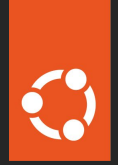
# CLI Design for Designers & Developers

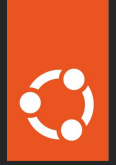
Dr. Hartmut Obendorf
Director Design Cloud & Developer Experience
Canonical

FOSDEM, OSS Design Devroom
February 2nd, 2025



# Design skills for better CLIs

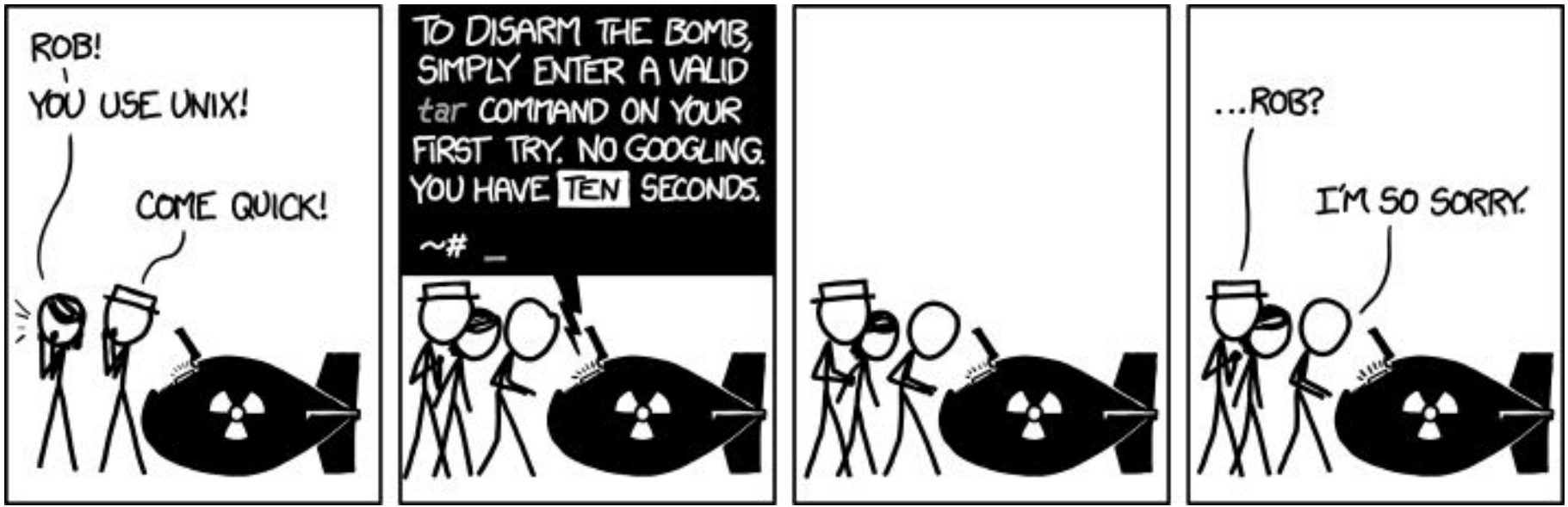
| | |
|---|---|
| Accessibility | (almost built-in) |
| Visual Hierarchy | Spacing<br>Typography |
| Colors | (optional) |
| Iconography | (some, not yours) |
| Copy | Commands, Messages, Help |
| **Information Architecture** | **Conceptual Architecture** |
| **Flows** | **Views, States, Feedback** |
| **Error Handling** | **Recovery, Robustness** |



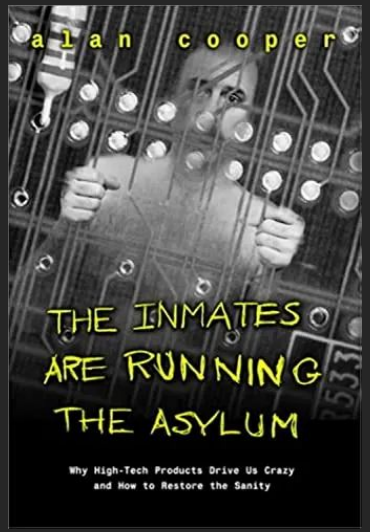
# CLI? Design?

> A **command-line interface** (CLI) is a means of interacting with a computer program by inputting lines of text called command lines. CLIs emerged in the mid-1960s [...] as an **interactive** and more **user-friendly** alternative to the non-interactive mode available with punched cards.

> ‘‘People think it’s this veneer – that the designers are handed this box and told, ‘Make it look good!’ That’s not what we think design is. It’s not just what it looks like and feels like. **Design is how it works**.”


ROB!
YOU USE UNIX!
COME QUICK!
TO DISARM THE BOMB, SIMPLY ENTER A VALID tar COMMAND ON YOUR FIRST TRY. NO GOOGLING. YOU HAVE TEN SECONDS.
~# _
...ROB?
I'M SO SORRY.

# Developers are in need of a design mindset.

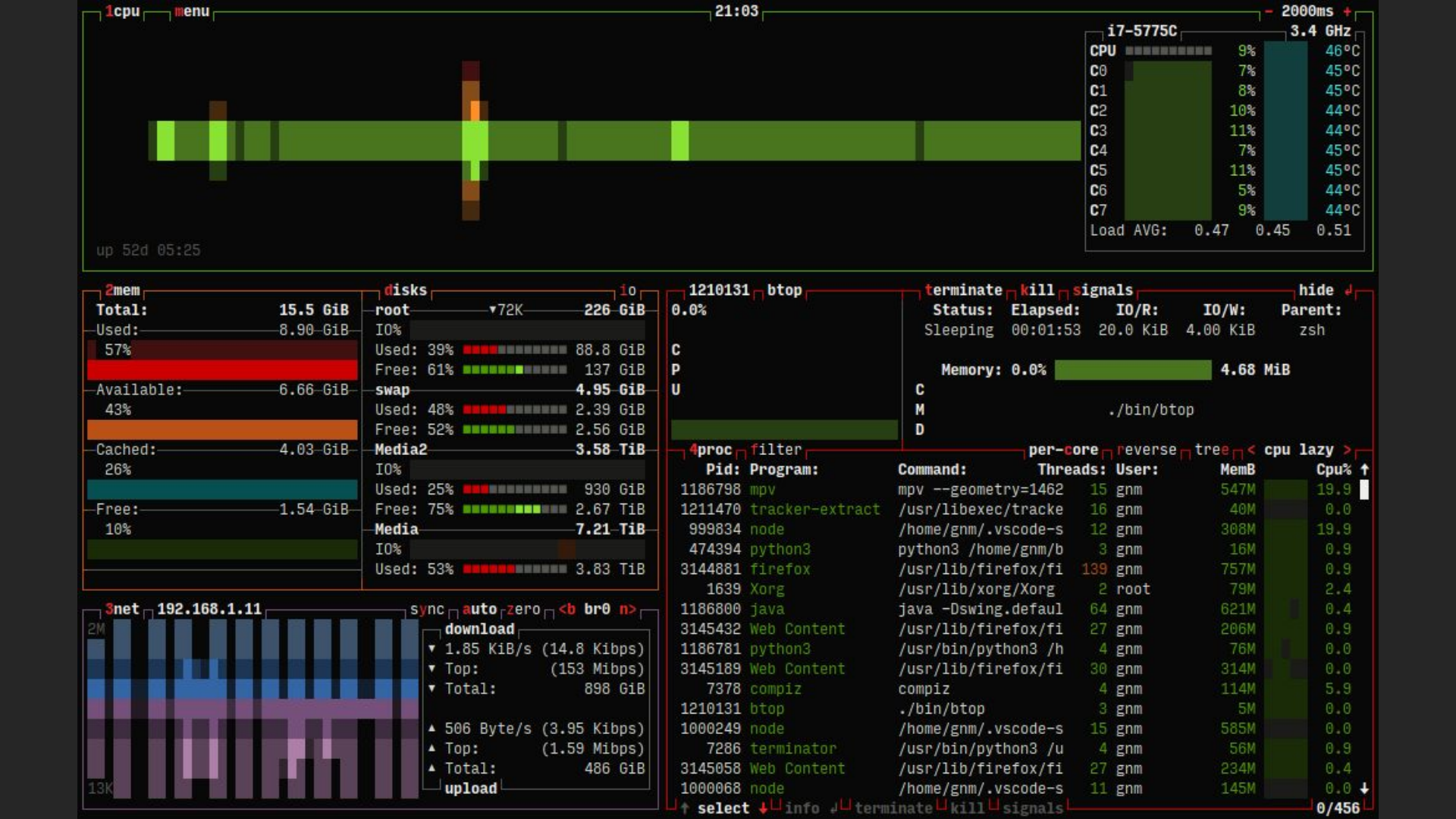



```
tar cvfj php.tar.bz2 /home/mint/php

update-rc.d sshd start 20 2 3 4 5 . stop 20 0 1 6 .

ln -s there here

rm -rf /home/oops\/ /media

mogrify -fill red wedding.jpg
```

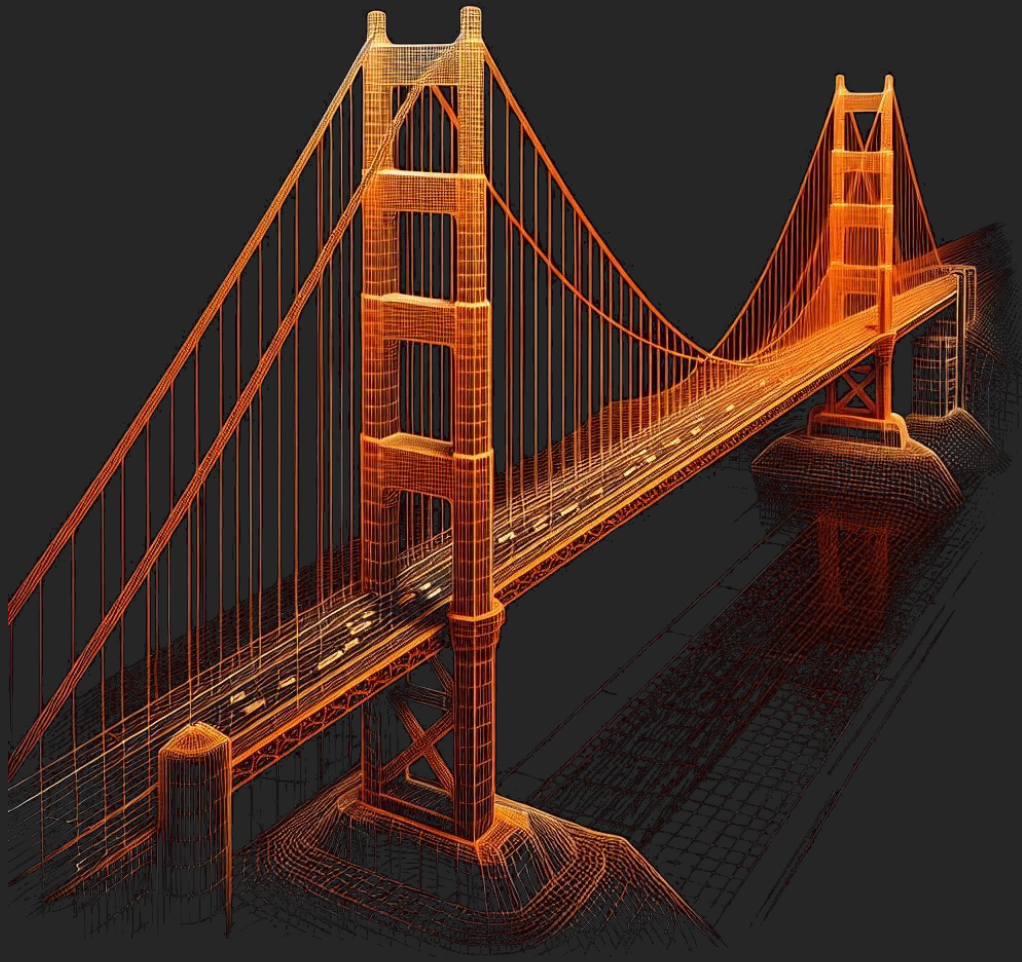
# CLIs are for developers. And by developers?

> I feel intimidated by this. I don't understand it.
> —anonymous Designer

Command Line Interfaces are a world of their own.


1cpu menu
21:03
- 2000ms +
i7-5775C 3.4 GHz
CPU 9% 46°C
C0 7% 45°C
C1 8% 45°C
C2 10% 44°C
C3 11% 44°C
C4 7% 45°C
C5 11% 45°C
C6 5% 44°C
C7 9% 44°C
Load AVG: 0.47 0.45 0.51
up 52d 05:25
2mem
Total: 15.5 GiB
Used: 8.90 GiB
57%
Available: 6.66 GiB
43%
Cached: 4.03 GiB
26%
Free: 1.54 GiB
10%
disks io
root ▾72K 226 GiB
IO%
Used: 39% 88.8 GiB
Free: 61% 137 GiB
swap 4.95 GiB
Used: 48% 2.39 GiB
Free: 52% 2.56 GiB
Media2 3.58 TiB
IO%
Used: 25% 930 GiB
Free: 75% 2.67 TiB
Media 7.21 TiB
IO%
Used: 53% 3.83 TiB
1210131 btop
0.0%
C
P
U
terminate kill signals hide
Status: Elapsed: IO/R: IO/W: Parent:
Sleeping 00:01:53 20.0 KiB 4.00 KiB zsh
Memory: 0.0% 4.68 MiB
C
M ./bin/btop
D
3net 192.168.1.11
sync auto zero <b br0 n>
2M
13K
download
▾ 1.85 KiB/s (14.8 Kibps)
▾ Top: (153 Mibps)
▾ Total: 898 GiB
▴ 506 Byte/s (3.95 Kibps)
▴ Top: (1.59 Mibps)
▴ Total: 486 GiB
upload
4proc filter per-core reverse tree < cpu lazy >
Pid: Program: Command: Threads: User: MemB Cpu% ↑
1186798 mpv mpv --geometry=1462 15 gnm 547M 19.9
1211470 tracker-extract /usr/libexec/tracke 16 gnm 40M 0.0
999834 node /home/gnm/.vscode-s 12 gnm 308M 19.9
474394 python3 python3 /home/gnm/b 3 gnm 16M 0.9
3144881 firefox /usr/lib/firefox/fi 139 gnm 757M 0.9
1639 Xorg /usr/lib/xorg/Xorg 2 root 79M 2.4
1186800 java java -Dswing.defaul 64 gnm 621M 0.4
3145432 Web Content /usr/lib/firefox/fi 27 gnm 206M 0.9
1186781 python3 /usr/bin/python3 /h 4 gnm 76M 0.0
3145189 Web Content /usr/lib/firefox/fi 30 gnm 314M 0.0
7378 compiz compiz 4 gnm 114M 5.9
1210131 btop ./bin/btop 3 gnm 5M 0.0
1000249 node /home/gnm/.vscode-s 15 gnm 585M 0.0
7286 terminator /usr/bin/python3 /u 4 gnm 56M 0.9
3145058 Web Content /usr/lib/firefox/fi 27 gnm 234M 0.4
1000068 node /home/gnm/.vscode-s 11 gnm 145M 0.0 ↓
↑ select ↓ info ↵ terminate kill signals
0/456

# Building bridges



## Developers

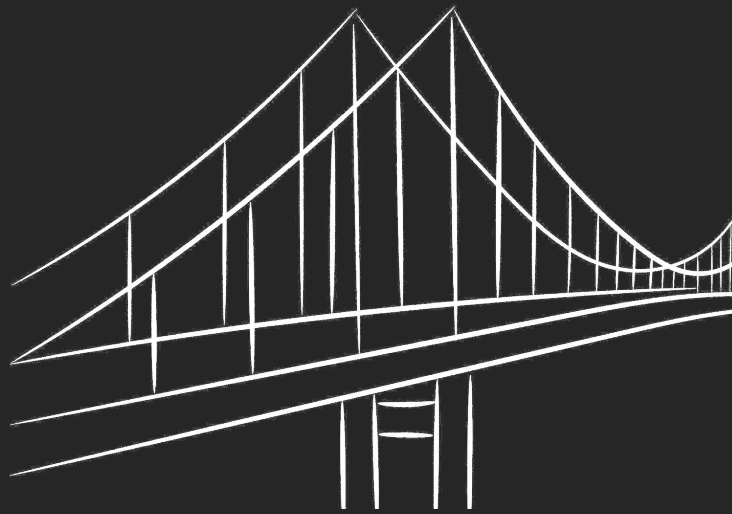
know what is (or might be) possible (most of the time)

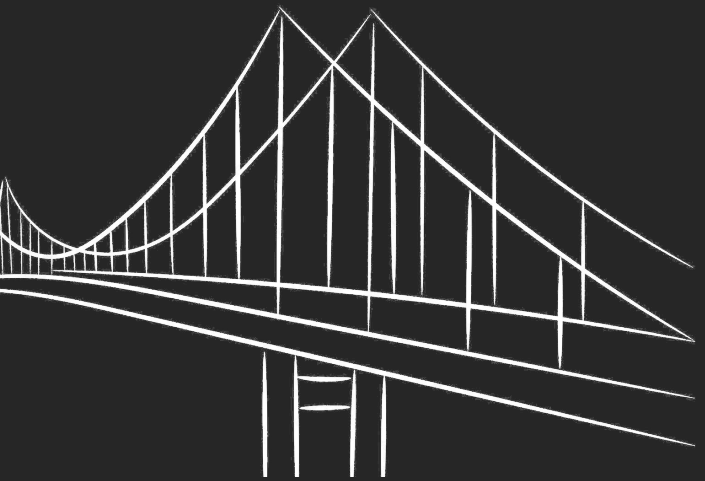
are avid users of CLIs (not necessarily the one in question)

## so they can

share the known limits, and explore new possibilities

bring a CLI mindset, conventions, and a sense of delight





## Designers

bring a curiosity to understand users' needs & goals

bring their skills to provide structure and flow

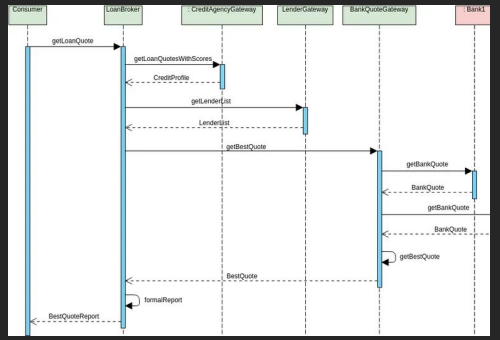
## so they can

listen and observe context to find the value to deliver

(re)define the solution space to meet and establish mental models

# Ingredients for better CLIs

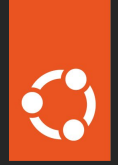
# CLIs are different


No graphical UI.
Can’t do this in Penpot.

Used by experts.
Nice challenge↔do the job

No continuous state.
How do we notify the user?



# Design skills for better CLIs

| | |
|---|---|
| Accessibility | (almost built-in) |
| Visual Hierarchy | Spacing<br>Typography |
| Colors | (optional) |
| Iconography | (some, not yours) |
| Copy | Commands, Messages, Help |
| **Information Architecture** | **Conceptual Architecture** |
| **Flows** | **Views, States, Feedback** |
| **Error Handling** | **Recovery, Robustness** |

# ~~Information~~ Conceptual Architecture

Objects

What is your material?
Identify the primary & secondary objects

Actions

What are the users' goals?
How to operationalize them and harmonize small steps?

Grammar

What form, what logical structure serves your domain best?
Define the shape of the solution space.

Analysis



Deconstruction



Synthesis

# Canonical Grammar

## The standard grammar for Canonical command-line interfaces

At Canonical, most of the commands we build are complex – they manipulate more than one type of object, and often cover more than one aspect of managing the behaviour of the software they interact with. With this standard Canonical command grammar, we strive to create a precise, minimal interface.

**Commands are verbs**. Every command that acts on a primary object of a command (e.g. snaps for snap, virtual machines for multipass) *must* be a verb.

> Choosing the right verb is not trivial: it needs to imply or trigger recall of the object type it refers to. And when a command acts on different object types, it needs to help the user differentiate between these types as they are not explicitly stated in the command (e.g. install or refresh a snap vs. login to a store).

**Commands are logically grouped.** Not all commands are equal. Some commands act on the same type of objects or act in a logically coherent domain. We differentiate between these domains by grouping commands (e.g. build lifecycle vs. store management for snapcraft).

> Ideally, verbs in one command group are implicitly connected with the object they act upon, semantically close to each other, and different from verbs used in other command groups.

**When verbs alone are not sufficient** to distinguish between objects, use the **verb-noun** form (e.g. `set-quota` for snaps).

# Canonical Grammar

```
$ snap --help
Usage:
    snap <command> [<options>...]

    The snap command lets you install, configure, refresh and remove snaps.
    Snaps are packages that work across many different Linux distributions,
    enabling secure delivery and operation of the latest apps and utilities.

Global options:
  -h, --help       Show this help message and exit
  -q, --quiet      Only show warnings and errors, not progress
  -v, --verbose    More verbose output, repeat to increase detail
  -V, --version    Show the application version and exit

Basic commands:
  find             Find packages to install
  install          Install snaps on the system
  refresh          Refresh snaps in the system
  remove           Remove snaps from the system
  info             Show detailed information about snaps
  list             List installed snaps

Other topics:
      Management   components, revert, switch, enable, disable, create-cohort
     Permissions   connections, interface, connect, disconnect
         History   changes, tasks, abort, watch
         Daemons   services, start, stop, restart, logs
   Configuration   get, set, unset, wait
         Aliases   alias, aliases, unalias, prefer
         Account   login, logout, whoami
       Snapshots   saved, save, check-snapshot, restore, forget
          Device   model, remodel, reboot, recovery
          Quotas   set-quota, remove-quota, quotas, quota
```

# Canonical Grammar

```
$ snap --help
Usage:
    snap <command> [<options>...]

    The snap command lets you install, configure, refresh and remove snaps.
    Snaps are packages that work across many different Linux distributions,
    enabling secure delivery and operation of the latest apps and utilities.

Global options:
  -h, --help       Show this help message and exit
  -q, --quiet      Only show warnings and errors, not progress
  -v, --verbose    More verbose output, repeat to increase detail
  -V, --version    Show the application version and exit

Basic commands:
  find             Find packages to install
  install          Install snaps on the system
  refresh          Refresh snaps in the system
  remove           Remove snaps from the system
  info             Show detailed information about snaps
  list             List installed snaps

Other topics:
      Management   components, revert, switch, enable, disable, create-cohort
     Permissions   connections, interface, connect, disconnect
         History   changes, tasks, abort, watch
         Daemons   services, start, stop, restart, logs
   Configuration   get, set, unset, wait
         Aliases   alias, aliases, unalias, prefer
         Account   login, logout, whoami
       Snapshots   saved, save, check-snapshot, restore, forget
          Device   model, remodel, reboot, recovery
          Quotas   set-quota, remove-quota, quotas, quota
```

# Flows: Views, States, Feedback

Views

- What is important? Inverse Attention Distribution (“above the fold”)
- How to provide structure? Tabular Data / Machine-readable Data

States + Feedback

- What is important? Just enough feedback: Signal vs.Noise
- How to keep Users in control? Multiplicity of (Verbosity) Modes

Progress + Notifications

- What is important? Transient, meaningful, precise
- How to make sure to reach Users? Long-running operations & prios

# Canonical Progress

Adding detail to improve progress perception

Buffered progress bar: downloading and extracting can be combined in a single step. This can reduce the overall execution time, but it poses a challenge for visualising progress for two separate processes. If they are not independent, a buffered progress bar can be a good choice.

```
38% Downloading+Extracting... ▰▰▰▰▰▰▰▰▰▰▰▰▱▱▱▱──────────────────────────────

38% Downloading+Extracting... ▰▰▰▰▰▰▰▰▰▰▰▰====──────────────────────────────
```

When a command has several subtasks that are executed in parallel, showing progress for the individual subtasks can help to add a sense of progress even if the overall progress cannot be determined precisely.

Showing subtask progress can be done in a single line of text (individual bars corresponding to individual subtasks: ⠤⠤⠶⣿⣿). This is useful if you expect to have a number of subtasks where it is important to understand that some progress faster than others, or some fail, but detailed progress for each subtask would be overwhelming.

```
⠤⠤  Starting cluster mlflow (8 nodes)  [⠶⣿⣿⠤⠤⣿⣿⣿⣿  ]
```

If details about subtasks contribute meaningfully to the user's understanding of what is happening and how far it has progressed, sub-tasks can also be displayed across several rows.

```
⠤⠤  Starting cluster mlflow (3 nodes)
├─ ml-flow-tracking  Downloading+Extracting…  38% ▰▰▰▰▰▰▰▱▱▱──────────────
├─ ml-flow-projects  Preparing…                0% ──────────────────────────
└─ ml-flow-models    Deploying…               18% ▰▰▰──────────────────────
```

# Error Handling: Recovery, Robustness

| | |
|---|---|
| More Feedback | What did go wrong?<br>Precise messaging. Context.<br>Detailed logs if available. |
| Suggestions | For first-time users and infrequent errors?<br>How can users triangulate?<br>What are potential steps for recovery? |
| Robustness | What could the user mean?<br>When risk is low, consider guessing. Alternatively, start a conversation: did you mean ____? |

# Ubuntu App Recovery

```
$ docker
Command 'docker' not found, but can be installed with:
sudo snap install docker          # version 27.2.0, or
sudo apt  install docker.io       # version 24.0.7-0ubuntu4.1
sudo apt  install podman-docker   # version 4.9.3+ds1-1ubuntu0.2
See 'snap info docker' for additional versions.
```

# Canonical Detail Levels

| Verbosity | Type of Output |
|---|---|
| **quiet** | none, return code only (0=success) |
| **brief** | transient output for steps, preferably only a single line per high-level task |

```
$ snap install melodie
melodie 2.0.0 from Feugas (feugy) installed
```

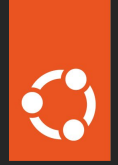
| | |
|---|---|
| **default** | every user-recognizable step should produce a trace, transient progress |

```
$ cargo build
   Compiling crc32fast v1.3.2
   Compiling png v0.16.6
   Compiling rx v0.5.2 (/home/hartmut/workspace/rx)
    Finished dev [unoptimized + debuginfo] target(s) in 2.07s
```

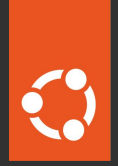
| | |
|---|---|
| **verbose** | all steps and sub-steps should be reflected in output to aid error analysis and recovery |

```
$ apt install autotalent
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed
  autotalent
0 to upgrade, 1 to newly install, 0 to remove and 66 not to upgrade.
Need to get 27,8 kB of archives.
After this operation, 64,5 kB of additional disk space will be used.
Get:1 http://de.archive.ubuntu.com/ubuntu noble/universe amd64 autotalent amd64
0.2-6build1 [27,8 kB]
Fetched 27,8 kB in 0s (153 kB/s)
Selecting previously unselected package autotalent.
(Reading database ... 437021 files and directories currently installed.)
```

# What's in it for you?



**Limitations are good.**
Moonlighting as CLI designer will make you more aware, and could make you a better designer overall.



# We are looking forward to hearing from you.

hartmut.obendorf@canonical.com
opendesign@canonical.com

We help you contribute to OSS as a designer
CLI prototyping: proto* (scan to become a beta-tester)
WIP: CLI guidelines & design system