when working on Linux. Some tips are elementary, and some are fairly specific, sophisticated, or obscure. This page is not long, but if you can use and recall all the items here, you know a lot.

Much of this originally appeared on Quora, but given the interest there, it seems it's worth using Github, where people more talented than I can readily suggest improvements. If you see an error or something that could be better, please submit an issue or PR! (Of course please review the meta section and existing PRs/issues first.)

Meta

Scope:

- This guide is both for beginners and the experienced. The goals are *breadth* (everything important), *specificity* (give concrete examples of the most common case), and *brevity* (avoid things that aren't essential or digressions you can easily look up elsewhere). Every tip is essential in some situation or significantly saves time over alternatives.
- This is written for Linux. Many but not all items apply equally to MacOS (or even Cygwin).
- The focus is on interactive Bash, though many tips apply to other shells and to general Bash scripting.

Notes:

- To keep this to one page, content is implicitly included by reference. You're smart enough to look up more detail elsewhere once you know the idea or command to Google. Use `apt-get` / `yum` / `dnf` / `brew` (as appropriate) to install new programs.
- Use Explainshell to get a helpful breakdown of what commands, options, pipes etc. do.

Basics

- Learn basic Bash. Actually, type `man bash` and at least skim the whole thing; it's pretty easy to follow and not that long. Alternate shells can be nice, but Bash is powerful and always available (learning *only* zsh, fish, etc., while tempting on your own laptop, restricts you in many situations, such as using existing servers).

- Learn at least one text-based editor well. Ideally Vim (`vi`), as there's really no competition for random editing in a terminal (even if you use Emacs, a big IDE, or a modern hipster editor most of the time).

- Learn about redirection of output and input using `>` and `<` and pipes using `|` . Learn about stdout and stderr.

- Learn about file glob expansion with `*` (and perhaps `?` and `{` ... `}`) and quoting and the difference between double `"` and single `'` quotes. (See more on variable expansion below.)

- Be familiar with Bash job management: `&` , **ctrl-z**, **ctrl-c**, `jobs` , `fg` , `bg` , `kill` , etc.

- Know `ssh` , and the basics of passwordless authentication, via `ssh-agent` , `ssh-add` , etc.

- Basic file management: `ls` and `ls -l` (in particular, learn what every column in `ls -l` means), `less` , `head` , `tail` and `tail -f` (or even better, `less +F`), `ln` and `ln -s` (learn the differences and advantages of hard versus soft links), `chown` , `chmod` , `du` (for a quick summary of disk usage: `du -sk *`), `df` , `mount` .

- Basic network management: `ip` or `ifconfig` , `dig` .

- Know regular expressions well, and the various flags to `grep` / `egrep` . The `-i` , `-o` , `-A` , and `-B` options are worth knowing.

- Learn to use `apt-get` , `yum` , or `dnf` (depending on distro) to find and install packages. And make sure you have `pip` to install Python-based command-line tools (a few below are easiest to install via `pip`).

Everyday use

- In Bash, use **Tab** to complete arguments and **ctrl-r** to search through command history.

- In Bash, use **ctrl-w** to delete the last word, and **ctrl-u** to delete the whole line. Use **alt-b** and **alt-f** to move by word, **ctrl-k** to kill to the end of the line, **ctrl-l** to clear the screen. See `man readline` for all the default keybindings in Bash. There are a lot. For example **alt-.** cycles through previous arguments, and **alt-*** expands a glob.

- To go back to the previous working directory: `cd -`

- If you are halfway through typing a command but change your mind, hit **alt-#** to add a `#` at the beginning and enter it as a comment (or use **ctrl-a**, **#**, **enter**). You can then return to it later via command history.

- Use `xargs` (or `parallel`). It's very powerful. Note you can control how many items execute per line (`-L`) as well as parallelism (`-P`). If you're not sure if it'll do the right thing, use `xargs echo` first. Also, `-I{}` is handy. Examples:

```
find . -name '*.py' | xargs grep some_function
cat hosts | xargs -I{} ssh root@{} hostname
```

- `pstree -p` is a helpful display of the process tree.
- Use `pgrep` and `pkill` to find or signal processes by name (`-f` is helpful).
- Know the various signals you can send processes. For example, to suspend a process, use `kill -STOP [pid]` . For the full list, see `man 7 signal`
- Use `nohup` or `disown` if you want a background process to keep running forever.
- Check what processes are listening via `netstat -lntp` .
- See also `lsof` for open sockets and files.
- In Bash scripts, use `set -x` for debugging output. Use strict modes whenever possible. Use `set -e` to abort on errors. Use `set -o pipefail` as well, to be strict about errors (though this topic is a bit subtle). For more involved scripts, also use `trap` .
- In Bash scripts, subshells (written with parentheses) are convenient ways to group commands. A common example is to temporarily move to a different working directory, e.g.

```
# do something in current dir
(cd /some/other/dir && other-command)
# continue in original dir
```

- In Bash, note there are lots of kinds of variable expansion. Checking a variable exists: `${name:?error message}` . For example, if a Bash script requires a single argument, just write `input_file=${1:?usage: $0 input_file}` . Arithmetic expansion: `i=$(( (i + 1) % 5 ))` . Sequences: `{1..10}` . Trimming of strings: `${var%suffix}` and `${var#prefix}` . For example if `var=foo.pdf` , then `echo ${var%.pdf}.txt` prints `foo.txt` .
- The output of a command can be treated like a file via `<(some command)` . For example, compare local `/etc/hosts` with a remote one:

```
diff /etc/hosts <(ssh somehost cat /etc/hosts)
```

- Know about "here documents" in Bash, as in `cat <<EOF ...` .
- In Bash, redirect both standard output and standard error via: `some-command >logfile 2>&1` . Often, to ensure a command does not leave an open file handle to standard input, tying it to the terminal you are in, it is also good practice to add `</dev/null` .
- Use `man ascii` for a good ASCII table, with hex and decimal values. For general encoding info, `man unicode` , `man utf-8` , and `man latin1` are helpful.
- Use `screen` or `tmux` to multiplex the screen, especially useful on remote ssh sessions and to detach and re-attach to a session. A more minimal alternative for session persistence only is `dtach` .
- In ssh, knowing how to port tunnel with `-L` or `-D` (and occasionally `-R`) is useful, e.g. to access web sites from a remote server.
- It can be useful to make a few optimizations to your ssh configuration; for example, this `~/.ssh/config` contains settings to avoid dropped connections in certain network environments, use compression (which is helpful with scp over low-bandwidth connections), and multiplex channels to the same server with a local control file:

```
TCPKeepAlive=yes
ServerAliveInterval=15
ServerAliveCountMax=6
Compression=yes
ControlMaster auto
ControlPath /tmp/%r@%h:%p
ControlPersist yes
```

- A few other options relevant to ssh are security sensitive and should be enabled with care, e.g. per subnet or host or in trusted networks: `StrictHostKeyChecking=no` , `ForwardAgent=yes`
- To get the permissions on a file in octal form, which is useful for system configuration but not available in `ls` and easy to bungle, use something like

```
stat -c '%A %a %n' /etc/timezone
```

- For interactive selection of values from the output of another command, use `percol` .
- For interaction with files based on the output of another command (like `git`), use `fpp` (PathPicker).
- For a simple web server for all files in the current directory (and subdirs), available to anyone on your network, use: `python -m SimpleHTTPServer 7777` (for port 7777 and Python 2).

Processing files and data

- To locate a file by name in the current directory, `find . -iname '*something*'` (or similar). To find a file anywhere by name, use `locate something` (but bear in mind `updatedb` may not have indexed recently created files).
- For general searching through source or data files (more advanced than `grep -r`), use `ag` .
- To convert HTML to text: `lynx -dump -stdin`
- For Markdown, HTML, and all kinds of document conversion, try `pandoc` .
- If you must handle XML, `xmlstarlet` is old but good.
- For JSON, use `jq` .
- For Excel or CSV files, csvkit provides `in2csv` , `csvcut` , `csvjoin` , `csvgrep` , etc.
- For Amazon S3, `s3cmd` is convenient and `s4cmd` is faster. Amazon's `aws` is essential for other AWS-related tasks.
- Know about `sort` and `uniq` , including uniq's `-u` and `-d` options -- see one-liners below.
- Know about `cut` , `paste` , and `join` to manipulate text files. Many people use `cut` but forget about `join` .
- Know that locale affects a lot of command line tools in subtle ways, including sorting order (collation) and performance. Most Linux installations will set `LANG` or other locale variables to a local setting like US English. But be aware sorting will change if you change locale. And know i18n routines can make sort or other commands run *many times* slower. In some situations (such as the set operations or uniqueness operations below) you can safely ignore slow i18n routines entirely and use traditional byte-based sort order, using `export LC_ALL=C` .
- Know basic `awk` and `sed` for simple data munging. For example, summing all numbers in the third column of a text file: `awk '{ x += $3 } END { print x }'` . This is probably 3X faster and 3X shorter than equivalent Python.
- To replace all occurrences of a string in place, in one or more files:

```
perl -pi.bak -e 's/old-string/new-string/g' my-files-*.txt
```

- To rename many files at once according to a pattern, use `rename`. For complex renames, `repren` may help.

```
# Recover backup files foo.bak -> foo:
rename 's/\.bak$//' *.bak
# Full rename of filenames, directories, and contents foo -> bar:
repren --full --preserve-case --from foo --to bar .
```

- Use `shuf` to shuffle or select random lines from a file.
- Know `sort`'s options. Know how keys work (`-t` and `-k`). In particular, watch out that you need to write `-k1,1` to sort by only the first field; `-k1` means sort according to the whole line.
- Stable sort (`sort -s`) can be useful. For example, to sort first by field 2, then secondarily by field 1, you can use `sort -k1,1 | sort -s -k2,2`
- If you ever need to write a tab literal in a command line in Bash (e.g. for the -t argument to sort), press **ctrl-v [Tab]** or write `$'\t'` (the latter is better as you can copy/paste it).
- For binary files, use `hd` for simple hex dumps and `bvi` for binary editing.
- Also for binary files, `strings` (plus `grep`, etc.) lets you find bits of text.
- To convert text encodings, try `iconv`. Or `uconv` for more advanced use; it supports some advanced Unicode things. For example, this command lowercases and removes all accents (by expanding and dropping them):

```
uconv -f utf-8 -t utf-8 -x '::Any-Lower; ::Any-NFD; [:Nonspacing Mark:] >; ::Any-NFC; ' <
```

- To split files into pieces, see `split` (to split by size) and `csplit` (to split by a pattern).
- Use `zless`, `zmore`, `zcat`, and `zgrep` to operate on compressed files.

System debugging

- For web debugging, `curl` and `curl -I` are handy, or their `wget` equivalents, or the more modern `httpie`.
- To know disk/cpu/network status, use `iostat`, `netstat`, `top` (or the better `htop`), and (especially) `dstat`. Good for getting a quick idea of what's happening on a system.
- For a more in-depth system overview, use `glances`. It presents you with several system level statistics in one terminal window. Very helpful for quickly checking on various subsystems.
- To know memory status, run and understand the output of `free` and `vmstat`. In particular, be aware the "cached" value is memory held by the Linux kernel as file cache, so effectively counts toward the "free" value.
- Java system debugging is a different kettle of fish, but a simple trick on Oracle's and some other JVMs is that you can run `kill -3 <pid>` and a full stack trace and heap summary (including generational garbage collection details, which can be highly informative) will be dumped to stderr/logs.
- Use `mtr` as a better traceroute, to identify network issues.
- For looking at why a disk is full, `ncdu` saves time over the usual commands like `du -sh *`.
- To find which socket or process is using bandwidth, try `iftop` or `nethogs`.
- The `ab` tool (comes with Apache) is helpful for quick-and-dirty checking of web server performance. For more complex load testing, try `siege`.
- For more serious network debugging, `wireshark`, `tshark`, or `ngrep`.

- Know about `strace` and `ltrace`. These can be helpful if a program is failing, hanging, or crashing, and you don't know why, or if you want to get a general idea of performance. Note the profiling option (`-c`), and the ability to attach to a running process (`-p`).
- Know about `ldd` to check shared libraries etc.
- Know how to connect to a running process with `gdb` and get its stack traces.
- Use `/proc`. It's amazingly helpful sometimes when debugging live problems. Examples: `/proc/cpuinfo`, `/proc/xxx/cwd`, `/proc/xxx/exe`, `/proc/xxx/fd/`, `/proc/xxx/smaps`.
- When debugging why something went wrong in the past, `sar` can be very helpful. It shows historic statistics on CPU, memory, network, etc.
- For deeper systems and performance analyses, look at `stap` (SystemTap), `perf`, and `sysdig`.
- Confirm what Linux distribution you're using (works on most distros): `lsb_release -a`
- Use `dmesg` whenever something's acting really funny (it could be hardware or driver issues).

One-liners

A few examples of piecing together commands:

- It is remarkably helpful sometimes that you can do set intersection, union, and difference of text files via `sort` / `uniq`. Suppose `a` and `b` are text files that are already uniqued. This is fast, and works on files of arbitrary size, up to many gigabytes. (Sort is not limited by memory, though you may need to use the `-T` option if `/tmp` is on a small root partition.) See also the note about `LC_ALL` above and `sort` 's `-u` option (left out for clarity below).

```
cat a b | sort | uniq > c   # c is a union b
cat a b | sort | uniq -d > c   # c is intersect b
cat a b b | sort | uniq -u > c   # c is set difference a - b
```

- Use `grep . *` to visually examine all contents of all files in a directory, e.g. for directories filled with config settings, like `/sys`, `/proc`, `/etc`.
- Summing all numbers in the third column of a text file (this is probably 3X faster and 3X less code than equivalent Python):

```
awk '{ x += $3 } END { print x }' myfile
```

- If want to see sizes/dates on a tree of files, this is like a recursive `ls -l` but is easier to read than `ls -lR`:

```
find . -type f -ls
```

- Use `xargs` or `parallel` whenever you can. Note you can control how many items execute per line (`-L`) as well as parallelism (`-P`). If you're not sure if it'll do the right thing, use xargs echo first. Also, `-I{}` is handy. Examples:

```
find . -name '*.py' | xargs grep some_function
cat hosts | xargs -I{} ssh root@{} hostname
```

- Say you have a text file, like a web server log, and a certain value that appears on some lines, such as an `acct_id` parameter that is present in the URL. If you want a tally of how many requests for each `acct_id`:

```
cat access.log | egrep -o 'acct_id=[0-9]+' | cut -d= -f2 | sort | uniq -c | sort -rn
```

- Run this function to get a random tip from this document (parses Markdown and extracts an item):

```
function taocl() {
  curl -s https://raw.githubusercontent.com/jlevy/the-art-of-command-line/master/README.m
    pandoc -f markdown -t html |
    xmlstarlet fo --html --dropdtd |
    xmlstarlet sel -t -v "(html/body/ul/li[count(p)>0])[$RANDOM mod last()+1]" |
    xmlstarlet unesc | fmt -80
}
```

Obscure but useful

- `expr` : perform arithmetic or boolean operations or evaluate regular expressions
- `m4` : simple macro processor
- `screen` : powerful terminal multiplexing and session persistence
- `yes` : print a string a lot
- `cal` : nice calendar
- `env` : run a command (useful in scripts)
- `look` : find English words (or lines in a file) beginning with a string
- `cut` and `paste` and `join` : data manipulation
- `fmt` : format text paragraphs
- `pr` : format text into pages/columns
- `fold` : wrap lines of text
- `column` : format text into columns or tables
- `expand` and `unexpand` : convert between tabs and spaces
- `nl` : add line numbers
- `seq` : print numbers
- `bc` : calculator
- `factor` : factor integers
- `gpg` : encrypt and sign files
- `toe` : table of terminfo entries
- `nc` : network debugging and data transfer
- `ngrep` : grep for the network layer
- `dd` : moving data between files or devices
- `file` : identify type of a file
- `stat` : file info
- `tac` : print files in reverse
- `shuf` : random selection of lines from a file
- `comm` : compare sorted files line by line
- `hd` and `bvi` : dump or edit binary files
- `strings` : extract text from binary files
- `tr` : character translation or manipulation

- `iconv` or uconv: conversion for text encodings
- `split` and `csplit` : splitting files
- `7z` : high-ratio file compression
- `ldd` : dynamic library info
- `nm` : symbols from object files
- `ab` : benchmarking web servers
- `strace` : system call debugging
- `mtr` : better traceroute for network debugging
- `cssh` : visual concurrent shell
- `wireshark` and `tshark` : packet capture and network debugging
- `host` and `dig` : DNS lookups
- `lsof` : process file descriptor and socket info
- `dstat` : useful system stats
- `glances` : high level, multi-subsystem overview
- `iostat` : CPU and disk usage stats
- `htop` : improved version of top
- `last` : login history
- `w` : who's logged on
- `id` : user/group identity info
- `sar` : historic system stats
- `iftop` or `nethogs` : network utilization by socket or process
- `ss` : socket statistics
- `dmesg` : boot and system error messages
- `hdparm` : SATA/ATA disk manipulation/performance
- `lsb_release` : Linux distribution info
- `lshw` : hardware information
- `fortune` , `ddate` , and `sl` : um, well, it depends on whether you consider steam locomotives and Zippy quotations "useful"

More resources

- awesome-shell: A curated list of shell tools and resources.
- Strict mode for writing better shell scripts.

Disclaimer

With the exception of very small tasks, code is written so others can read it. With power comes responsibility. The fact you *can* do something in Bash doesn't necessarily mean you should! ;)

© 2015 GitHub, Inc. Terms Privacy Security Contact Status API Training Shop Blog About Help