DIARI DE TERMINAL

Command not found

Diari d'algú que aprén a fer-se entendre pel terminal

$ whoami

command not found

Samuel Soriano

Command not found

Samuel Soriano

Índice

Avís legal

Autor: Samuel Soriano

Primera edició: 2026

Versió: 0.1.0 — compilat el 2026-07-22

ISBN: 9798188579388

Este llibre és un diari personal d'aprenentatge, escrit en primer lloc per a mi mateix. Si t'ha arribat a les mans i t'ajuda o et fa riure una miqueta, m'alegre.

Introducció

Per què escric este diari i per a qui: per a mi mateix, que vaig aprendre informàtica dins d'un terminal, vaig passar trenta anys sense tornar-hi, i necessitava apuntar el camí de tornada.

Este llibre no estava previst. El que estava previst era, com a molt, una carpeta de notes soltes: coses que anava aprenent, comandos que no volia tornar a buscar, frases que la IA em deia i que m'agradava com sonaven. La carpeta es deia `notes`, sense més imaginació, i una part d'ella va morir una vesprada de diumenge d'una manera que ja contaré quan toque. El llibre que tens a les mans és, en bona part, el que vaig reconstruir després — i potser per això té el to que té: no és el quadern polit d'algú que sap on va, és el diari d'algú que va apuntant per no perdre's dos voltes pel mateix lloc.

Convé que sàpies des d'ara qui t'ho conta. No sóc un principiant que va descobrir el terminal l'any passat. Vaig aprendre informàtica *dins* d'un terminal, fa quasi quaranta anys, amb un Amstrad a la

taula del menjador i `DIR` i `COPY CON` com a primeres paraules. Després va vindre tot el que va vindre —les finestres, el ratolí, vint anys donant classes d'informàtica amb entorns gràfics pensats perquè ningú haguera de tocar una línia de comandos— i aquell idioma se'm va anar oblidant com s'obliden els idiomes: sense adonar-te'n, per falta d'ús. El que em va fer tornar no va ser la nostàlgia. Va ser una IA. El meu fill em va dir, un dimarts de gener, que provara una d'eixes eines noves que et contesten dins del terminal mateix, i vaig descobrir que la porta de tornada a la casa vella l'obria, de totes les coses possibles, la tecnologia més nova que existia. Esta ironia em va semblar prou bona com per a anar apuntant-la.

El que no m'esperava —i és la raó de veres per la qual esta introducció existeix, escrita quan tota la resta ja estava acabada— és el que m'ha fet l'escriptura mateixa. Com a professor, ho sabia en teoria: no saps una cosa fins que has d'explicar-la. Doncs bé, m'ha passat amb el meu propi diari. Un exemple, el que més vergonya em fa i per això el pose ací, al davant de tot: durant mesos vaig patir un

problema recurrent que qualsevol que treballe per SSH coneix — se'm tallava la connexió amb el servidor de l'institut i perdia la faena a mitges, scripts a mig córrer, sessions senceres esfumades. I durant eixos mateixos mesos jo tenia escrit, en este mateix llibre, el capítol on la IA m'explica que `tmux` existeix precisament per a això: perquè la sessió sobreviva al tall i t'espere on la vas deixar. L'havia escrit. L'havia entés prou com per a explicar-lo. I vaig continuar perdent faena igualment, perquè apuntar una cosa no és el mateix que saber-la. Va ser repassant estes pàgines, mesos després, quan la lliçó em va caure damunt amb tot el seu pes de ridícul: la resposta portava tot este temps al meu propi diari, esperant que el seu autor el llegira. Ara, quan es talla la connexió, ja no maldic. Escric:

```
$ tmux attach
```

i la sessió està viva, exactament on era. Cada volta que ho faig em recorde que este llibre m'ha ensenyat coses a mi abans que a ningú — que potser és l'única garantia honesta que et puc oferir sobre ell.

Del que trobaràs ací dins, dos avisos. Primer: açò no és un manual. Els capítols van en orde cronològic, amb els meus errors dins, sense retocar-los perquè queden més dignes; cada un porta al final un apunt tècnic amb el que vaig aprendre de veres, per si vols alguna cosa més que riure't de mi — pots llegir-los, saltar-te'ls o anar directament a ells, segons el dia. Segon: els títols dels capítols són comandos i missatges reals de terminal, tal com em van aparéixer en pantalla. Algun et semblarà críptic ara. Cap ho serà quan arribes.

I si tens més de quaranta anys i fa temps que no comences res de zero, este diari és, sense que jo ho sabera quan l'escrivia, per a tu.

$ _

Installe una IA de línia de comandos sense saber què és una línia de comandos: un cursor parpellejant i jo mirant-nos fixament.

Era un dimarts de gener, de vesprada, i portava més de vint anys donant classes d'informàtica sense haver obert un terminal de veres, per gust, des de... pensant-ho bé, des de l'institut. El meu primer ordinador va ser un Amstrad PC1512, i el primer que vaig aprendre no va ser a fer doble clic: va ser `DIR`, `CD` i `COPY CON`. Vaig aprendre informàtica *des del* terminal, molt abans de saber què era una finestra. Això és el que fa més ridícula esta història: no era un principiant complet, era algú que havia oblidat un idioma que parlava de menut.

Perquè després va vindre Windows, i després anys formant professors amb Lliurex i entorns gràfics pensats perquè ningú haguera de tocar una línia de comandos, i el terminal es va convertir en allò que obria dos voltes l'any per a un `ping` o un `ipconfig`, quan algun ordinador de l'aula no volia

connectar-se a res. Havia passat de viure-hi a fer-li visites de cortesia.

Va ser el meu fill qui em va ficar la puça a l'orella. Tenia l'ordinador obert a la taula de la cuina, jo posant-li un plat de sopa al costat, i em va dir, sense alçar la vista de la pantalla: «Pare, prova d'instal·lar açò, és una IA que et va contestant ahí mateix, al terminal». Ho va dir amb la mateixa naturalitat amb què m'hauria dit que provara una app nova de fotos. Per a ell, el terminal no era una cosa estranya ni hostil; era, simplement, un altre lloc on parlar amb la màquina. Per a mi era, de sobte, una habitació de la casa d'infància que no visitava des de feia trenta anys.

Vaig pensar: «Val, açò ho sé fer». Havia format professors durant anys. Sabia què era una carpeta, un fitxer, un permís. Coneixia les paraules. El que no tenia eren els reflexos.

Vaig obrir el terminal i vaig copiar l'ordre d'instal·lació que m'havia passat el meu fill. Vaig prémer Intro. Van eixir línies i línies de text que no vaig llegir —ningú llig eixe text, ni de jove ni ara— i

al final, un missatge que semblava dir que tot havia anat bé.

I després, silenci. Un cursor. Parpellejant.

```
$ _
```

El mateix símbol de sempre. Però ara no esperava un `DIR` ni un `COPY`. Esperava... no sabia què esperava. Ningú m'havia dit què fer amb allò. No hi havia cap botó «Nou document». No hi havia cap menú desplegable amb les opcions ordenades per categories. Hi havia una línia buida i la sensació, molt clara i molt incòmoda, de què la pilota estava en el meu terra i jo no sabia ni els límits del camp.

Vaig escriure «hola». La màquina em va contestar alguna cosa educada, amb aquella cortesia una miqueta excessiva que tenen estes eines quan no saben molt bé qui tenen davant. I jo, formador d'informàtica amb dues dècades d'experiència, vaig sentir exactament el mateix que sent un alumne el primer dia de curs quan li pose un exercici i em mira com si li haguera parlat en xinés.

Va ser humiliant d'una manera molt concreta: no perquè no sabera fer-ho —encara no havia intentat fer res—, sinó perquè no sabia ni què podia

demanar-li. Als huitanta, el terminal et deia les regles amb el seu silenci: hi havia unes poques ordres i prou. Açò era diferent: podia demanar-li qualsevol cosa, en qualsevol paraula, i eixe ventall infinit m'aclaparava més que qualsevol pantalla en negre.

El meu fill, que continuava mirant la seua pantalla, va dir sense mirar-me: «Demana-li que et cree un fitxer de text». Ho vaig fer. Va funcionar. I vaig sentir, per un instant molt breu i molt ridícul per a algú de la meua edat i el meu ofici, la mateixa emoció que vaig sentir la primera volta que un `COPY CON` meu va funcionar, fa trenta anys: no havia fet res d'extraordinari, però ho havia fet jo, i havia funcionat.

Eixa nit no vaig tancar el terminal fins després de mitjanit. No perquè haguera aprés molt —vaig aprendre poquíssim, la veritat—, sinó perquè per primera volta en molt de temps tenia la sensació de ser, altra volta, principiant en alguna cosa que ja havia sigut meua. I resulta que això, a certa edat, és un luxe que no saps que trobes a faltar fins que el tens al davant, parpellejant, esperant que li digues alguna cosa.

```
$ _
```

Encara hi és. Ara ja sap que sóc jo.

Apunt tècnic: instal·lar una eina de terminal, de veres

Deixe un moment el diari per a explicar, sense embuts, què vaig fer eixa nit —perquè si estàs llegint açò per aprendre i no només per riure't de mi, açò és la part que et serveix.

La majoria d'eines de terminal (i les IA de línia de comandos no són una excepció) s'instal·len d'una d'estes dos maneres:

1. Amb el gestor de paquets del sistema, si l'eina ja hi és empaquetada. Cada sistema té el seu:

```
sudo apt install nom-de-l-eina      # Debian, Ubuntu i
  derivats
brew install nom-de-l-eina          # macOS (amb
  Homebrew)
winget install nom-de-l-eina        # Windows
```

2. Amb una comanda d'instal·lació que et dóna el fabricant, sovint amb este patró:

```
curl -fsSL https://exemple.com/install.sh | sh
```

Açò és exactament el que vaig copiar i enganxar eixa nit sense llegir-ho. I ací va l'apunt important: eixe patró (`curl ... | sh`) baixa un script d'internet i l'executa **directament**, sense que el veges abans. És còmode —per això s'ha popularitzat tant— però també és un acte de confiança total en qui t'ha donat eixa comanda. No és el mateix copiar-la d'una web oficial coneguda que d'un missatge d'algú a qui no coneixes.

Si vols mirar abans de saltar (i és un bon hàbit per a agafar des del primer dia), pots baixar l'script sense executar-lo i llegir-lo primer:

```
curl -fsSL https://exemple.com/install.sh -o install.sh
less install.sh
sh install.sh
```

Tres passos en lloc d'un. Trenta segons més. Jo eixa nit no els vaig fer. Ho explique ací perquè tu, si vols, sí que els faces.

Apunt tècnic: quina IA de terminal tries?

El meu fill em va passar una eina concreta eixa nit, però no és l'única, ni de bon tros. Val la pena que sàpies què hi ha abans de triar, perquè no totes juguen amb les mateixes regles.

Les més conegudes són **de pagament o amb capa gratuïta limitada**, i venen d'una empresa concreta que decideix com funcionen i què poden veure: Claude Code (d'Anthropic), Gemini CLI (de Google) o Codex (d'OpenAI) en són exemples. Funcionen bé, són les que més sents nomenar, i és normal començar per ahí —jo mateix vaig començar per una d'estes.

Però n'hi ha una altra família que m'interessa remarcar perquè xoca amb els meus vint anys ensenyant amb programari privatiu a les aules: eines **de codi obert**, com **OpenCode**, que fan bàsicament el mateix —parlar amb una IA des del terminal— però amb el codi a la vista de qui el vullga llegir, sense dependre d'una sola empresa, i sovint amb la llibertat

de connectar-les al model d'IA que tu trigues (fins i tot un que córrega al teu propi ordinador). No és una diferència menor. És la mateixa diferència que hi ha entre llogar una casa i tindre les claus.

No cal triar-ne una per sempre —jo mateix en faig servir més d'una segons el dia—, però sí que val la pena saber, des del primer dia, que l'elecció existeix. Ningú m'ho va dir a mi fins que ja portava setmanes enganxat a la primera que vaig provar.

`echo $SHELL`

Primera immersió: què és un terminal, què és una shell i per què tot té tres noms (m'ho ha hagut d'explicar la IA).

Tres dies després del dia zero, ja em creia una miqueta expert. Havia obert el terminal quatre voltes, havia demanat que em creara dos fitxers més i havia sobreviscut. Eixa vespra volia fer una cosa senzilla: netejar la pantalla, que se m'havia omplit de línies i línies de conversa. Als huitanta ho tenia clar. Ho tenia gravat als dits, no al cap.

Vaig escriure `cls`.

```
$ cls
bash: cls: command not found
```

Ahí estava. El propi títol del llibre, en carn pròpia, la primera setmana. Vaig quedar-me mirant la pantalla amb una indignació completament irracional, com si l'ordinador m'haguera faltat el respecte a mi personalment. Trenta anys escrivint `cls` cada vegada que una pantalla se m'omplia de text, i ara resulta que eixa paraula no volia dir res per a ningú.

Li ho vaig preguntar a la IA, més per orgull ferit que per curiositat real: «per què no funciona `cls`?». I ahí va començar tot un forat de conill que no m'esperava. Resulta que `cls` és de MS-DOS i del CMD de Windows. Ací, en Linux, la comanda equivalent és `clear`. Fins ahí, bé, un canvi de vocabulari, com quan vas a un altre país i has de dir «lleig» en lloc de «pesado». Però llavors vaig preguntar una cosa que em pensava innocent —«i què és exactament, ací, açò on estic escrivint?»— i la resposta va vindre carregada de paraules que jo feia servir com a sinònims sense saber que no ho eren: terminal, shell, consola, prompt, bash.

Quatre paraules per a, en el meu cap de fa trenta anys, una sola cosa: la pantalla negra amb lletres. Vaig sentir eixa vergonya molt particular de descobrir que una cosa que et pensaves simple tenia capes a soles perquè mai t'havies aturat a preguntar-t'ho. És com quan un alumne et pregunta per què un fitxer «.docx» s'anomena així i tu, que portes vint anys ensenyant-ho, has de reconéixer que mai t'ho havies plantejat de veres.

Vaig acabar escrivint `echo $SHELL` només per veure què eixia. La màquina em va contestar `/bin/bash`. Un nom propi. Resulta que jo, cada volta que obria eixa finestra negra, estava convidant a algú concret a la conversa —un programa amb nom i cognoms, `bash`—, i no ho sabia. Trenta anys parlant amb algú sense preguntar-li com es deia.

Apunt tècnic: terminal, shell i consola no són la mateixa cosa

Val la pena aclarir-ho de veres, perquè jo vaig tardar trenta anys a fer-ho i tu no cal que tardes tant.

El terminal (o emulador de terminal) és la finestra: el programa que dibuixa la pantalla negra, gestiona el text, les fonts, les dreceres de teclat. En Linux en pots tindre diversos: GNOME Terminal, Konsole, xterm… En macOS ve amb l'aplicació Terminal.app, encara que molta gent prefereix iTerm2. En Windows modern, Windows Terminal. És, per entendre'ns, el «marc» del quadre.

La shell és el programa que corre *dins* d'eixe terminal i que de veres interpreta les comandes

que escrius. `bash` és la més habitual en Linux, però n'hi ha més: `zsh` (per cert, la shell per defecte en macOS des de 2019, quan Apple va deixar `bash` per raons de llicència), `fish`, o `PowerShell` i `cmd.exe` en Windows. Pots saber quina tens amb:

```
$ echo $SHELL
/bin/bash
```

La consola, històricament, era el terminal físic connectat directament a l'ordinador (una pantalla i un teclat, sense xarxa pel mig). Avui la paraula s'usa sovint com a sinònim informal de «terminal», i en la pràctica del dia a dia ningú et renyirà si els confons. Jo, ara que ho sé, els diferencie —però confesse que als meus apunts de classe, durant vint anys, els he fet servir com si foren la mateixa paraula.

I `cls` contra `clear`? Herència històrica: `cls` ve de MS-DOS i encara viu en `cmd.exe` de Windows. `clear` és la comanda equivalent en la majoria de shells de tipus Unix —bash, zsh, i per tant també a Linux i a macOS, que per davall és un Unix més. Si vens, com jo, d'un passat en DOS, este és probablement el primer «fals amic» que et trobaràs. No serà l'últim.

command not found

El missatge d'error que dóna títol al llibre i la lliçó que amaga: la màquina no et jutja, només no t'entén.

Havia passat una setmana des del dia zero i ja em sentia prou valent per a fer una cosa sol, sense el meu fill al costat mirant-me de reüll per si m'encallava. Volia instal·lar una segona eina, una que havia vist recomanada en un fòrum, per a comparar-la amb la primera. Vaig seguir les instruccions de la seua pàgina web, línia per línia, com qui segueix una recepta en una cuina que no és la seua. `curl` per ací, un parell de línies més, i al final el mateix missatge tranquil·litzador d'abans: instal·lació completada.

Vaig teclejar el nom de l'eina amb l'orgull discret de qui ja no necessita ajuda.

```
$ eina-nova
bash: eina-nova: command not found
```

I ahí va tornar, exactament igual que amb `cls` la setmana anterior, però esta volta amb un matís diferent: no era jo qui recordava una comanda vella, era

una eina *que jo mateix acabava d'instal·lar* i que la màquina em deia que no existia. Vaig sentir una cosa molt concreta i molt absurda: por d'haver «trencat» l'ordinador. Trenta anys treballant amb ordinadors i encara em sobrevé eixe pànic irracional de mestre que li ha tocat alguna cosa a un alumne i pensa que ho ha desconfigurat tot per sempre.

Vaig repetir la instal·lació. Mateix resultat: instal·lació completada, i després `command not found`, com una porta que s'obri i es torna a tancar sense deixar-me passar.

Li ho vaig preguntar a la IA, esta volta sense orgull ferit, directament amb angoixa: «he instal·lat una eina i em diu que no la troba, l'he trencada?». I la resposta em va calmar més del que m'esperava, perquè no parlava de trencar res. Parlava d'un lloc.

Resulta que quan escrius el nom d'una comanda, el terminal no busca per tot l'ordinador com faria jo si buscara unes ulleres perdudes per la casa. Busca només en unes carpetes molt concretes, una llista curta que té memoritzada, i si l'eina no està físicament dins d'eixa llista de carpetes, li és igual que l'hages instal·lat fa un minut o fa un any: per

a ell, no existeix. No m'estava jutjant. No sabia ni que jo hi era. Simplement estava mirant en el lloc equivocat, i jo no li havia dit on mirar.

Eixa frase, «no et jutja, només no t'entén», la vaig escriure eixa mateixa nit en un post-it que encara tinc enganxat al costat del monitor. Perquè no és només una explicació tècnica sobre carpetes: és, ara que ho pense, la mateixa lliçó que done als meus alumnes quan es bloquegen davant d'un exercici i es pensen que l'error és seu, que són «negats per a açò». No ho són. Simplement ningú els ha explicat encara on mira la màquina, o en quin idioma cal parlar-li. `Command not found` no vol dir «ets estúpid». Vol dir «no sé què em demanes, o no sé on buscar-ho». És fred, és sec, i per això mateix és honest: no hi ha cap retret amagat darrere.

Aquella nit vaig entendre que gran part d'esta aventura no anava a ser aprendre comandos de memòria, com als huitanta. Anava a ser aprendre a llegir eixos missatges d'error sense sentir-me jutjat per ells. Una cosa és no saber una cosa. Una altra, molt diferent i molt més paralitzant, és pensar-te que eixe no saber diu alguna cosa de tu.

Apunt tècnic: què és el `PATH` i per què «no ho troba»

El missatge `command not found` quasi sempre amaga una de tres coses, i val la pena saber distingir-les:

1. Una errada d'escriptura. El clàssic. Has escrit `sl` en lloc de `ls`, o has deixat un espai de més. Revisa-ho primer, sempre.

2. L'eina no s'ha instal·lat de veres. La instal·lació ha fallat a mitges sense que t'ho digués prou clar. Es pot comprovar buscant el fitxer directament (per exemple amb `find` o, més senzill, tornant a executar la instal·lació i llegint els missatges esta volta).

3. L'eina existix, però no és al `PATH`. Este és el cas més confús i el que em va passar a mi. El `PATH` és una variable del sistema —una llista de carpetes, separades per `:` en Linux/macOS o per `;` en Windows— on el terminal busca, i **només** ahí, quan li dius el nom d'una comanda sense indicar-li la ruta completa. Pots vore la teua amb:

```
$ echo $PATH
```

```
/usr/local/bin:/usr/bin:/bin:/home/samuel/.local/bin
```

Si l'eina que has instal·lat ha anat a parar a una carpeta que no és cap d'estes, el terminal no la trobarà mai, encara que hi siga. Per a saber on ha anat a parar una comanda concreta (quan sí que funciona), es fa servir:

```
$ which eina-nova
/home/samuel/.local/bin/eina-nova
```

I si `which` tampoc la troba, és que efectivament no és enlloc del `PATH`. La solució sol ser una d'estes dos: moure l'executable a una carpeta que ja hi siga (com `/usr/local/bin`), o afegir la carpeta on és a la llista del `PATH`, normalment editant l'arxiu de configuració de la teua shell (`~/.bashrc` o `~/.zshrc` en Linux/macOS; les variables d'entorn del sistema en Windows). Jo eixa nit vaig fer la primera opció, perquè em va semblar menys arriscada. No m'equivocava: quan no saps què fas, com menys toques, millor.

pwd

Perdut pel sistema d'arxius sense finestres ni icones; tres lletres com a mantra per a saber on estic.

Volia fer una cosa ben senzilla: que la IA em resumira un PDF que m'havia baixat eixa mateixa vesprada des del navegador. A l'escriptori, açò és un gest de mig segon: obric la carpeta de baixades, veig la icona, la faig clic i prou. El fitxer «està» on el veig, i jo mai m'he hagut de preguntar res més sobre ell.

Vaig seguir un tutorial en anglés que deia, ben clar: `cd Downloads`. Ho vaig copiar sense pensar-m'ho.

```
$ cd Downloads
bash: cd: Downloads: No such file or directory
```

Vaig tornar-ho a intentar, per si l'havia escrit malament. Mateix resultat. La carpeta hi era, jo l'havia vista fa cinc minuts al navegador, amb el PDF dins ben visible. Com podia no existir?

Vaig preguntar-li a la IA, i la resposta em va fer riure i emprenyar-me alhora: «Quin idioma tens configurat en l'ordinador?». Valencià. «Doncs pro-

bablement la carpeta no s'anomena *Downloads*, sinó *Baixades*». I efectivament, ahí era. El meu propi ordinador, configurat en la meua pròpia llengua per orgull i per coherència, m'havia parat una trampa que cap tutorial en anglés avisava.

```
$ cd Baixades
$
```

Silenci. Cap error. Però tampoc cap confirmació. I ahí va vindre el problema de veres: no tenia ni idea d'on estava. A l'escriptori, quan obres una carpeta, la finestra t'ho diu tot el temps, amb lletra grossa, a dalt: estàs ací. Al terminal, si no li ho demanes, no et diu res. Vaig fer `cd` un parell de voltes més, provant carpetes a l'atzar, com qui camina a les palpentes per una casa amb els ulls tapats —una casa que, a més, coneixia de tota la vida, però ara sense poder-la vore.

Va ser eixa desorientació física, més que qualsevol error de sintaxi, el que em va fer sentir realment perdut per primera volta en tota esta aventura. No era un missatge d'error concret que poguera arreglar amb una comanda. Era no saber, en cap

moment, on era jo mateix dins del meu propi ordinador.

Li ho vaig confessar a la IA, quasi avergonyit: «no sé on estic». I la resposta va ser una sola paraula, tres lletres:

```
$ pwd
/home/samuel/Baixades
```

`pwd`. *Print working directory.* Imprimeix on estàs ara mateix. Vaig quedar-me mirant eixa línia una bona estona. `/home/samuel/Baixades`. Tres paraules separades per barres que em deien, amb tota la precisió del món, exactament on era. Ni «per ahí», ni «a la carpeta de sempre»: una adreça exacta, des de l'arrel de tot el sistema fins a mi.

Vaig començar a escriure `pwd` cada vegada que dubtava, quasi com un tic. Abans de fer res, `pwd`. Després de moure'm, `pwd`. No perquè em calguera sempre, sinó perquè eixe gest —preguntar-me on sóc abans de decidir què faig— em donava una calma que no sabia que necessitava. Als huitanta mai vaig necessitar-ho, perquè aleshores tot era una sola carpeta, un sol nivell, sense laberint. Ara el laberint

era real, i tenir tres lletres per a aturar-me i mirar al voltant valia més que qualsevol mapa.

Apunt tècnic: moure's pel sistema d'arxius sense vore'l

Quan no tens finestres ni icones, necessites tres comandos bàsics per a no perdre't. Els tres més importants són estos:

`pwd` (*print working directory*) et diu on ets ara mateix, amb la ruta completa des de l'arrel:

```
$ pwd
/home/samuel/Baixades
```

`ls` (*list*) et mostra què hi ha dins d'eixa carpeta —l'equivalent a obrir la finestra i mirar:

```
$ ls
factura.pdf  informe.pdf  notes.txt
```

`cd` (*change directory*) et mou a una altra carpeta. Ací és on val la pena distingir dos tipus de ruta:

- **Ruta absoluta**: comença per `/` i descriu el camí sencer des de l'arrel del sistema. Funciona des de qualsevol lloc: `cd /home/samuel/Baixades`.

- **Ruta relativa**: descriu el camí des d'on ja estàs. `cd Baixades` només funciona si ja estàs al lloc just per damunt.

Tres dreceres que val la pena memoritzar de seguida:

```
cd ~      # torna directament a la teua carpeta d'usuari
  (la "casa")
cd ..     # puja un nivell, a la carpeta pare
cd -      # torna a la carpeta on eres just abans
```

I un detall que a mi em va enganyar: si el sistema operatiu no està en anglés, els noms de les carpetes habituals també canvien —*Downloads* pot ser *Baixades*, *Descargas*, *Téléchargements*, segons l'idioma. Cap tutorial genèric ho avisa mai. Si un `cd` et diu que la carpeta no existeix i estàs segur que hi és, comprova primer amb `ls` com s'anomena de veres, en lloc d'assumir que la saps.

En Windows, l'equivalent clàssic seria `cd` (que funciona igual) i `dir` en lloc de `ls`. Però si obris PowerShell, la sorpresa és agradable: `pwd` i `ls` també hi funcionen, perquè són àlies dels seus comandos natius (`Get-Location` i `Get-ChildItem`). Un pontet

inesperat entre mons que em pensava separats del tot.

`chsh`

El dia que vaig descobrir que hi ha més d'una shell (bash, zsh, fish...) i vaig entrar en pànic de tria.

Va ser en una reunió de claustre, de totes les coses. Un company més jove, del departament de FP, va obrir el seu portàtil per a compartir pantalla i, mentre buscava l'arxiu, va passar mig segon pel terminal. Un mig segon va ser prou. El seu terminal tenia colors —de veres colors, no el verd fosforescent que jo recordava d'algun western informàtic—, un indicador amb el nom de la carpeta i una icona de git, i quan va començar a escriure una comanda, l'ordinador li suggeria, en gris clar, com acabar-la, com si li llegira el pensament.

El meu, en comparació, era una pantalla negra amb lletres blanques i un `$` pelat. Funcional. Honest. I, de sobte, vint anys més vell.

«Això és fish?», li vaig preguntar, com si jo supiera de què parlava. «Zsh, amb un tema», va contestar ell, sense alçar la vista, com si fóra la cosa més òbvia del món. Zsh. Jo, després del capítol de `cls`,

ja sabia que existia —la IA me n'havia parlat de passada—, però mai m'havia parat a pensar que jo, personalment, poguera *triar* quina shell fer servir. Sempre havia donat per fet que el terminal era com la llengua materna: no la tries, la tens.

Eixa nit vaig obrir el meu terminal amb una missió: canviar de shell. I ahí va començar la crisi de veres, perquè quan li vaig preguntar a la IA «quina shell dec fer servir», la resposta no va ser una recomanació neta, sinó una llista: `bash`, la de sempre, sòbria i present a tot arreu; `zsh`, amb un ecosistema enorme de personalització (l'`oh-my-zsh` que el meu company tenia muntat); `fish`, encara més amigable de fàbrica però amb una sintaxi una miqueta diferent de la resta; i, més avall, coses amb noms com `nushell` que ni vaig gosar obrir l'enllaç.

Em vaig quedar paralitzat. No era un pànic tècnic, era un pànic de decisió pura, el mateix que em passa al supermercat davant de vint marques de iogurt diferents quan només vull un iogurt. I per damunt, una vergonya afegida: acabava de dominar —«dominar» és molt dir, acabava de sobreviure a— quatre comandos de bash, i ara em deien que tot allò

podia ser diferent en un altre lloc, que potser havia aprés «l'idioma equivocat» sense saber-ho.

Li ho vaig confessar a la IA amb estes paraules exactes: «tinc por de canviar i que tot el que he aprés no valga per a res». I la resposta, esta volta, em va llevar un pes de damunt: `cd`, `ls`, `pwd`, el `PATH`, els permisos, tots els conceptes de les últimes setmanes eren *quasi* idèntics en bash, zsh i fish. El que canvia és la superfície —els colors, l'autocompletat, algun detall de sintaxi en scripts més avançats—, no els fonaments. No havia aprés «l'idioma equivocat». Havia aprés la gramàtica de base, i les shells eren, més que idiomes diferents, accents diferents del mateix idioma.

Amb això clar, vaig decidir provar zsh, sense grans cerimònies:

```
$ chsh -s $(which zsh)
Password:
```

Em va demanar la contrasenya —un moment de tensió absurda, com si estiguera a punt de signar un contracte important— i després, res. Cap confirmació espectacular. Vaig tancar la finestra del terminal, la vaig tornar a obrir, esperant-me una trans-

formació immediata... i vaig trobar exactament el mateix $ d'abans, sense colors ni suggeriments. Vaig pensar que ho havia fet malament un altra volta.

No l'havia fet malament. Simplement havia canviat de shell, però una zsh «nua», sense cap tema ni configuració, s'assembla molt a bash. Els colors i l'autocompletat del meu company no eren de zsh en si mateix, sinó de tota la decoració que ell hi havia afegit per damunt, any rere any. Vaig entendre, aleshores, que la crisi de tria no calia haver-la patida tan fort: no triava una casa nova per a sempre, triava una paret que després podia pintar del color que volguera. I la paret de bash, la vella, tampoc calia enderrocar-la —ho podia haver decorat igual sense mudar-me enlloc.

Apunt tècnic: canviar de shell (i quan no cal fer-ho)

Saber quines shells tens instal·lades:

```
$ cat /etc/shells
/bin/bash
/bin/zsh
```

```
/usr/bin/fish
```

Canviar la shell per defecte (Linux i macOS), amb la ruta completa d'una de la llista anterior:

```
$ chsh -s /bin/zsh
```

Cal tancar sessió o obrir un terminal nou perquè es note el canvi —a mi també em va confondre que no passara res «en calent». En macOS, per cert, zsh ja ve com a shell per defecte des de 2019, així que si hi treballes probablement no cal ni tocar res.

Saber en quina shell estàs ara mateix, si dubtes:

```
$ echo $SHELL
/bin/zsh
```

I l'apunt important, per a estalviar-te la meua crisi: **canviar de shell no esborra el que ja saps**. `cd`, `ls`, `pwd`, el `PATH`, els permisos, els conceptes bàsics són pràcticament idèntics en bash i zsh (fish s'allunya un poc més en la sintaxi de scripts, però no en l'ús diari). El que sol enamorar la gent de zsh o fish no és la shell en si, sinó la **decoració**: temes com `oh-my-zsh`, suggeriments de comandos, colors

segons l'estat del git. Tot això es pot afegir també —amb un poc menys de comoditat— sobre bash. No cal mudar-se de casa per a pintar les parets.

En Windows, l'equivalent no és `chsh` sinó triar la shell per defecte des de la configuració de Windows Terminal (pots tindre PowerShell, `cmd.exe` o, si tens WSL instal·lat, una bash de veres corrent per davall de Windows).

rm -rf

El primer error irreversible (o quasi): la revelació que el terminal no té paperera.

Des del dia zero havia anat guardant notes soltes en una carpeta que vaig anomenar, amb poca imaginació, `notes`. Res formal: fragments del que anava aprenent, frases que la IA em deia i m'agradaven, fins i tot algun esborrany de com contar tot açò —sense saber-ho encara, estava escrivint els primers apunts d'este mateix llibre. Dins d'eixa carpeta en tenia una altra, `descartes`, amb proves fallides, arxius de test, coses que sabia que no valien per a res.

Un diumenge de vesprada, amb ganes d'ordre, vaig decidir netejar `descartes`. Vaig teclejar `cd descartes`, o això em pensava. En realitat —ho vaig descobrir després, massa tard— eixe `cd` no havia funcionat perquè havia escrit el nom malament, i la comanda havia fallat en silenci amb un error que no vaig llegir, ocupat com estava pensant en el diumenge que tenia per davant. Vaig seguir escrivint com si haguera entrat.

```
$ rm -rf *
```

Sense preguntar-me `pwd` primer. El mantra de tres lletres que m'havia salvat al capítol quart, aquell que havia promés no oblidar, i el vaig oblidar igualment, perquè ja em creia prou expert per a saltar-me'l. Vaig prémer Intro amb la mateixa tranquil·litat amb què repengaves un cendrer a la brossa.

Vaig escriure `ls` just després, per confirmar que `descartes` havia quedat buida.

```
$ ls
$
```

Buida. Completament buida. Però no era `descartes`. Era `notes`. Totes les meues notes de sis setmanes, els fragments, els esborranys, tot allò —fet, en un segon, no-res.

El cor se m'ha girat menys de voltes en situacions més greus de la vida real. Vaig fer el gest instintiu de qui porta trenta anys amb ordinadors: buscar la paperera, obrir-la, arrossegar els arxius cap arrere. No hi havia cap paperera. `rm` no mou res a cap lloc on esperar penedit. `rm` esborra, i prou. Vaig preguntar-

li a la IA, amb els dits tremolant una miqueta, si hi havia alguna manera de recuperar-ho.

La resposta va ser honesta, i per això mateix dura d'escoltar: potser, amb eines especialitzades de recuperació de disc, i sense cap garantia, i com més tard ho intentara i com més escriguera a eixe disc mentrestant, pitjor. No vaig tindre paciència ni esperança per a intentar-ho. Vaig acceptar la pèrdua.

El que em va quedar, més que la ràbia, va ser una revelació incòmoda sobre com havia entés el terminal fins aleshores: com un espai més segur del que és de veres, perquè m'havia acostumat que sempre hi havia una xarxa de seguretat davall — desfer, paperera, «estàs segur?». El terminal no té eixa xarxa per defecte. Fa exactament el que li dius, amb una obediència tan perfecta que resulta perillosa. No et pregunta si n'estàs segur. Assumeix que ja ho estaves quan ho vas escriure.

No vaig tornar a escriure `rm -rf` sense mirar tres voltes on estava. I vaig aprendre, també, que allò que de veres importa no hauria d'existir en un sol lloc mai més. Vaig tornar a escriure algunes de les notes, de memòria, pitjor que l'original, però vives.

Les altres, les que no vaig recordar, encara les trobe a faltar de tant en tant, com qui troba a faltar una conversa de la qual no va prendre nota.

Apunt tècnic: `rm` no té marxa arrere (i com protegir-te'n)

Què fan de veres els flags de `rm -rf`:

- `rm` esborra arxius, directament i per sempre.
- `-r` (*recursive*) permet esborrar una carpeta i tot el que hi ha dins, a qualsevol profunditat.
- `-f` (*force*) evita que et pregunte res, ni tan sols si l'arxiu és protegit contra escriptura. Silenci total.

La combinació `rm -rf` és, per tant, «esborra açò sencer, sense preguntar res». Extremadament útil quan saps exactament què fas. Extremadament perillosa quan no.

El parany dels comodins. `rm -rf *` esborra tot el que hi ha en la carpeta actual —per això és tan important saber `pwd` abans d'executar-ho (capítol 4). Encara pitjor és este patró, clàssic en els fòrums d'horror informàtic:

```
rm -rf "$CARPETA"/*
```

Si la variable `$CARPETA` no s'ha definit bé i queda buida, la comanda es converteix en `rm -rf /*` —esborrar-ho literalment tot des de l'arrel del sistema. Ha passat de veres, a gent amb més experiència que jo.

Tres hàbits que hauria d'haver tingut des del primer dia:

1. **Mira abans de disparar.** Executa `ls` (o, encara millor, `echo rm -rf *` per a vore què s'expandiria) abans del `rm` de veres.
2. **Fes que `rm` et pregunte.** Es pot configurar un àlies perquè demane confirmació sempre:

   ```
   alias rm='rm -i'
   ```

   (Afig-ho a `~/.bashrc` o `~/.zshrc` per a que quede fix.)
3. **Usa una paperera de veres per al dia a dia.** Eines com `trash-cli` (Linux) o la comanda `trash` (macOS, instal·lable amb Homebrew) mouen els arxius a una paperera real en lloc d'esborrar-los directament —recuperen la xarxa de seguretat que jo trobava a faltar.

I un últim apunt que no és tècnic, sinó de sentit comú: si alguna cosa importa de veres, no hauria de viure només en un lloc. Un repositori git, una còpia al núvol, el que siga. El terminal no perdona; una còpia, sí.

En Windows, per cert, l'equivalent per línia de comandos (`del` en `cmd.exe`, `Remove-Item` en PowerShell) tampoc passa per la paperera de reciclatge —eixa protecció és només de l'explorador d'arxius gràfic. La sensació de seguretat que dóna Windows és, en este sentit, un miratge igual de fràgil que el meu.

history

Un capítol curt. El silenci de després.

No vaig obrir el terminal durant tres dies.

No és que li tinguera por —o sí, potser una miqueta—, però sobretot no tenia res per a dir-li. `notes` ja no hi era, i amb ella, sis setmanes de fragments que no tornarien. La IA m'ho havia explicat amb tota l'honestedat del món: podia intentar-se una recuperació, sense garanties, i com més esperara i més escriguera al disc, pitjor. No vaig tindre ni forces ni fe per a intentar-ho.

El tercer dia, per fi, vaig tornar a obrir-lo. No per a escriure res nou. Només per curiositat trista, vaig teclejar una comanda que no havia usat mai fins aleshores:

```
$ history | tail -20
```

I ahí van aparéixer, un darrere l'altre, els últims comandos d'eixa vesprada. `cd descartes`. L'error que no vaig llegir. `rm -rf *`. `ls`. El moment exacte, escrit en fred, sense drama, sense els batecs que

jo recordava. El terminal no guarda el que perds. Només guarda el que li demanes.

Vaig mirar eixa línia una bona estona. `rm -rf *`. Tan curta. Tan definitiva. Cap indici, en eixe text pla, de tot el que s'havia esfumat darrere.

No vaig esborrar l'`history`. El vaig deixar ahí, com qui deixa una cicatriu a la vista en lloc de tapar-la amb una mànega llarga. Vaig tancar el terminal, i eixa nit, per primera volta en tres dies, vaig tornar a obrir un document en blanc.

No per a recuperar el que havia perdut. Per a començar de nou.

Permission denied

Permisos, `sudo` i el respecte que fa ser root; per què la IA em frenava quan jo volia córrer.

Després del desastre de `notes`, vaig passar unes quantes setmanes fent-ho tot amb la prudència exagerada de qui ha tocat el foc una volta. Comprovava, remirava, preguntava dos voltes. Fins que un company de l'institut em va passar un script per automatitzar una tasca avorrida —generar les actes d'una reunió a partir d'una plantilla— i vaig sentir que ja era hora de tornar a córrer una miqueta.

```
$ ./generar_actes.sh
bash: ./generar_actes.sh: Permission denied
```

I ací, sense pensar-m'ho ni dos segons, vaig fer una cosa que em fa vergonya confessar: vaig recordar, de no sé on, que «quan un ordinador et diu que no, li poses `sudo` davant i ja està». Ho havia sentit dir, de passada, en algun vídeo, com qui recorda una frase feta sense entendre'n l'origen. Vaig escriure `sudo ./generar_actes.sh` a la línia, però abans de prémer Intro, per pura rutina de les últimes setma-

nes, li ho vaig ensenyar a la IA: «açò no acaba de funcionar, li pose `sudo` i prou?».

La resposta no va ser un sí ràpid. Va ser una pregunta: «Saps per què et diu que no, abans de forçar-ho?». No ho sabia. I la IA no em va deixar seguir fins que no ho miràrem junts.

Em va frenar. Literalment, em va frenar quan jo volia córrer, i la primera reacció que vaig tindre va ser una punta d'impaciència —«fes-ho i prou, no cal tanta cerimònia per a un script d'actes»—, la mateixa impaciència, ara que ho pense, que deuen sentir alguns dels meus alumnes quan els demane que revisen l'exercici abans d'entregar-lo. «Ja funciona, per què he de mirar-ho més?».

Vam mirar els permisos del fitxer:

```
$ ls -l generar_actes.sh
-rw-r--r-- 1 samuel samuel 842 jul. 20 09:14
  generar_actes.sh
```

La IA em va traduir eixa filera de lletres que jo havia vist mil voltes sense llegir-la mai de veres: `rw-r--r--`. Jo podia llegir i escriure eixe arxiu, però ningú —ni jo mateix— tenia permís per a *executar-lo*. No faltava ser root. Faltava, simplement, marcar

l'arxiu com a executable. `sudo` no hauria arreglat res de fons: hauria forçat l'execució fent-me passar per root, amb tots els poders del sistema desbloquejats de colp, per a solucionar un problema que no tenia absolutament res a vore amb ser root.

«I si ho haguera fet igualment?», li vaig preguntar, ja més calmat. La resposta em va quedar gravada: `sudo` no és un pedaç per a quan una cosa no funciona. És obrir totes les portes de la casa de colp, no només la que et cal travessar. Si el script tenia un error, executat com a root eixe error podria fer coses que, com a usuari normal, mai hauria pogut fer —esborrar arxius de sistema, per exemple, amb la mateixa obediència cega que vaig descobrir el dia del `rm -rf`. `sudo` no pregunta si n'estàs segur. Fa exactament el mateix que faria sense `sudo`, però sense cap dels límits que normalment et protegixen de tu mateix.

Vaig arreglar-ho de veres:

```
$ chmod +x generar_actes.sh
$ ./generar_actes.sh
Actes generades correctament.
```

Sense root. Sense obrir cap porta de més. Només afegint el permís exacte que faltava, ni un més.

Eixa nit vaig entendre una cosa que no m'esperava d'un llibre sobre terminals: que hi ha una diferència enorme entre *fer que un error desaparega* i *entendre per què hi era*. La primera opció és més ràpida. La segona és l'única que de veres t'ensenya alguna cosa. I vaig pensar, també, en com de sovint jo mateix, davant d'un alumne bloquejat, li done la solució ràpida només per acabar abans, en lloc d'aturar-me a preguntar-li «saps per què et passa açò?». La IA, eixa nit, va fer amb mi el que jo hauria d'haver fet més voltes amb ells.

Apunt tècnic: permisos, `chmod` i per què `sudo` no és un botó màgic

Llegir els permisos d'un arxiu, amb `ls -l`:

```
$ ls -l generar_actes.sh
-rw-r--r-- 1 samuel samuel 842 jul. 20 09:14
  generar_actes.sh
```

Els deu primers caràcters es lligen en quatre blocs. El primer diu si és un directori (`d`) o un arxiu

normal (-). Els tres blocs de tres que segueixen són **lectura (`r`), escriptura (`w`) i execució (`x`)**, repetits per a tres nivells: el propietari, el grup, i «tots els altres». `rw-r--r--` vol dir: el propietari pot llegir i escriure (però no executar), i el grup i la resta només poden llegir.

Afegir permís d'execució, quan és això el que falta —el cas més habitual amb scripts baixats o creats de nou:

```
$ chmod +x generar_actes.sh
```

Canviar-ho amb números, si prefereixes eixa notació (cada permís val un número: lectura 4, escriptura 2, execució 1, i se sumen per nivell):

```
$ chmod 755 generar_actes.sh   # rwx per al propietari,
  r-x per a la resta
```

Canviar el propietari d'un arxiu (açò sí que sol requerir permisos elevats):

```
$ sudo chown samuel:samuel arxiu.txt
```

I l'apunt de fons, el que de veres importa recordar: **`sudo` (*superuser do*) t'executa la comanda com si fores l'usuari `root`, l'usuari sense límits**

del sistema. Té sentit fer-lo servir quan la tasca *de veres* requerix eixe nivell —instal·lar programari a nivell de sistema, per exemple. No té sentit fer-lo servir com a reflex davant de qualsevol error, perquè elimina totes les proteccions que normalment t'impedixen fer-te mal a tu mateix per accident. Abans de teclejar `sudo`, val la pena preguntar-se —o preguntar-li-ho a la IA, com vaig fer jo— «per què em diu que no?», en lloc de «com li faig callar?».

En Windows, el concepte equivalent és «Executar com a administrador», i en macOS els permisos i `sudo` funcionen exactament igual que en Linux, perquè per davall també és un Unix.

6

Descobrir les pipes i entendre per fi la filosofia Unix: programes xicotets que es parlen entre ells.

Estava corregint la primera tanda d'exercicis del curs de terminal que havia acabat proposant al meu propi institut —sí, vaig acabar donant-lo jo, amb el que havia aprés en estes setmanes—, i volia saber una cosa senzilla: quants dels meus capítols d'este mateix llibre parlaven encara de la paraula «por». Havia decidit, després del capítol 6, que valia la pena revisar-ho: si el llibre s'havia tornat massa por i massa poc descoberta.

La manera antiga de fer-ho —l'única que coneixia— hauria sigut obrir cada arxiu, un per un, i llegir-lo sencer buscant la paraula amb els ulls. Catorze capítols. Un diumenge sencer.

Li ho vaig demanar a la IA, i la comanda que em va donar em va deixar mirant la pantalla més temps del que hauria admés:

```
$ cat capitulos/*.md | grep -c "por"
7
```

Una línia. Un segon. Set capítols amb la paraula «por». Però el que em va fascinar no va ser el resultat, sinó com s'havia construït eixa línia. `cat capitulos/*.md` no feia res sofisticat: només ajuntava el contingut de tots els arxius, un darrere l'altre, i l'escopia per pantalla. `grep -c "por"` tampoc feia res complicat: comptava quantes línies contenien eixa paraula. Cap dels dos, per separat, contestava la meua pregunta. Junts, sí.

El símbol que els unia, eixa barra vertical —`|`—, no era un connector qualsevol. Era, literalment, l'eixida d'un programa convertida en l'entrada de l'altre, sense passar per cap arxiu intermedi, sense que jo haguera de fer res més que posar-los en fila. Una canonada de veres, d'aigua que ix d'una aixeta i entra directament en la següent, sense taronja de per mig.

Li vaig preguntar a la IA per què no hi havia una sola comanda que fera «comptar paraules en un munt d'arxius» de colp, com una eina tota-en-una. La resposta em va obrir una porta que encara no havia trepitjat: en Unix, la filosofia és exactament la contrària. Cada eina fa una cosa, i només una,

i la fa bé: `cat` ajunta, `grep` busca, `wc` compta, `sort` ordena, `uniq` elimina repetits. Cap d'elles sap res de les altres. No calia. El que les feia poderoses no era la seua intel·ligència individual, sinó la seua capacitat d'encadenar-se amb qualsevol altra que parlara el mateix idioma —text, simple i pla— entrant per un costat i eixint per l'altre.

Em vaig quedar pensant en açò més del que esperava, perquè em recordava, d'una manera estranya, com funciona un claustre ben avingut. Jo no sé fer de tot: no sóc l'expert en necessitats educatives especials, ni el que millor porta l'administració, ni el que millor calma un pare enfadat. Cadascú de nosaltres fa una cosa, i quan una faena passa per les mans justes, en l'orde just, ix millor que si un sol de nosaltres, per bo que fóra, l'haguera fet tota sol. La pipa no és només una tècnica de programació. És, ara que ho pense, com haurien de funcionar moltes coses que fem entre humans i que insistim a voler fer en solitari.

Vaig acabar eixa vesprada encadenant comandos com qui juga: comptar quants capítols tenien la paraula «terminal» (`grep -l "terminal"`

`capitulos/*.md | wc -l`), veure quins arxius eren els més llargs (`wc -l capitulos/*.md | sort -n`). Cap resposta que necessitara de veres, ja. Ho feia només pel plaer de vore com les peces encaixaven soles.

Apunt tècnic: la pipa (`|`) i quatre eines que hi combinen bé

Què fa exactament el `|`: agafa l'eixida (l'*stdout*) del programa de l'esquerra i la posa directament com a entrada (*stdin*) del programa de la dreta. Sense arxius temporals, sense passos intermedis.

`cat` (*concatenate*) mostra el contingut d'un o més arxius, seguits:

```
$ cat capitulos/*.md
```

`grep` busca línies que continguen un text concret. Amb `-c` compta coincidències; amb `-l` mostra només els noms dels arxius on apareix; amb `-r` busca en tot un directori i els seus subdirectoris:

```
$ grep -l "terminal" capitulos/*.md
```

`wc` (*word count*) compta línies (`-l`), paraules (`-w`) o caràcters (`-c`):

```
$ ls capitulos | wc -l
15
```

`sort` ordena línies (alfabèticament, o numèricament amb `-n`), i **`uniq`** elimina línies repetides consecutives (sovint es combinen: `sort | uniq` per a treure duplicats de veres).

Amb només estes quatre eines i el `|` per a encadenar-les, es responen la majoria de preguntes del dia a dia sobre arxius de text —sense necessitat de cap eina «especialitzada» que faça exactament allò que et cal, perquè quasi mai n'hi ha una que ho faça exactament com tu vols. Sempre és més fàcil combinar peces xicotetes que trobar la peça perfecta ja feta.

man man

Com canvia aprendre quan tens copilot: del manual que no entenia al diàleg que sí; preguntar bé és el nou memoritzar.

Vaig decidir, un dissabte de matí amb massa café i massa orgull, que ja n'hi havia prou de preguntar-li-ho tot a la IA. Volia demostrar-me a mi mateix que podia aprendre «de veres», a la vella usança, sense copilot. Volia ordenar els arxius de la carpeta `capitulos` per grandària, i en lloc de preguntar-ho, vaig fer una cosa que no feia des de fa vint anys: vaig obrir el manual.

```
$ man ls
```

I ahí em vaig trobar de cara amb un vell conegut que no recordava tan hostil. Pàgines i pàgines de text corregut, sense colors, sense exemples, amb noms de flags —`-S`, `--sort=WORD`, `-r`, `--reverse`— explicats en una prosa tan seca que semblava escrita expressament per a desanimar-te. Vaig llegir tres paràgrafs i no en vaig entendre cap del tot. Vaig recordar, de colp, per què als huitanta la gent no

«aprenia» l'ordinador llegint manuals: aprenia per prova i error, i quan tenies sort, algú al costat te'l traduïa a paraules normals.

Amb un toc d'ironia que no vaig poder evitar-me, vaig escriure:

```
$ man man
```

El manual del manual. Volia vore si almenys eixe m'ajudava a llegir els altres millor. No em va ajudar gens. Era, si cap, encara més àrid.

Vaig rendir-me al cap de vint minuts, amb la mateixa sensació de derrota que sent un alumne quan es baralla amb un enunciat de matemàtiques escrit per algú que ha oblidat com és no entendre'l. Li ho vaig preguntar a la IA, quasi avergonyit d'haver-ho intentat en solitari i haver fracassat: «vull ordenar els arxius de `capitulos` per grandària, de més gran a més xicotet». En dos segons, la resposta:

```
$ ls -lS capitulos
```

Una línia. Cap flag memoritzat. Cap pàgina de manual. Només havia hagut de dir, en la meua pròpia llengua, què volia aconseguir.

I ahí és on em vaig aturar a pensar de veres. Als huitanta, aprendre informàtica volia dir memoritzar: memoritzar comandos, memoritzar flags, memoritzar l'orde exacte dels paràmetres perquè el sistema no et perdonava ni una coma fora de lloc. Jo havia sigut bo en açò de jove —tenia bona memòria, i la informàtica d'aleshores premiava eixa mena de memòria mecànica. Però ara, cinquanta anys, la memòria ja no és el que era, i durant un moment m'havia fet por que això volguera dir que ja no podia «tornar a ser bo» en açò.

La IA em va fer entendre una cosa que canviava completament la pregunta: ja no calia memoritzar les respostes. Calia aprendre a fer bones preguntes. I això és una habilitat diferent, que no depén tant de la memòria com de saber què vols dir amb precisió i honestedat sobre allò que no saps. «Vull ordenar per grandària» és una frase que qualsevol persona amb prou vocabulari pot formular. `ls -lS` és una seqüència de caràcters que només algú que l'ha memoritzada pot escriure de memòria.

Vaig pensar, també, en els meus alumnes més joves, els que han crescut ja preguntant-li coses a

un assistent abans que buscant-les en un manual. Durant anys havia pensat que això els feia més ganduls, que «no aprenien de veres» perquè no memoritzaven. Eixe dissabte vaig entendre que potser estava jutjant-los amb la vara de mesurar equivocada. Preguntar bé —amb precisió, sabent què vols i sabent reconéixer quan la resposta no encaixa— no és una manera més fàcil d'aprendre. És una manera diferent, i potser no pitjor.

Això sí: el manual no va desaparéixer de la meua vida del tot. Vaig aprendre, eixes mateixes setmanes, que la millor manera d'usar-lo no era llegir-lo de dalt a baix com un llibre, sinó fer-lo servir com a diccionari: quan la IA em donava una comanda amb un flag que no coneixia, ara sí que anava a `man` a buscar eixe flag concret, ja amb la pregunta feta i només la peça que em faltava per confirmar. El manual no havia deixat de servir. Havia deixat de ser el primer pas, per a convertir-se en l'últim.

Apunt tècnic: quan usar `man` i quan preguntar

`man <comanda>` obri el manual d'eixa comanda. Es navega amb les fletxes, s'ix amb `q`, i es busca text dins amb `/paraula` seguit d'Intro:

```
$ man ls
$ man man
```

No tots els sistemes tenen `man` per defecte — macOS sí, la majoria de distribucions Linux també, però algunes instal·lacions mínimes no. Si falta, sol instal·lar-se amb el gestor de paquets del sistema.

Una alternativa més curta i sovint més clara, present en moltes comandes modernes, és el flag `--help`:

```
$ ls --help
```

Dóna un resum més compacte que `man`, pensat per a consultar ràpid en lloc de llegir sencer.

Quan val la pena cadascú:

- **Pregunta-li a la IA** quan saps *què vols aconseguir* però no saps quina comanda o flag et fa

falta. És l'ús més eficient: descriure l'objectiu, rebre la comanda exacta.

- **Consulta `man` o `--help`** quan ja tens la comanda i vols entendre un flag concret amb precisió, o quan vols vore *totes* les opcions disponibles d'una eina que faràs servir sovint —eixa visió de conjunt és una cosa que la conversa, per la seua pròpia naturalesa de pregunta-resposta, no sempre dóna tan bé.

Cap de les dos maneres és «la de veres». Totes dos són eines, i el que ha canviat —això és, en el fons, tot este capítol— és que ara pots triar quina fer servir en cada moment, en lloc d'estar obligat a una de sola.

`rsync -av`

Una conversa sencera, sense retocar, sobre com no tornar a perdre `notes` mai més.

No vull contar esta com les altres. Vull posar-la ací tal com va passar, perquè crec que és l'única manera de mostrar de veres què vol dir «preguntar bé» després de tot el que el capítol anterior deia sobre el tema. Cap resum meu explicaria millor el procés que el procés mateix.

SAMUEL: vull fer una còpia de seguretat de la meua carpeta de notes

IA: Puc ajudar-te. Abans de proposar-te res, tres preguntes: on és eixa carpeta ara mateix? On vols que vaja la còpia? I amb quina freqüència la necessites —una còpia puntual, ara mateix, o alguna cosa que es faça sola cada dia?

SAMUEL: és `~/Libros/command-not-found/notes`. la còpia que vaja a un disc extern que tinc muntat a `/media/samuel/backup`. i sí, m'agradaria que fóra

automàtica, cada dia, perquè l'última volta que ho vaig fer a mà me'n vaig oblidar durant sis setmanes i ja saps com va acabar

IA: Ho recorde (capítol 6). Anem per parts, doncs: automàtica i diària. Una primera proposta senzilla:

```
rsync -av ~/Libros/command-not-found/notes
  /media/samuel/backup/
```

`rsync` copia només el que ha canviat des de l'última volta —més ràpid que copiar-ho tot cada dia—, i `-av` vol dir «mode arxiu, amb detall»: conserva dates i permisos, i et mostra què fa. Perquè s'execute sola cada dia, la programaríem amb `cron`. Abans de continuar: vols que cada còpia sobreescriga l'anterior, o vols poder tornar arrere a un dia concret si un dia esborres alguna cosa sense voler?

SAMUEL: eixa segona. si l'haguera tingut fa uns mesos m'hauria estalviat un disgust

IA: Aleshores `rsync` sol no és prou — sobreescriuria l'anterior. Fem que cada còpia vaja a una carpeta amb la data:

```
rsync -av ~/Libros/command-not-found/notes
  /media/samuel/backup/notes_$(date +%F)/
```

`$(date +%F)` s'expandix a la data d'avui en format `2026-07-13`. Cada dia, una carpeta nova. Un detall important: si el disc extern no està connectat el dia que `cron` intenta fer la còpia, la comanda fallarà en silenci i no te n'assabentaràs. Vols que afegim alguna manera de saber-ho?

SAMUEL: no ho havia pensat. sí, m'agradaria saber-ho d'alguna manera

IA: La manera més senzilla, sense complicar-nos amb notificacions: que el script escriga en un arxiu de registre si ha anat bé o si ha fallat, i que tu el pugues consultar quan vulgues. Un script complet quedaria així:

```
#!/bin/bash
DESTI="/media/samuel/backup/notes_$(date +%F)"
if [ -d "/media/samuel/backup" ]; then
    rsync -av ~/Libros/command-not-found/notes "$DESTI"
      \
        && echo "$(date): còpia OK" >> ~/backup.log \
        || echo "$(date): còpia FALLIDA" >>
          ~/backup.log
else
    echo "$(date): disc no connectat" >> ~/backup.log
fi
```

Guarda'l com `~/backup_notes.sh` i dóna-li permís d'execució amb `chmod +x` —el del dia del `Permission denied`, ja el recordes. Vols que continuem amb el `cron` perquè s'execute sol cada dia?

SAMUEL: sí, i explica'm bé això de cron perquè no ho he tocat mai

IA: `cron` és un programa que porta dècades als sistemes Unix, dedicat exclusivament a executar comandos a hores fixes. S'edita amb:

```
$ crontab -e
```

I s'hi afegix una línia amb este format: minut, hora, dia del mes, mes, dia de la setmana, i la comanda. Per a executar el teu script cada dia a les set del matí:

```
0 7 * * * /home/samuel/backup_notes.sh
```

Els asteriscos volen dir «qualsevol»: qualsevol dia del mes, qualsevol mes, qualsevol dia de la setmana. Només fixem el minut (0) i l'hora (7).

SAMUEL: i si vull comprovar que ha funcionat sense esperar-me a demà?

IA: Executa el script a mà una volta, ara mateix, i mira el registre:

```
$ ~/backup_notes.sh
$ cat ~/backup.log
```

Si diu «còpia OK», ja saps que funciona; `cron` només és qui prem el botó per tu cada matí.

Ho vaig fer, eixa mateixa nit. Vaig executar el script a mà, vaig mirar el registre, vaig vore «còpia OK», i per primera volta en tot este viatge, vaig sentir que no havia aprés només una comanda solta, sinó que havia construït, entre pregunta i pregunta, una cosa completa que no existia una hora abans. Cap de les meues preguntes havia sigut brillant. Simplement havia contestat, una i altra volta, el que em preguntaven —i això, ara ho sé, és tota la diferència entre una conversa que t'ensenya i una que només et dóna respostes soltes.

Ctrl+C

La primera volta que la IA es va equivocar: desconfiança sana i l'hàbit de llegir abans d'executar.

Portava ja mesos confiant-me a la IA amb una tranquil·litat que, mirat en fred, feia una miqueta de basarda. Li preguntava, em donava la comanda, jo la copiava, la pegava, prement Intro amb la mateixa naturalitat amb què un alumne copia allò que dicta el professor sense qüestionar-s'ho. Havia funcionat tantes voltes seguides que havia deixat, sense adonar-me'n, de llegir abans d'executar.

Estava intentant alliberar espai al disc —el meu portàtil de l'institut anava carregant-se d'anys d'arxius de cursos vells— i li vaig demanar que m'ajudara a trobar i esborrar arxius temporals antics. Em va donar açò:

```
$ find / -name "*.tmp" -mtime +30 -delete
```

Vaig llegir «find», vaig llegir «tmp», vaig llegir «delete», i quasi vaig prémer Intro amb el mateix automatisme de sempre. Però eixe «quasi» és tota la

diferència d'este capítol: alguna cosa em va grinyolar per dins, un ressò llunyà del capítol 6, de `notes` esborrada per sempre sense paperera. Vaig aturar-me un segon abans d'executar, i vaig llegir de veres, no per damunt: `find /`. No `find .` ni `find ~`. La barra sola. Des de l'arrel de tot el sistema.

«Açò no buscaria en tot el disc, incloent-hi carpetes de sistema?», li vaig preguntar, amb la mateixa veu amb què li hauria preguntat a un alumne «segur que has llegit bé l'enunciat?». I la resposta, esta volta, no em va donar la raó del tot ni em va deixar tranquil: sí, `find /` buscaria literalment per tot arreu, i encara que el filtre `-mtime +30` (arxius de més de trenta dies) i el nom `*.tmp` reduïen molt el risc, no era impossible que trobara i esborrara algun arxiu temporal de sistema que algun programa encara necessitara. Més segur —em va reconéixer la IA mateixa, quan li ho vaig fer notar— hauria sigut acotar la busca a la meua pròpia carpeta d'usuari.

Eixe «em va reconéixer» és el que em va deixar més pensatiu que l'error en si. La IA no havia mentit ni m'havia enganyat a posta. Simplement havia donat la resposta més genèrica a una pregunta

genèrica, sense saber —perquè jo no ho havia dit— que només volia netejar la meua pròpia carpeta d'usuari, no rebuscar en tot el sistema operatiu. L'error no era seu del tot. Era, en bona part, meu, per haver preguntat sense prou precisió i per haver estat a punt d'executar sense llegir.

Vaig reescriure la comanda jo mateix, per primera volta amb seguretat real i no per rutina:

```
$ find ~ -name "*.tmp" -mtime +30 -delete
```

`~`, no `/`. La meua casa, no la casa de tots.

I encara vaig fer una cosa més, una que no havia fet mai fins aleshores: vaig executar-la primer sense el `-delete`, només per a vore la llista d'arxius que trobaria, abans de confirmar que de veres volia esborrar-los.

```
$ find ~ -name "*.tmp" -mtime +30
/home/samuel/.cache/curs_2019/apunts_temp.tmp
/home/samuel/.cache/lliurex_backup.tmp
```

Dos arxius, tots dos inofensius. Ara sí, amb el `-delete` afegit, amb confiança de veres i no de fe cega.

Eixa nit vaig entendre que la confiança que li tenia a la IA no havia de ser ni la confiança absoluta d'un principiant que ho creu tot, ni la desconfiança total d'algú que rebutja tota ajuda per por. Havia de ser una cosa intermèdia, la mateixa que li demane als meus alumnes quan els parle de fonts d'informació: no cregues tot el que llegixes, però tampoc ho rebutges tot per sistema. Llig, pensa, pregunta per què, i només aleshores, prem Intro.

Apunt tècnic: llegir abans d'executar, i com aturar-se a temps

`Ctrl+C` interromp qualsevol comanda que s'estiga executant en eixe moment al terminal —útil quan alguna cosa s'ha penjat, va per mal camí, o simplement vols aturar-la abans que acabe. No desfà el que ja s'haja fet, però evita que continue fent-ho.

Abans d'executar una comanda que esborra o modifica coses en massa, dos hàbits senzills:

1. **Prova-la primer sense l'acció destructiva.** Moltes comandes, com `find`, permeten vore què *trobarien* abans de dir-los què *fer* amb això:

   ```
   $ find ~ -name "*.tmp" -mtime +30        # només
     mostra
   $ find ~ -name "*.tmp" -mtime +30 -delete  # ara
     sí, esborra
   ```

2. **Llig cada part de la comanda, no només el verb.** En `find / -name "*.tmp" -mtime +30 -delete`, el que canvia tot és eixe primer paràmetre: `/` (tot el sistema) contra `~` (la teua carpeta) contra `.` (la carpeta actual, com al capítol 4).

Sobre confiar en una IA: una IA de terminal no menteix a posta, però respon a la pregunta que li has fet, no a la que tenies al cap. Com més concreta i completa siga la pregunta —«només en la meua carpeta d'usuari», «sense tocar res de sistema»—, més fiable serà la resposta. I encara que la pregunta siga perfecta, la comanda final sempre val la pena llegir-la abans de prémer Intro: la IA proposa, però qui executa —i qui n'és responsable— eres tu, amb el dit sobre la tecla.

grep CRON /var/log/syslog

El dia que vaig perdre quatre hores senceres darrere d'un error que no tenia cap solució ràpida.

Volia contar esta història sense embellir-la, perquè crec que el llibre, fins ara, l'havia posat massa fàcil: cada volta que alguna cosa fallava, arribava una resposta que ho arreglava en un parell de línies. No sempre és així. A vegades es perden hores senceres, i la IA no té una vareta màgica per a estalviar-te-les.

Feia tres setmanes que tenia el meu script de còpies de seguretat funcionant —eixe que vaig construir amb la IA en aquella conversa que vaig transcriure sencera, `backup_notes.sh`, programat amb `cron` per a executar-se cada matí a les set. L'havia provat a mà, havia vist «còpia OK» al registre, i m'havia oblidat d'ell amb la tranquil·litat de qui creu que un problema ja està resolt per sempre.

Un dijous, per pura casualitat, vaig obrir el disc extern per a buscar una nota d'una setmana arrere. No hi era. Vaig mirar dos setmanes arrere. Tampoc.

La carpeta `backup` tenia còpies fins fa exactament tres setmanes, i cap més. `cron` havia deixat de funcionar, en algun moment, sense avisar-me de res.

Vaig obrir el registre, `~/backup.log`, esperant trobar-hi la resposta escrita amb tota claredat, com sempre havia passat fins aleshores.

Estava buit. Ni «OK» ni «FALLIDA». Res. Com si el script, senzillament, no s'haguera executat mai més.

Vaig executar-lo a mà. Va funcionar perfectament, «còpia OK» i tot. El problema no era el script en si —era el script executant-se *des de cron*, no des de mi. I ahí van començar les quatre hores més frustrants que recorde d'este llibre sencer.

Vaig preguntar-li a la IA, per descomptat. Em va donar tres hipòtesis raonables: potser el disc no estava muntat a l'hora exacta que `cron` intentava la còpia; potser un problema de permisos; potser la ruta `~` no s'expandia igual dins de `cron` que en una terminal normal meua. Vaig comprovar les tres, una per una, durant la primera hora. El disc estava muntat a les set del matí —ho vaig confirmar deixant l'ordinador encés tota la nit i mirant-ho. Els

permisos eren correctes. I quan vaig canviar el `~` per la ruta completa `/home/samuel/...`, tampoc va canviar res.

Cap a la segona hora, ja amb la paciència prima, vaig fer una cosa que no havia fet mai: buscar directament als registres del sistema, no del meu script, sinó de `cron` mateix.

```
$ grep CRON /var/log/syslog
```

I ahí, per fi, una pista de veres: `cron` sí que intentava executar el meu script cada matí, però el script fallava d'immediat, abans fins i tot d'arribar a la línia de `rsync`. La IA em va suggerir una cosa que no havia considerat: `cron` executa els seus treballs amb un entorn mínim, gairebé buit, molt diferent del que jo tenia obrint un terminal normal. El meu `$PATH`, ple de coses que havia anat afegint al `~/.bashrc` —incloent-hi, sense saber-ho, la ruta on estava instal·lat `rsync` en el meu sistema—, simplement no existia dins de `cron`. El script cridava `rsync`, i `cron`, amb el seu PATH minúscul, no sabia on trobar-lo.

Quatre hores. Per una línia que faltava al principi del script:

```
PATH=/usr/local/bin:/usr/bin:/bin
```

Vaig quedar-me mirant eixa línia, ridículament curta comparada amb les quatre hores que m'havia costat trobar-la, i no vaig sentir la satisfacció ràpida d'altres capítols. Vaig sentir, més aïna, un cansament net, honest, el mateix que sent després d'una tarda sencera corregint exàmens sense fer cap pausa. No tot problema té una solució elegant a l'abast de la mà. De vegades, la solució és senzilla, però trobar-la costa hores de descartar, un per un, tots els camins que no eren.

I encara així —potser per això mateix—, quan finalment vaig vore «còpia OK» tornar a aparéixer al registre l'endemà al matí, sense que jo haguera tocat res, la satisfacció va ser més gran que la de qualsevol capítol anterior. No perquè la solució fóra millor. Perquè m'havia costat de veres.

Apunt tècnic: per què `cron` falla en silenci (i com fer-lo parlar)

`cron` **executa amb un entorn mínim**, molt més pobre que el d'una sessió de terminal normal: un `$PATH` bàsic, sense les variables que hages afegit tu al `~/.bashrc` o `~/.zshrc`. Un script que funciona perfectament a mà pot fallar sota `cron` per esta única raó.

Solucions habituals:

1. **Fixa el `PATH` dins del propi script**, a la primera línia útil:

   ```
   PATH=/usr/local/bin:/usr/bin:/bin
   ```

2. **Usa rutes absolutes per a tot**, comandos inclosos, en lloc de confiar que estaran al `PATH`:

   ```
   /usr/bin/rsync -av ...
   ```

3. **Redirigix sempre l'eixida i els errors a un arxiu de registre**, amb `2>&1` (un símbol estrany del qual parlaré més avant), perquè si torna a fallar, almenys queda escrit per què.

Per a investigar quan `cron` no fa el que esperes:

```
$ grep CRON /var/log/syslog      # Linux amb syslog
  clàssic
$ journalctl -u cron             # Linux amb systemd
$ log show --predicate 'process == "cron"' --last 1d
  # macOS
```

I un últim consell, aprés a força d'hores perdudes: proveu sempre un script pensat per a `cron` amb un entorn tan buit com el que tindrà de veres, abans de confiar que «si funciona a mà, funcionarà sol»:

```
$ env -i /home/samuel/backup_notes.sh
```

`env -i` executa la comanda amb un entorn pràcticament net, exactament el tipus de sorpresa que et pot esperar de matinada si no ho proves abans.

~/.bashrc

Dotfiles, alias i un prompt a mida: el dia que el terminal va deixar de ser un lloc hostil i va començar a ser ma casa.

Hi ha una frase que li vaig dir a la IA sense pensar-la massa, un vespre qualsevol d'estes últimes setmanes, i que després no vaig poder deixar de rumiar: «cada volta que òbric el terminal em sent com si entrara a casa d'un altre». Tot funcionava, ja no em feia por, ja sabia moure'm —però era una casa nua, sense fotos a la paret, sense el desori familiar que fa que una casa siga *la teua* casa i no un pis d'hotel.

Encara escrivia `cls` de tant en tant, per pur costum de dècades, i el terminal em contestava amb el seu `command not found` de sempre, el mateix que li dóna nom a este llibre. Ja no em molestava —sabia exactament per què passava des d'aquella primera setmana del `cls`—, però continuava sent una xicoteta fricció diària, un recordatori que jo m'havia

d'adaptar jo a la casa, en lloc que la casa s'adaptara un poquet a mi.

«Es pot fer que `cls` funcione ací, encara que no siga la comanda "correcta"?», li vaig preguntar a la IA, ja mig en broma. La resposta em va obrir una porta que no sabia que existia: sí, i no només això —quasi qualsevol cosa del terminal es pot personalitzar, si saps on tocar.

Eixe «on tocar» resulta que és un arxiu que porta anys ahí, invisible, esperant que algú l'obrira: `~/.bashrc` (o `~/.zshrc`, des que vaig canviar de shell al capítol 5). Un arxiu de configuració que s'executa cada volta que obres un terminal nou, i on pots deixar escrites totes les teues manies perquè et reben cada volta que arribes.

La primera línia que hi vaig afegir, amb un somriure que ningú va vore:

```
alias cls='clear'
```

Xicoteta, ridícula quasi, però amb un pes simbòlic que no esperava: era la primera volta, en tot este viatge, que en lloc d'adaptar-me jo al terminal, era el terminal qui s'adaptava un poc a mi.

Vaig continuar afegint coses, les setmanes següents, com qui va penjant quadres per les parets d'una casa nova. Un alies per a no escriure `ls -la` cada volta:

```
alias ll='ls -la'
```

Un altre per a tornar ràpid a la carpeta del llibre, farta ja de teclejar la ruta sencera cada volta:

```
alias llibre='cd ~/Libros/command-not-found'
```

I finalment, el que més em va enamorar de tot: un prompt a mida. El prompt és eixe text que apareix just abans d'on escrius, el `$` pelat que havia vist des del primer dia —i que dóna nom al primer capítol d'este llibre. Vaig descobrir que eixe `$` també es podia personalitzar, i li vaig demanar a la IA que m'ajudara a fer-ne un que mostrara la carpeta on estava (adéu per fi als desorients del capítol 4) i l'hora, en un color verd suau que no cridava massa l'atenció:

```
export PS1='\[\e[32m\]\w \[\e[0m\]\$ '
```

La primera volta que vaig obrir un terminal nou després de guardar tots eixos canvis, vaig sentir una

cosa que no m'esperava sentir mai per un programa de línies negres: familiaritat. El `pwd` ja no calia teclejar-lo cada volta perquè ara vivia sempre a la vista, al prompt mateix. El `cls` funcionava. Els meus dreceres hi eren. Era, per fi, un lloc que reconeixia com a meu.

Vaig pensar, tancant l'ordinador eixa nit, en com havia fet exactament el mateix, any rere any, amb la meua pròpia aula: cadires movides d'una manera concreta, pòsters que jo havia triat, una manera pròpia de començar cada classe que no eixia de cap manual de pedagogia. Un espai no es fa casa perquè algú te'l done fet. Es fa casa perquè hi deixes la teua empremta, xicoteta i acumulada, dia rere dia.

Apunt tècnic: dotfiles, alies i el prompt

`~/.bashrc` (bash) o `~/.zshrc` (zsh) és un arxiu de configuració que s'executa cada volta que obris un terminal nou. Tot el que hi escrigues —variables, alies, funcions— estarà disponible en cada sessió futura. S'anomenen genèricament **dotfiles** perquè

comencen per un punt (`.`), la convenció Unix per a arxius «ocults» de configuració.

Crear un alies, és a dir, un nom curt per a una comanda llarga o que t'agrada escriure diferent:

```
alias cls='clear'
alias ll='ls -la'
```

Després d'afegir línies a l'arxiu, cal recarregar-lo perquè el terminal ja obert se n'assabente (un terminal nou ja les llig soles):

```
$ source ~/.bashrc
```

Personalitzar el prompt amb la variable `PS1` (bash) o `PROMPT` (zsh) —hi ha generadors en línia que et deixen triar colors i informació (carpeta, hora, branca de git) sense haver de memoritzar els codis d'escapament un per un.

On viuen estos arxius en cada sistema:

- **Linux i macOS:** `~/.bashrc` o `~/.zshrc`, segons la shell (recorda el capítol 5: zsh és el valor per defecte a macOS des del 2019).
- **Windows:** amb PowerShell, l'equivalent és el perfil (`$PROFILE`), un arxiu `.ps1` que s'executa a l'inici de cada sessió; amb WSL, s'usa el mateix

`~/.bashrc`/`~/.zshrc` que en Linux, perquè per davall corre un Linux de veres.

Un últim consell, després de mesos ficant coses en el meu: no cal fer-ho tot de colp. Una casa es decora a poc a poc, no en un vespre. Cada volta que et molesta una xicoteta fricció repetida, eixe és el moment de preguntar-li a la IA «es pot arreglar açò en el meu `.bashrc`?». Quasi sempre la resposta és sí.

`git merge`

El fill que em va ficar la puça a l'orella, un any després, truca a la porta amb un problema de veres.

Va vindre a dinar un diumenge, com quasi cada diumenge, i mentre jo acabava de parar taula el vaig vore assentar-se davant del meu portàtil, obert encara del matí. Va obrir un terminal —sense preguntar, com fa qui es sent a casa seua— i es va quedar mirant la pantalla més estona del que jo esperava.

«Pare, això és teu?»

El meu prompt verd, amb la carpeta actual sempre a la vista. Els meus alies. `ll` en lloc de `ls -la`. Fins i tot `cls`, funcionant, gràcies a l'alies que li vaig confessar amb una mica de vergonya que havia posat perquè no aconseguia desaprendre'l.

Va somriure d'una manera que no li havia vist des que era xicotet, ensenyant-li a un amic un dibuix que havia fet ell mateix. «No em pensava que t'hi ficaries de veres», va dir. «Jo només et vaig dir que provares una cosa».

Vam dinar, i cap al café em va contar el que de veres l'havia dut ahí eixe diumenge amb el portàtil sota el braç: un projecte de la universitat, en equip, i un embolic de `git` que no sabia com desfer. Havia treballat tota la setmana en una branca pròpia, i quan va intentar ajuntar-la amb la dels companys, el terminal li havia escopit un mur de text que no entenia, ple de fletxes i asteriscos i la paraula `CONFLICT` repetida quatre voltes.

«I la IA què et diu?», li vaig preguntar, mig en broma, recordant totes les voltes que ell m'havia dit exactament la mateixa frase a mi, un any arrere.

«Li ho he preguntat, però em dóna passos i em fa por seguir-los sense entendre'ls. Si m'equivoque ací, es carreguen la faena de tot l'equip, no només la meua».

I ahí, per primera volta en tot este llibre, vaig ser jo qui es va assentar davant del teclat, i ell qui va mirar per damunt de la meua espatlla.

No en sabia tant de `git` com per a resoldre-ho amb els ulls tancats —encara sóc, en el fons, un aprenent en açò—, però sabia una cosa que ell, amb tota la pressa de l'entrega de dilluns, havia oblidat:

llegir abans d'executar. Vam obrir junts l'arxiu en conflicte, vam llegir les marques que `git` deixa —`<<<<<<<`, `=======`, `>>>>>>>`— per a indicar exactament què venia de cada branca, i li vaig explicar, amb les meues paraules de professor més que amb les de programador, que allò no era un mur, era una pregunta: «quina d'estes dos versions vols, o vols les dos?».

Vam triar, línia a línia, amb calma, exactament com jo triava paraules quan corregia un examen. `git add` sobre l'arxiu resolt, `git commit`, i quan va vore el missatge confirmant que la fusió s'havia completat, el va llegir dos voltes, com no s'havia cregut que fóra tan senzill una volta enteses les peces.

«Pensava que necessitaria hores», va dir.

«Jo n'he perdut quatre, algun dia, per coses molt més xicotetes», li vaig contestar, i em vaig sentir estranyament orgullós de poder dir-ho amb normalitat, sense que sonara a confessió.

Es va acomiadar content, amb el `merge` resolt i el dinar de diumenge fet una miqueta llarg. Jo em vaig quedar assentat davant del terminal una estona

més, sense fer-hi res en concret, només pensant que fa un any eixe mateix fill m'havia posat davant d'una pantalla negra que no entenia, i que ara, dotze mesos després, era jo qui li havia ensenyat a llegir-la. No perquè jo en sàpia més que ell —encara en sap molt més, de fet—, sinó perquè, per un instant, la calma que jo havia aprés a força d'errors va poder prestar-se-li a ell, que encara no havia tingut temps de fer-se'n.

No sé si a açò se li diu ser pare. Però se li assembla molt.

Apunt tècnic: entendre (i resoldre) un conflicte de `git merge`

Quan dos branques de `git` han modificat les mateixes línies d'un arxiu, un `git merge` no sap sol quina versió és la correcta, i marca el conflicte directament dins de l'arxiu:

```
<<<<<<< HEAD
El teu text (la branca on estàs ara).
=======
El text de l'altra branca.
```

```
>>>>>>> branca-companys
```

Per a resoldre'l:

1. Obri l'arxiu i decidix, línia a línia, què vols conservar —una versió, l'altra, o una barreja de les dos.
2. Esborra les marques (`<<<<<<<`, `=======`, `>>>>>>>`) i deixa només el text final que vols.
3. Marca'l com a resolt i confirma la fusió:

   ```
   $ git add arxiu.md
   $ git commit
   ```

Per a vore, abans de tocar res, quins arxius tenen conflicte:

```
$ git status
```

Apareixeran marcats com a *both modified*. I si en algun moment vols rendir-te i tornar-ho tot arrere, sense por, sempre es pot abandonar la fusió sencera:

```
$ git merge --abort
```

Cap conflicte de `git` és tan urgent com sembla el primer cop. Es pot llegir amb calma, línia a línia, exactament igual que es llig qualsevol altra cosa abans d'executar-la.

apt install

Tour per les descobertes que em van fer sentir dins del club: grep, ssh, tmux, git i altres eines dels majors.

Hi ha una experiència que tots els docents coneixem, encara que no la parlem massa: eixa reunió de claustre on algú esmenta, de passada, com si fóra la cosa més normal del món, una eina o un mètode que tu no havies sentit anomenar mai, i tot el grup assenteix com si ho sabera tothom. Un instant curtíssim de vergonya —«hauria de saber açò?»— seguit, si tens sort, de curiositat de veres.

Em va passar exactament açò, però amb el terminal, un matí que un company d'informàtica va dir, com qui no vol la cosa: «jo em connecte al servidor de l'institut per SSH i faig el manteniment amb `tmux` obert, per si es talla la connexió». Vaig assentir, com sempre. I eixa vesprada, per fi sol amb la IA, vaig confessar la meua ignorància sense vergonya —ja n'havia perduda prou, de vergonya, en estos mesos— i li vaig demanar que m'ensenyara eixes

eines «dels majors», les que fan servir els que porten anys en açò.

El que va vindre a continuació va ser el més paregut a una visita guiada per un barri que no sabia que existia dins de la meua pròpia ciutat.

`grep`, ja el coneixia una miqueta des del capítol de les canonades, però la IA em va ensenyar que era molt més potent del que havia usat: buscar amb expressions regulars, buscar en tot un projecte sencer amb `-r`, ressaltar les coincidències amb colors. Vaig entendre que no era una eina «per a comptar paraules en un llibre», com l'havia fet servir jo, sinó una de les eines més usades del món per a bussejar dins de codi i registres de sistema.

`ssh` em va obrir una porta literal: connectar-me a un altre ordinador, des del meu terminal, com si estiguera assegut davant. Vaig provar-ho amb el servidor de l'institut, amb el cor una miqueta accelerat la primera volta —«i si trenque alguna cosa de lluny?»—, i quan vaig vore el prompt canviar de nom, indicant que ja no estava al meu portàtil sinó a la màquina remota, vaig sentir una emoció sem-

blant a la de conduir per primera volta: el mateix volant, un cotxe distint.

`tmux` —el que havia esmentat el meu company— em va costar més d'entendre, i encara més d'estimar. És un «multiplexor de terminal»: et deixa obrir diverses sessions dins d'una de sola, i sobretot, deixar-les corrent encara que tanques la connexió. La IA em va posar un exemple que em va convéncer de colp: si estàs fent una còpia de seguretat llarga per SSH i se't talla el wifi, sense `tmux` la còpia s'atura a mitges; amb `tmux`, continua corrent al servidor remot com si res, i quan et reconnectes, ahí la trobes, exactament on la vas deixar.

`git`, l'havia usat ja, de fet —era com havia aprés a guardar les versions d'este mateix llibre—, però sempre amb quatre comandes memoritzades a cegues (`add`, `commit`, `push`) sense entendre gran cosa més. Eixa nit vaig entendre per fi què és de veres: no un «botó de guardar avançat», sinó un sistema que recorda cada canvi, permet tornar arrere, i —el que més em va sorprendre— permet que diverses persones treballen sobre el mateix projecte sense trepitjar-se el treball les unes a les altres.

El que compartien totes estes eines, i açò sí que em va fer sentir «dins del club» de veres, no era la seua complexitat sinó la seua edat: `grep` té més de cinquanta anys, `ssh` porta des dels noranta, `git` és relativament jove i ja té dos dècades llargues. No eren moda passatgera. Eren, cadascuna a la seua manera, eines que generacions senceres de gent havien anat perfeccionant perquè funcionaven de veres. Fer-les servir no era «posar-me al dia» amb tecnologia nova. Era, més aïna, entrar per fi en una conversa que portava mig segle oberta.

Apunt tècnic: instal·lar i conéixer quatre eines clàssiques

Instal·lar programari nou al sistema es fa amb el gestor de paquets de cada SO:

```
$ sudo apt install tmux        # Debian/Ubuntu i
  derivats
$ brew install tmux            # macOS, amb Homebrew
$ winget install tmux          # Windows
```

`ssh usuari@servidor` obri una sessió remota. La primera volta et demanarà confirmar

l'«empremta» (fingerprint) del servidor —una mesura de seguretat per a assegurar que et connectes a qui penses:

```
$ ssh samuel@servidor-institut.local
```

`tmux` obri una sessió persistent. Els comandos bàsics per a començar:

```
$ tmux                  # obri una sessió nova
# Ctrl+b, després d      # es "desconnecta" sense
  tancar-la
$ tmux attach           # torna a connectar-hi
```

`git` en el seu ús més bàsic, per a guardar versions d'un projecte:

```
$ git add arxiu.md
$ git commit -m "descripció del canvi"
$ git push
```

`grep -r "paraula" .` busca «paraula» en tots els arxius de la carpeta actual i les seues subcarpetes —l'ús més habitual del dia a dia, molt més enllà del que jo n'havia fet al capítol de les canonades.

Cap d'estes eines cal dominar-les totes de colp. Jo encara no domine `tmux` del tot, i em va bé igual. El «club» no exigix saber-ho tot des del primer dia.

Exigix, només, no tindre por de preguntar quan algú esmenta una eina que no coneixes.

neomutt

Un dia vaig descobrir que no calia eixir del terminal per a res: correu, notícies, fins i tot el xat.

Va ser una tonteria la que ho va començar: tenia el terminal obert, amb el meu prompt verd i les meues carpetes ben ordenades, quan em va sonar una notificació al mòbil i, sense pensar-ho, vaig deixar el teclat per a mirar el correu al navegador. Vint minuts després, entre una pestanya i una altra —el correu, les notícies, un grup de WhatsApp de l'institut—, ja no recordava què estava fent al terminal quan havia començat.

Eixa nit, encara una miqueta empipat amb mi mateix, li vaig preguntar a la IA una cosa que em semblava mig absurda: «hi ha manera de llegir el correu sense obrir el navegador?». La resposta em va sorprendre: no només hi havia manera, hi havia tota una cultura de gent que vivia, literalment, dins del terminal per a quasi tot.

La primera eina que vaig instal·lar va ser **neomutt**, un client de correu que no s'assembla en res

al Gmail que conec de tota la vida —sense botons, sense colors cridaners, però amb dreceres de teclat que, al cap d'uns dies, em feien sentir més ràpid llegint i responent correu del que havia sigut mai amb el ratolí. Vaig configurar-hi el compte de l'institut, i la primera volta que vaig llegir un correu sencer sense alçar les mans del teclat, vaig sentir la mateixa satisfacció xicoteta que quan vaig estrenar el meu propi prompt a mida.

Després va vindre **newsboat**, un lector de feeds RSS. Portava anys seguint tres o quatre blogs d'educació i tecnologia educativa a força d'entrar-hi un per un, quan em'n recordava, que no era sovint. Amb newsboat els vaig apuntar tots en una llista, i cada matí, amb el café, un `r` per a refrescar i una llista neta de tot el que havia eixit nou, sense anuncis, sense res que no fóra el text que volia llegir.

I finalment, la que em va fer riure més: **nchat**, un client de xat de terminal que connecta amb WhatsApp i Telegram. El vaig instal·lar mig en broma, convençut que no funcionaria bé, i em vaig quedar mirant com els missatges del grup de coordinadors

de curs apareixien, en text pla, a la mateixa pantalla negra on hores abans havia estat corregint un script.

El que em va sorprendre, passades unes setmanes vivint així, no va ser la tecnologia en si —al cap i a la fi, era la mateixa informació de sempre—, sinó com de diferent em sentia consumint-la. Sense anuncis. Sense el disseny pensat, a posta, per a retindre'm més temps del que necessitava. Sense eixe impuls d'obrir «una pestanya més» que mai sabia com havia arribat a obrir. El terminal, per la seua pròpia austeritat, em tornava el control d'una cosa que no sabia que havia perdut: quant de temps volia dedicar-li a cada cosa, i quan ja n'hi havia prou.

Vaig pensar, aquells dies, en com els meus alumnes es queixen sovint que no poden concentrar-se, i en com jo mateix, cinquanta anys, patia exactament el mateix mal amb el mòbil i el navegador. No tenia la solució per a ells. Però per a mi, almenys, havia trobat una casa —eixa paraula que ja no em deixava des del dia del `.bashrc`— on ni tan sols els distractors sabien entrar.

Apunt tècnic: quatre eines TUI per a viure al terminal

Es diu **TUI** (*Text User Interface*) a les aplicacions que, tot i tindre menús, finestres i navegació com un programa gràfic, funcionen senceramente dins del terminal, amb text i teclat.

`neomutt` — client de correu:

```
$ sudo apt install neomutt   # Debian/Ubuntu
$ brew install neomutt       # macOS
```

Requerix una configuració inicial (compte, servidor IMAP) a `~/.config/neomutt/`, però una volta feta, gestiona correu amb una rapidesa que cap client gràfic iguala per a qui es fa a les dreceres de teclat.

`newsboat` — lector de feeds RSS/Atom:

```
$ sudo apt install newsboat
$ brew install newsboat
```

Els feeds es llisten a `~/.newsboat/urls`, un per línia; prement `r` es refresquen tots de colp.

`nchat` — client de xat de terminal (WhatsApp, Telegram):

```
$ brew install nchat          # macOS i, compilant,
  Linux
```

Vincula's amb el compte mòbil escanejant un codi QR, igual que faria WhatsApp Web.

Altres que val la pena conéixer, encara que no les use totes cada dia: `btop` o `htop` (monitor de recursos del sistema, visual i en colors), `ranger` o `nnn` (explorador d'arxius per teclat), i `lazygit` (interfície visual per a git, per a qui prefereix vore l'estat d'un cop d'ull sense memoritzar comandos).

Cap d'estes eines és imprescindible. Ningú necessita deixar el navegador del tot. Però val la pena provar-ne alguna, encara que només siga per a descobrir, com jo, que a vegades el problema no era la informació que consumies, sinó tot el que venia empaquetat al seu voltant sense que ho haguares demanat.

Ctrl+Shift+O

Quatre terminals a la vegada, sense obrir quatre finestres: el descobriment que em va ordenar l'escriptori.

Estava enmig de la construcció de l'script d'actes —el projecte que contaré al capítol següent— i tenia obertes, literalment, sis finestres de terminal diferents damunt de la taula: una per a escriure l'script, una altra per a provar-lo, una tercera vigilant el registre d'errors amb `tail -f`, una quarta per si em calia consultar alguna cosa amb `git`... Passava més temps buscant amb `Alt+Tab` quina finestra era quina que escrivint codi de veres.

Li vaig comentar la meua frustració a un company, mig en broma, i em va mirar la pantalla amb cara de qui veu algú que encara talla el pa amb un ganivet de mantega. «Tu no fas servir panells?». No en tenia ni idea de què em parlava.

Eixa vesprada vaig instal·lar **Tilix**, un emulador de terminal que, en lloc d'obrir una finestra nova cada volta que en necessites una, et deixa **partir la mateixa finestra en trossos**, com si retallares un

full de paper en columnes i files, cadascuna amb el seu propi terminal independent dins.

```
$ sudo apt install tilix
```

La primera volta que vaig prémer `Ctrl+Shift+O` —dividir horitzontalment— i vaig vore la finestra partir-se en dos, cadascuna amb el seu propi prompt, vaig sentir la mateixa emoció xicoteta de descobrir el prompt personalitzat al `.bashrc`. Vaig afegir una divisió vertical amb `Ctrl+Shift+E`, i en menys d'un minut tenia els quatre terminals que necessitava, tots a la vista alhora, dins d'una sola finestra, navegables amb `Alt` més una fletxa en lloc de rebuscar entre finestres soltes per la barra de tasques.

El primer dia el vaig fer servir més per jugar que per treballar de veres: partia la pantalla, la tornava a ajuntar, en tornava a fer quatre trossos només per vore com de ràpid podia organitzar-me. Però al cap d'una setmana, treballant de veres en l'script, vaig notar la diferència real: podia escriure una línia a l'editor en un panell, executar-la a l'altre, i vore l'error al tercer, tot sense alçar els ulls de la mateixa

finestra ni perdre mai el fil del que estava fent. La fricció de canviar de context —eixe segon i mig de buscar quina finestra era quina— havia desaparegut sencerament.

Vaig recordar, aleshores, `tmux`, aquell de les eines dels majors, i li vaig preguntar a la IA en què es diferenciaven de veres, perquè em semblaven fer coses molt paregudes. La resposta em va aclarir una confusió que portava dies arrossegant: `tmux` divideix una sessió de terminal *des de dins*, funciona igual en local que per SSH en un servidor remot, i sobreviu encara que es talle la connexió. `Tilix` divideix la *finestra gràfica* de l'escriptori, és exclusiu del meu ordinador local, i no sobreviu si tanque la finestra. No calia triar entre els dos —eren eines per a moments distints: `Tilix` per a organitzar el meu escriptori de cada dia, `tmux` per a quan estava treballant a distància i no podia permetre'm perdre la sessió si es tallava la connexió.

Vaig acabar fent-los servir junts, sense conflicte: `Tilix` obrint els meus panells habituals a l'escriptori, i dins d'un d'ells, quan calia connectar-me al servidor de l'institut, `tmux` fent la seua faena

per a que la connexió no es perdera. Dos eines que semblaven competir i que, mirades bé, ni tan sols jugaven al mateix camp.

Apunt tècnic: `Tilix` i la diferència amb `tmux`

Instal·lació:

```
$ sudo apt install tilix       # Debian/Ubuntu
```

(A macOS, l'equivalent més usat és `iTerm2`, que ja apareixia al capítol 2, i que també permet dividir la finestra en panells. A Windows, `Windows Terminal` ho fa nativament amb `Alt+Maj+-` i `Alt+Maj++`.)

Dreceres bàsiques de `Tilix`:

Drecera	Acció
`Ctrl+Shift+O`	Dividir horitzontalment
`Ctrl+Shift+E`	Dividir verticalment
`Alt` + fletxa	Moure's entre panells
`Ctrl+Shift+W`	Tancar el panell actual
`Ctrl+Shift+N`	Nova pestanya

`Tilix` contra `tmux`, en una frase: `Tilix` organitza l'escriptori local en panells visuals; `tmux` manté sessions vives que sobreviuen a un tall de connexió, tant en local com en remot. No competixen —es complementen. Un bon flux de treball: `Tilix` per a l'estructura del dia a dia, i dins d'un dels seus panells, `tmux` quan cal treballar per SSH a un servidor que no es pot permetre perdre la sessió.

./generar_actes.sh --tots

Un projecte sencer, de principi a fi: el dia que vaig deixar de necessitar l'script d'un altre.

Recordareu, potser, aquell script d'un company que em va donar el meu primer ensurt de `Permission denied`, allà pel capítol vuit. `generar_actes.sh`. Jo només l'havia usat, amb el `chmod +x` just que li faltava. Mai m'havia parat a mirar què hi havia dins.

A finals de curs, eixe mateix company es va jubilar, i amb ell se'n va anar l'únic que sabia mantindre l'script. Al primer claustre de setembre, la cap d'estudis va preguntar, mig desesperada, si algú sabia «arreglar allò de les actes, que ha deixat de funcionar amb el nou format de plantilles». Ningú va dir res durant uns segons llargs. Aleshores, jo mateix —encara em sorprén escriure-ho— vaig alçar la mà.

No tenia ni idea de si seria capaç. Però per primera volta en tot este any, la idea de posar-m'hi no em va fer por. Vaig decidir no arreglar l'script vell —era d'un altre, i el vaig trobar ple de dreceres que ni jo ni ningú més entendria d'ací un any—, sinó ferne un de nou, meu, ben documentat, per a que qui el trobara després no haguera de patir el que jo vaig patir.

El disseny, primer, en paper: llegir una llista d'alumnes d'un arxiu CSV, omplir una plantilla amb les seues notes, i generar un PDF per grup. Li ho vaig plantejar així a la IA, sense donar-li encara cap detall tècnic, només l'objectiu.

L'script, construït a trossos, provant cada peça abans de la següent —una lliçó que m'ha costat un llibre sencer d'aprendre: no escriure-ho tot de colp i esperar que funcione, sinó peça a peça, comprovant cada una:

```
#!/bin/bash
set -e   # atura l'script si qualsevol comanda falla,
  en lloc de continuar a cegues

CSV="alumnes.csv"
PLANTILLA="plantilla_acta.md"
```

```
EIXIDA="actes_generades"

mkdir -p "$EIXIDA"

while IFS=, read -r grup alumne nota; do
    sed "s/{{ALUMNE}}/$alumne/; s/{{NOTA}}/$nota/"
      "$PLANTILLA" \
        > "$EIXIDA/${grup}_${alumne}.md"
done < "$CSV"

echo "Generades $(ls "$EIXIDA" | wc -l) actes a
  $EIXIDA/"
```

El `set -e` de la primera línia el vaig afegir després d'un ensurt xicotet: sense ell, si el CSV tenia una línia mal formatada, l'script continuava com si res, generant actes buides en lloc d'aturar-se i avisar-me. Amb `set -e`, qualsevol error atura tot el procés d'immediat —una xicoteta xarxa de seguretat, com la que vaig trobar a faltar amb `rm -rf` fa tants capítols.

Provar-lo amb un CSV de tres alumnes inventats abans de tocar les dades reals —el mateix hàbit de «prova-ho abans amb poc» que vaig aprendre al capítol del `Ctrl+C`. Va funcionar a la primera, però vaig provar-ho igualment amb un CSV trencat a

posta, només per a comprovar que `set -e` de veres l'aturava. Ho va fer.

Versionar-lo amb `git`, esta volta entenent de veres cada pas, no memoritzant-lo a cegues com feia al capítol de les eines dels majors:

```
$ git init
$ git add generar_actes.sh plantilla_acta.md
$ git commit -m "primera versió funcional"
```

Compartir-lo, per fi, amb qui l'haguera de fer servir després de mi: el vaig pujar a un repositori del centre, i vaig deixar-lo executable amb els permisos justos —`chmod 755`, el del dia del `Permission denied`, ni un permís de més— perquè qualsevol company el poguera córrer sense necessitat de fer-se root per a res.

Quan la cap d'estudis el va provar, dues setmanes després, i les actes van eixir correctes a la primera, no va sentir-ho com un miracle. El vaig sentir jo, per dins, però no calia dir-ho en veu alta. Havia agafat un problema real, l'havia partit en trossos manejables, havia usat, sense haver-los de buscar un per un, quasi tots els comandos i conceptes que este llibre porta explicant des del primer capítol

—permisos, `git`, redirecció d'errors, provar abans d'executar—, i n'havia eixit alguna cosa que funcionava de veres, per a algú més que jo mateix.

Vaig pensar, tancant l'ordinador eixa nit, que era la primera volta que el terminal deixava de ser, per a mi, un lloc on em passaven coses. Havia passat a ser un lloc des d'on jo feia coses. La diferència, tot i semblar xicoteta, ho és tot.

Apunt tècnic: idees clau d'un projecte de terminal ben construït

`set -e` atura un script bash a la primera comanda que falle, en lloc de continuar executant la resta amb dades possiblement corruptes. Imprescindible en qualsevol script que no siga d'un sol ús.

Provar amb dades falses abans de tocar les reals —un CSV inventat, un directori de prova— és l'equivalent, en un projecte sencer, del «prova-ho primer sense l'acció destructiva» del dia del `find /`.

Estructura mínima recomanada per a qualsevol script que altres faran servir: 1. Un comentari a dalt explicant què fa i com s'usa. 2. Comprovacions

bàsiques (existeix l'arxiu d'entrada? té permisos?) abans d'actuar. 3. Missatges clars, per pantalla o a un registre, de què ha passat.

Compartir-lo amb permisos correctes, ni de més ni de menys:

```
$ chmod 755 generar_actes.sh   # executable per a tots,
  editable només pel propietari
```

Versionar-lo amb `git` des del primer dia, encara que siga un projecte xicotet i personal —el dia que algú altre l'haja de tocar, o que tu mateix necessites recordar per què vas fer un canvi, t'ho agrairàs.

Cap d'estes idees és nova en este llibre. El que canvia, en un projecte com este, és que totes es fan servir juntes, encadenades, per un objectiu real —i eixa és, potser, la millor manera de saber si de veres has aprés alguna cosa: no si recordes cada comanda per separat, sinó si saps combinar-les quan una faena de veres t'ho demana.

tailscale up

El dia que vaig descobrir que la meua terminal cabia a la butxaca.

Estava a la sala d'espera del dentista, amb eixa mescla particular d'avorriment i nervis que sempre em fa mirar el mòbil més del compte, quan em va vibrar un missatge de la cap d'estudis: «l'script de les actes torna a fallar, ho necessitem per a la reunió d'esta vesprada». L'script del capítol anterior. El meu.

El primer impuls, l'antic, va ser una punxada d'angoixa —«no arribaré a casa fins a les set»— seguida d'una resignació ràpida: ja ho miraria a la nit. Però eixe mateix matí havia llegit, sense parar-hi massa atenció, que un company de l'institut connectava al servidor de manteniment «des del mòbil, com si estiguera davant de l'ordinador». Ho havia arxivat mentalment com una d'eixes coses de gent més moderna que jo. Ara, assentat en una cadira de plàstic incòmoda, em vaig preguntar si valia la pena preguntar-li a la IA si allò era de veres possible.

Ho era. Es diu **Tailscale**, i el que fa és, explicat amb les meues paraules d'ara: crea una xarxa privada entre tots els teus dispositius —el portàtil de casa, el mòbil, qualsevol ordinador que hi afiges—, com si estigueren tots endollats al mateix cable, encara que un estiga a casa i l'altre a la sala d'espera d'un dentista a tres quilòmetres. Una volta instal·lat i connectat als dos costats, el mòbil pot parlar amb el portàtil de casa amb la mateixa naturalitat amb què parla amb qualsevol pàgina web.

Vaig instal·lar-lo al mòbil en dos minuts, seguint els passos que la IA em donava un per un —descarregar l'app, iniciar sessió amb el mateix compte que ja tenia al portàtil, prémer «Connectar»—, i després vaig obrir una altra app que no coneixia, **Termux**, un terminal de veres, complet, corrent dins del telèfon.

```
$ ssh samuel@portatil-casa
```

El cor se m'ha accelerat menys en moltes muntanyes russes que en eixe segon d'espera. I aleshores, el prompt. El meu prompt. Verd, amb la carpeta actual, exactament el mateix que havia decorat quan

em vaig fer la casa al `.bashrc`, ara apareixent a la pantalla xicoteta d'un mòbil a la sala d'espera d'un dentista.

```
samuel@portatil-casa ~/institut $
```

Estava, literalment, dins del meu propi ordinador, des de tres quilòmetres de distància, sense fils, sense res més que un mòbil que fins eixe matí només usava per a Whatsapp i per a matar l'estona.

Vaig anar directament a `~/backup.log`, per costum ja, i ahí estava la pista: el disc extern no estava connectat —me l'havia deixat desendollat la nit abans sense pensar-m'ho—, i sense ell, l'script que generava les actes fallava perquè escrivia una còpia prèvia al mateix disc abans de continuar. No podia endollar-lo des d'ahí, per descomptat. Però sí que podia, amb la IA ajudant-me a escriure la comanda des del mòbil amb els mateixos dits amb què normalment escric missatges, modificar l'script perquè la còpia prèvia fóra opcional si el disc no hi era, en lloc d'aturar tot el procés:

```
$ nano generar_actes.sh
```

Vaig editar tres línies, vaig guardar, vaig executar l'script sencer des d'ahí mateix, i vaig vore aparéixer «Generades 24 actes» a la pantalleta del mòbil. Vaig respondre a la cap d'estudis, encara amb el dentista cridant-me pel nom des de la porta: «ja està arreglat, les tens a la carpeta de sempre».

Vaig entrar a la consulta amb un somriure ridícul per a algú que estava a punt que li tocaren una carie. No havia sigut la solució tècnica el que m'havia emocionat —no era, ni de bon tros, la comanda més complicada d'este llibre. Havia sigut la sensació, nova de trinca, que la meua «casa» del `.bashrc` ja no estava lligada a un lloc físic. Podia dur-la damunt, a la butxaca, i entrar-hi des de qualsevol lloc amb la mateixa familiaritat de sempre. El terminal havia deixat de ser una habitació concreta d'una casa concreta. S'havia convertit en un lloc que anava on anava jo.

Apunt tècnic: connectar-se al teu ordinador des del mòbil, de veres

Tailscale crea una xarxa privada virtual (*VPN de malla*) entre tots els dispositius on l'instal·les i inicies sessió amb el mateix compte. No cal obrir ports al router ni configurar res complicat: els dispositius es troben entre ells sols, encara que estiguen en xarxes diferents.

Instal·lació, als dos costats de la connexió:

- Al portàtil o ordinador de sobretaula (Linux, macOS o Windows): es descarrega des de `tailscale.com`, s'instal·la, i s'activa amb:

  ```
  $ sudo tailscale up
  ```

- Al mòbil (Android o iPhone): es descarrega l'app de Tailscale des de la botiga corresponent, s'inicia sessió amb el mateix compte.

Per a tindre un terminal de veres al mòbil, no n'hi ha prou amb Tailscale sol —cal una app que òbriga un terminal:

- **Android:** `Termux`, un terminal complet i gratuït, amb el seu propi gestor de paquets (`pkg install openssh`).
- **iPhone:** apps com `Termius` o `Blink Shell` fan de client SSH complet.

Una cosa que em va sorprendre gratament: `Termux` és un Linux de veres, no una versió retallada, així que `tmux` —el de les eines dels majors— hi funciona exactament igual que al portàtil (`pkg install tmux`). No cal buscar cap alternativa especial per a Android. El mateix `Termux` permet també obrir diverses sessions a la vegada des del calaix lateral de l'app, i hi ha qui prefereix `screen`, el predecessor de `tmux`, o `JuiceSSH`, una app que no és cap multiplexor sinó un gestor de connexions SSH guardades —útil, però no fa la mateixa faena.

Un pas més enllà: Debian o Ubuntu dins de `Termux`

`Termux` té el seu propi sistema de paquets (`pkg`), paregut a `apt` però no idèntic, i algunes eines complexes —certes eines d'IA en línia de comandos, sobretot— esperen trobar-se en una distribució estàndard i fallen o donen errors estranys

sense un motiu clar. Quan açò passa, la solució és instal·lar una distribució completa a dins, amb `proot-distro`:

```
$ pkg install proot-distro
$ proot-distro install debian
$ proot-distro login debian
```

Avantatges: un `apt` de veres, compatibilitat quasi total amb qualsevol programa pensat per a Debian o Ubuntu, i la tranquil·litat de seguir qualsevol tutorial escrit per a un servidor normal sense haver-lo d'adaptar. `tmux`, `vim`, `git`, o una eina d'IA en línia de comandos, s'instal·len ahí dins exactament igual que en un ordinador de veres.

Inconvenients: `proot-distro` no és una màquina virtual real, sinó una capa de traducció que emula un sistema complet damunt del `Termux` original — i això té un cost: va més lent, sobretot en operacions de disc, ocupa més espai, i consumeix més bateria. Alguns programes que depenen molt de `systemd` o d'accés directe al maquinari no funcionen, perquè seguix sense haver-hi root de veres. I per a un simple `ssh` o un `tmux` senzill, com els d'este capítol, és afegir una capa de complexitat que no cal.

La meua regla, després de provar-ho: `Termux` a soles per a connectar-me i fer coses senzilles; `proot-distro` només quan una eina concreta ho exigix de veres i no hi ha alternativa més senzilla.

Connectar-se, una volta els dos dispositius estan a la mateixa xarxa Tailscale:

```
$ ssh usuari@nom-del-dispositiu
```

Tailscale assigna un nom fàcil de recordar a cada dispositiu (el nom de l'ordinador), en lloc d'haver de memoritzar cap adreça IP.

Un avís important, no menor: açò obri l'ordinador de casa a connexions remotes. Val la pena assegurar-se que la contrasenya de l'usuari és forta, o millor encara, configurar l'accés per clau SSH en lloc de contrasenya —i mantindre el sistema actualitzat. La comoditat de portar el terminal a la butxaca no hauria de vindre a costa de deixar la porta de casa mal tancada.

:q!

El ritual d'iniciació que ningú s'escapa: quedar-se atrapat dins de vim.

Necessitava editar un arxiu de configuració al servidor de l'institut, connectat per SSH, i vaig teclejar el que la IA em va donar sense parar-m'hi a mirar:

```
$ vim configuracio.conf
```

Es va obrir una pantalla plena de text, amb virgulilles (~) omplint totes les línies buides de sota. Vaig intentar escriure, com faria en qualsevol editor de tota la vida. Res. Les lletres que teclejava no apareixien on jo esperava —en compte d'escriure's, semblaven activar coses: es movia el cursor, desapareixien línies senceres, un so d'error curt i sec cada dos per tres.

Vaig sentir, per un moment breu i genuí, el mateix pànic del dia zero d'este llibre. Vaig intentar tancar-lo com tanque tot: `Ctrl+C`. No va passar res. `Esc`. Tampoc. Vaig arribar a plantejar-me tancar la finestra sencera del terminal i deixar-ho estar.

Vaig obrir el mòbil —gràcies, per cert, al descobriment del capítol anterior— i li vaig preguntar a la IA, amb la vergonya de qui sap que està a punt de fer la pregunta més típica d'internet: «com s'ix de vim?».

La resposta em va fer riure en veu alta, ahí mateix, a soles: «No et preocupes, li passa a tothom la primera volta —hi ha fins i tot una broma vella entre programadors sobre gent atrapada a vim per sempre». `Esc`, per a eixir del mode d'inserció si hi era, i després:

```
:q!
```

Dos punts, una q, un signe d'exclamació. Vaig prémer Intro amb la mateixa incertesa amb què s'obri una porta que no se sap on porta. I vaig eixir. Sa i estalvi, sense haver guardat res, exactament com volia.

La IA em va explicar, després, per què m'havia costat tant una cosa tan senzilla en aparença: `vim` no funciona com els editors que conec de tota la vida. Té **modes**. Un mode «normal», on cada tecla és una comanda —`h`, `j`, `k`, `l` per a moure's, `dd` per a

esborrar una línia, `x` per a esborrar un caràcter— i no escriu res per si sola. I un mode «inserció», que s'activa prement `i`, on per fi les tecles escriuen text com jo esperava des del principi. Jo havia estat en mode normal tota l'estona, prement lletres que `vim` interpretava com a comandos, no com a text.

Vaig decidir, eixa mateixa nit, buscar una alternativa més amable per al dia a dia, i la IA em va parlar de `nano`, molt més senzill —sense modes, sense sorpreses, amb les dreceres escrites a la part de baix de la pantalla perquè no calga memoritzarles. El vaig fer servir des d'aleshores per a quasi tot, incloent-hi, sense que el lector se n'haja adonat fins ara, l'edició de l'script del capítol anterior, allà a la sala d'espera del dentista.

(Li vaig preguntar també a la IA si hi havia un tercer editor que valguera la pena conéixer, i em va parlar d'`emacs` amb un to que no vaig saber interpretar del tot —«és potentíssim, i els seus usuaris el defensen amb una passió gairebé religiosa»— fins que vaig descobrir que `vim` contra `emacs` és, des de fa quaranta anys, una de les batalletes més velles i més sagrades d'internet. Vaig decidir, per esta volta, no

obrir un tercer front. Ja tenia prou amb no quedar-me atrapat en el primer.)

Però no vaig deixar `vim` del tot de banda. Vaig aprendre, a poc a poc, les quatre comandes bàsiques —entrar en mode inserció, eixir-ne, guardar, eixir sense guardar—, prou per a no quedar-me mai més atrapat. I vaig entendre, amb el temps, per què tanta gent amb dècades d'experiència l'estima: una volta domines els seus modes, no cal alçar mai les mans del teclat cap al ratolí, ni tan sols cap a les fletxes. Per a mi, encara, és un convidat que trate amb respecte i una miqueta de cautela. Per a d'altres, és casa seua des de fa trenta anys. Cadascú troba la seua.

Apunt tècnic: manual xicotet de `nano` i `vim`

`nano` — l'editor recomanat per a començar. S'obri i s'edita com qualsevol programa normal, sense modes:

```
$ nano arxiu.txt
```

Les dreceres més útils, sempre visibles a la part de baix de la pantalla (^ vol dir `Ctrl`):

Drecera	Acció
`Ctrl+O`	Guardar (*Write Out*)
`Ctrl+X`	Eixir
`Ctrl+K`	Tallar la línia actual
`Ctrl+U`	Enganxar
`Ctrl+W`	Buscar text

`vim` — més potent, més ràpid una volta el domines, però amb una corba d'aprenentatge real. Funciona per **modes**:

```
$ vim arxiu.txt
```

Tecla / comanda	Acció
`i`	Entrar en mode inserció (per fi, escriure text)
`Esc`	Tornar al mode normal
`:w`	Guardar
`:q`	Eixir (si no hi ha canvis sense guardar)

Tecla / comanda	Acció
`:wq`	Guardar i eixir
`:q!`	Eixir **sense** guardar, forçant-ho
`dd`	Esborrar la línia actual (en mode normal)
`/paraula`	Buscar «paraula» (en mode normal)

La regla d'or per a no quedar-se mai atrapat: prem `Esc` primer —per a assegurar-te que estàs en mode normal—, i després `:q!` si vols eixir sense complicacions. Amb això sol, ja no cal témer obrir `vim` per accident mai més.

Els dos editors vénen instal·lats, o quasi, a totes les distribucions de Linux i a macOS. En Windows, `nano` es pot instal·lar amb `winget`, i `vim` sol vindre disponible dins de WSL.

git worktree

Quan tens més d'una IA treballant en el mateix projecte: coordinació, branques i la temptació de deixar-les totes soltes.

Vaig tardar mesos a descobrir que el problema no era triar una IA.

El problema era que, de sobte, en tenia massa.

Al principi tot era senzill: una eina de terminal, una conversa, una pregunta. Jo escrivia, la IA responia, el terminal executava. El triangle era manejable, quasi íntim. Però un dia, treballant en un projecte menut —un script per ordenar materials del curs i generar-ne una versió neta per a publicar—, em vaig trobar amb tres finestres obertes: una amb Codex revisant una funció, una altra amb OpenCode proposant una refactorització, i una tercera amb una conversa de Claude on jo li demanava una explicació més pedagògica del mateix embolic.

Durant deu minuts em vaig sentir moderníssim.

Al quart d'hora, ja no sabia qui havia dit què.

Codex volia canviar el nom de dos arxius. OpenCode havia detectat una duplicació i proposava moure-la a una funció auxiliar. Claude, que no veia els arxius directament sinó el fragment que jo li havia pegat, suggeria una solució que encaixava amb una versió anterior del codi. Totes les respostes eren raonables per separat. Juntes, eren una reunió sense ordre del dia.

Vaig estar a punt de cometre l'error gros: deixar que dos agents tocaren la mateixa carpeta alhora.

No ho vaig fer per prudència, sinó perquè `git status` em va salvar abans.

```
$ git status
```

Hi havia canvis pendents en tres arxius. Alguns meus, alguns de l'agent, alguns que ja no recordava de qui eren. Vaig sentir una incomoditat coneguda: la mateixa que quan una classe sencera parla alhora i totes les veus tenen, individualment, alguna cosa a dir.

La primera regla em va caure damunt amb una claredat molt simple: **un projecte no necessita mol-**

tes IA treballant alhora; necessita un humà que les coordine.

Diversos agents poden ser útils. Molt. Però no com una banda de música entrant cadascú per un compàs diferent. Un pot revisar el codi. Un altre pot escriure documentació. Un tercer pot proposar alternatives. Però si tots tenen permís per a editar el mateix arxiu en la mateixa carpeta, el que tens no és productivitat: és una baralla educada.

Eixa nit vaig demanar-li a la IA una manera neta de provar alternatives sense embrutar el projecte principal. Em va parlar de branques, que jo ja coneixia una miqueta, i després d'una eina que fins aleshores m'havia passat desapercebuda: `git worktree`.

Una branca et permet tindre una línia de treball separada. Un `worktree` va un pas més enllà: et permet tindre eixa branca oberta en una altra carpeta física, al costat de la principal. Com si fera una habitació nova per a provar una reforma sense moure els mobles de casa.

La idea em va paréixer elegantíssimament pràctica.

```
$ git worktree add -b docs-agent ../projecte-docs
```

Ara podia tindre una carpeta per a una tasca concreta. En una, el projecte principal. En l'altra, una branca on un agent podia treballar la documentació sense tocar el que jo estava provant en la branca principal.

No vaig convertir-me de colp en un enginyer d'aquells que dibuixen diagrames de branques en una pissarra. Però vaig aprendre una regla domèstica:

Si dos agents han de treballar alhora, que no ho facen damunt de la mateixa taula.

Per a tasques menudes, em basta una norma encara més senzilla:

Primer agent: diagnostica, no edites.

Segon agent: proposa alternativa, no edites.

Només un agent, al final, aplica els canvis.

I jo mire el `diff`.

```
$ git diff
```

Açò últim és el que no es pot delegar. La IA pot comparar opcions, detectar errors, escriure codi, explicar-me una eixida del terminal i fer-me sentir

que la faena avança a una velocitat nova. Però decidir quina proposta entra al projecte continua sent faena meua. Si no, no estic usant diversos agents; estic deixant que diversos agents m'usen a mi com a pont confús entre respostes incompatibles.

Vaig acabar aquella nit amb una llibreta de normes molt més útil que qualsevol rànquing d'eines:

Una IA per tasca.

Una carpeta neta per canvi important.

Cap agent editant si no hi ha commit previ.

Cap resposta acceptada sense `git diff`.

Cap discussió llarga sense resum escrit.

I, sobretot, no confondre quantitat de veus amb qualitat de criteri.

Les IA poden treballar al mateix projecte. Algunes poden fer-ho al mateix temps. Però el projecte no és d'elles. El projecte és el lloc on jo decidisc quina conversa deixa rastre.

Apunt tècnic: diversos agents sense convertir el projecte en un embolic

La manera més simple d'usar més d'una IA és separar **funcions**, no soltar-les totes sobre el mateix codi:

- una IA pot diagnosticar un error;
- una altra pot proposar una alternativa;
- una altra pot revisar el `diff`;
- només una hauria d'editar en cada moment, llevat que treballen en branques o carpetes separades.

Abans de deixar editar un agent:

```
$ git status
$ git add .
$ git commit -m "punt estable abans de provar agent"
```

Per provar una alternativa en una branca:

```
$ git switch -c prova-agent
```

Per tindre una branca en una altra carpeta amb `worktree`:

```
$ git worktree add -b prova-agent ../projecte-prova
```

Això crea una carpeta germana (`../projecte-prova`) associada a la branca `prova-agent`. Pots obrir-la amb un altre terminal, un altre VS Code o un altre agent, sense tocar la carpeta principal.

Per veure els worktrees oberts:

```
$ git worktree list
```

Per eliminar-ne un quan ja no el necessites:

```
$ git worktree remove ../projecte-prova
```

No cal usar `worktree` per a tot. Per a molts casos, una branca normal i disciplina amb els commits és suficient. Però quan vols comparar dos enfocaments, o deixar un agent treballant una documentació mentre tu proves una altra cosa, és una ferramenta molt neta.

La regla de fons és esta: **els agents poden proposar i escriure, però el control de versions és el teu semàfor.** Si no hi ha commit, diff i decisió humana, no tens col·laboració amb IA; tens soroll amb permisos d'escriptura.

pass insert -m

El dia que vaig decidir no mirar mai més una contrasenya per pantalla, i per això mateix vaig ser més rigorós que mai.

La wiki interna del cicle d'informàtica portava anys viva sense que ningú li fera massa cas: apunts de pràctiques, procediments de manteniment de l'aula, l'esquema de xarxa que ningú recordava de memòria. Entrava amb el mateix usuari del LDAP del centre, eixe directori central que ningú entén del tot però que tots depenem d'ell en silenci. Fins que un matí de setembre va deixar de funcionar. Cap alumne, cap professor, ningú podia entrar-hi. El LDAP estava trencat de veres, i arreglar-ho no depenia de mi.

La wiki sí.

Vaig decidir, amb la IA al costat, desactivar temporalment eixe login extern i tornar a l'autenticació local, la de sempre, mentre algú més arreglava el problema de fons. Calia resetejar la contrasenya

d'administrador local, una que ningú havia tocat en anys perquè tot el món entrava per LDAP.

```
$ ./bin/passpw.php admin
```

La comanda em va donar una contrasenya nova a l'instant: llarga, amb símbols, sense cap paraula reconeixible. La vaig copiar. I ací em vaig fer la pregunta ridícula, mirant la pantalla: i ara açò on ho guarde?

Perquè no ho guardava enlloc, de veres. Portava mesos amb el mètode que qualsevol dels meus alumnes reconeixeria com a mal hàbit si l'haguera vist en un treball seu: un arxiu `notes_varies.txt` amb contrasenyes soltes entre apunts de classe, algun post-it perdut, un WhatsApp que m'havia enviat a mi mateix «per si de cas» i que ara no localitzava. La mateixa carpeta `notes` que em va ensenyar, fa un any, què vol dir no tindre paperera, tornava a fer de la seua.

Li ho vaig confessar a la IA, quasi avergonyit. Em va parlar d'una eina que jo no coneixia: `pass`, el gestor de contrasenyes de la línia de comandos, vell, sense colors ni interfície, basat en dos coses que ja

m'eren familiars per separat —GPG per a xifrar, `git` per a guardar l'historial.

La instal·lació va ser senzilla. La decisió que vaig haver de prendre després, no tant: `pass` xifra cada contrasenya amb una clau GPG, i eixa clau normalment es protegix amb una frase de pas — sense ella, ningú, ni tu, pot desxifrar res. Semblava l'opció òbvia. Però jo no volia que ningú, tampoc jo mateix a les tres de la matinada mig adormit, haguera de teclejar una frase secreta cada volta que la IA necessitara consultar o guardar una credencial en una sessió futura, mentre jo dormia o feia classe. Vaig fer, conscientment, l'elecció contrària: una clau GPG dedicada a `pass`, sense frase de pas.

Açò no vol dir renunciar a tota seguretat. Vol dir traslladar-la de lloc. Ja no depén d'un secret que algú haja de recordar i escriure —depén, com al dia del `Permission denied`, dels permisos del sistema d'arxius: qui pot llegir la meua carpeta d'usuari, i prou més. La protecció ja no és un secret al meu cap. És una porta que continua tancada, però que ja no necessita clau de gir cada volta.

Vaig organitzar les entrades per projecte, per costum de professor que ordena carpetes: `institut/dokuwiki-admin`, `institut/wifi-aula`, `casa/router`. La contrasenya, sempre a la primera línia; davall, el que calguera recordar-ne —usuari, URL, alguna nota— perquè mesos després no tornara a preguntar-me on havia guardat açò.

```
$ pass insert -m institut/dokuwiki-admin
```

I ací naix la norma que no negocie amb ningú, ni tan sols amb mi mateix quan tinc pressa: la contrasenya no apareix mai, en cap circumstància, impresa a la conversa amb la IA. Ho vaig aprendre de la manera tonta, com quasi tot en este llibre. Setmanes abans, comprovant que `pass` havia guardat bé una contrasenya de `sudo` d'un altre servidor, vaig fer `pass show` sense pensar-m'ho, per veure «si havia quedat bé guardada». Havia quedat perfectament guardada. I també havia quedat, en clar, a la pantalla de la conversa, en un historial que jo mateix podia rellegir després. Ningú més la va vore, per sort, però la lliçó em va caure com un got d'aigua freda: acabava d'exposar, en el lloc pensat per a no

exposar-la mai, la mateixa contrasenya que volia protegir.

Des d'aleshores, si la IA necessita una contrasenya, no la demana ni la mostra: la llig dins d'una variable i la passa directament a qui la necessita, sense que passe mai pel text del xat. I quan és ella qui genera una credencial nova, el procediment és sempre el mateix: generar-la a l'atzar, guardar-la amb `pass insert -m` junt amb usuari i URL, esborrar qualsevol arxiu temporal usat per a passar-la d'una màquina a una altra —les intermèdies incloses, si ha calgut saltar per SSH—, i avisar-me només de la ruta on ha quedat guardada. Mai la contrasenya solta.

La primera volta que ho vaig pensar amb calma, em va paréixer una contradicció: com pot ser més segur no mirar mai la clau de la meua pròpia casa? Però és exactament el contrari del dia que vaig instal·lar aquell primer script sense llegir-lo, confiant a cegues per pura pressa. Ací el que és cec no és la pressa: és el disseny. No mire la contrasenya perquè decidir no mirar-la és, precisament, la manera de garantir que no en quede cap rastre enlloc on no calga.

Apunt tècnic: `pass`, contrasenyes que no ixen mai a pantalla

Instal·lació:

- **Linux (Debian/Ubuntu):** `sudo apt install pass gnupg`
- **macOS (amb Homebrew):** `brew install pass gnupg`
- **Windows:** `pass` no és natiu, però funciona igual dins de WSL instal·lant-lo com en Linux.

Preparar la clau i el magatzem, la primera vegada:

```
$ gpg --full-generate-key
$ pass init <identificador-de-la-clau>
```

Al crear la clau, GPG demana una frase de pas. Deixar-la buida és una decisió, no un descuit: sense frase, `pass` llig i escriu credencials sense demanar res a ningú, però tota la protecció passa a dependre dels permisos de `~/.password-store`. Amb frase de pas, guanyes un secret criptogràfic real, però perds l'automatització: caldria teclejar-la cada volta.

Guardar una entrada, amb metadades a més de la contrasenya:

```
$ pass insert -m institut/dokuwiki-admin
```

La primera línia que escrius és la contrasenya. Les següents, el que vulgues recordar —usuari, URL, notes— fins acabar amb `Ctrl+D`.

Generar una contrasenya aleatòria i guardar-la directament, sense haver-la de pensar:

```
$ pass generate institut/wifi-aula 20
```

Consultar una entrada, quan qui la mira és un humà i prou:

```
$ pass show institut/dokuwiki-admin
```

La regla que de veres importa: eixe `pass show` és segur davant del meu propi terminal, on sóc l'únic que el veu. El que no ha d'ocórrer mai és que eixe text passe per una conversa escrita amb una IA, un registre o una captura de pantalla. Dins d'un procés automatitzat, la tècnica és llegir la contrasenya dins d'una variable i passar-la per l'entrada estàndard del programa que la necessita:

```
$ contrasenya=$(pass show institut/dokuwiki-admin |
  head -n1)
$ comanda-que-la-necessita <<< "$contrasenya"
$ unset contrasenya
```

Alguns programes ja preveuen este ús —`docker login` accepta `--password-stdin` en compte de rebre-la com a paràmetre solt—, pel mateix principi: la contrasenya viatja per una canonada, no per un text que algú puga rellegir després. Mai com a paràmetre solt d'una comanda: qualsevol cosa escrita així queda visible per a qui mire els processos en marxa, i sovint guardada a l'historial de la shell.

`~/.password-store` és, per davall, una carpeta normal amb un arxiu `.gpg` per entrada, amb la mateixa jerarquia que li done jo. Res impedix versionar-la amb `git`, com qualsevol altre projecte d'este llibre, amb la mateixa prudència de no confiar mai un secret a un repositori que no siga estrictament privat.

tmux attach

Aquell va nàixer d'una pèrdua; este, d'una alegria que encara em dura.

Hui no he aprés cap comanda nova. No ha fallat res, no he trencat res, no he instal·lat res. I és, probablement, el dia que més m'ha impressionat el terminal des que vaig començar este diari.

M'explique, perquè ni jo ho entenia mentre em passava.

Fa unes setmanes vaig ressuscitar un ordinador de sobretaula vell que teníem criant pols, el vaig deixar sempre encés en un racó de l'estudi, i el vaig afegir a la xarxa de Tailscale. La idea era modesta: que la faena llarga —còpies, scripts, el que fóra— correguera ahí i no al portàtil. Dins, com sempre, una sessió de `tmux`, que a hores d'ara ja òbric per costum, sense pensar-hi, com qui es posa el cinturó abans d'arrancar el cotxe.

Este matí, a la cuina, amb el portàtil i el primer café, m'hi he connectat i he escrit el de sempre:

```
$ tmux attach
```

I ahí estava la faena d'anit: la IA a mig explicar-me una reorganització dels materials del curs, amb una resposta sencera que m'havia quedat per llegir. L'he llegida, li he contestat, he deixat una altra resposta seua a mitges, i me n'he anat a l'institut.

A l'hora del pati, fent cua a la fotocopiadora, he tret el mòbil. Termux, `ssh`, `tmux attach`. La resposta que havia deixat a mitges a la cuina estava ahí, en el mateix punt exacte, esperant-me a la pantalleta com si la cuina i el passadís de l'institut foren la mateixa habitació. He escrit dos línies amb els polzes, li he encarregat que continuara ella per la seua banda, i he guardat el mòbil quan ha sonat el timbre.

I esta vesprada, a l'estudi, assegut per fi davant de l'ordinador de sobretaula mateix, ho he tornat a trobar tot: el matí de la cuina, el pati, la faena feta mentre jo no mirava. La mateixa conversa. El mateix prompt. La mateixa línia a mig escriure.

Tres llocs, tres aparells, tres pantalles de tres grandàries distintes. Una sola sessió.

Ha sigut en eixe tercer salt quan ho he vist de veres. La sessió no estava en cap dels tres aparells. No viatjava amb mi, no es copiava d'un lloc a l'altre,

no calia «passar-me res» del portàtil al mòbil. Estava —està ara mateix— a l'ordinador de l'estudi, i el portàtil, el mòbil i la pròpia pantalla de l'estudi no eren més que finestres distintes obertes sobre la mateixa habitació. Jo no m'enduia la faena a cap lloc; només treia el cap per una finestra o per una altra, segons on m'agafara el dia.

I ací ve la part que em fa una miqueta de vergonya, i que per això mateix apunte: açò no és cap descobriment. Fa mesos que use `tmux`. Va entrar en este diari el dia de les eines dels majors, quan vaig aprendre que una sessió sobreviu a un tall de connexió, i des d'aleshores l'he usat exactament per a això, com una assegurança: si cau el wifi, la faena no cau. Fins i tot sabia, perquè ho havia escrit ací, que Termux el fa anar al mòbil igual que al portàtil. Ho sabia tot. Ho usava tot. I no havia entés quasi res.

Perquè una assegurança contra talls i un lloc són dos idees molt distintes. La primera diu: «si passa una desgràcia, no perdràs la faena». La segona diu: «la teua taula de treball ja no està en cap màquina; està en un lloc, i totes les teues màquines són portes d'eixe lloc». Entre saber la primera i sentir la segona

hi ha, pel que es veu, mesos d'usar una eina cada dia sense haver-la entés del tot.

Encara sóc molt nou en açò. Hui n'he tingut la prova més neta: es pot fer servir una ferramenta durant mesos, correctament i tot, i que un dimarts qualsevol, fent cua a una fotocopiadora, et revele de colp per a què existia. No descarte que d'ací a un temps rellija este capítol i em faça tendresa tant d'entusiasme per una cosa que qualsevol informàtic considera evident. M'és igual. Ara mateix, mentre escric estes línies des del llit, al mòbil, sé que a l'estudi, a fosques, hi ha una conversa encesa que m'espera. Demà tornaré a traure el cap. Des d'on siga.

Apunt tècnic: on viu una sessió de `tmux` (i com tindre'n més d'una)

La clau de tot el capítol és una sola idea: **una sessió de `tmux` viu a la màquina on es va llançar**, no al dispositiu des del qual t'hi connectes. Si la llances a un ordinador que està sempre encés i accessible —el de casa, via Tailscale, en el meu cas—, el portàtil i

el mòbil no «tenen» la sessió: només s'hi connecten. Per això dóna exactament igual des d'on entres.

Quan comences a viure així, prompte en vols més d'una —una per al curs, una per a un projecte llarg—, i convé donar-los nom en crear-les:

```
$ tmux new -s curs
$ tmux new -s llibre
```

Per a vore quines sessions hi ha vives en una màquina:

```
$ tmux ls
```

Per a tornar exactament a la que toca:

```
$ tmux attach -t curs
```

`tmux attach` a seques, sense `-t`, et connecta a l'última sessió activa —suficient mentre només en tens una.

Una curiositat que descol·loca la primera volta: si dos dispositius fan `attach` a la mateixa sessió alhora, tots dos veuen el mateix en temps real. No són dos còpies: és un espill. El que s'escriu des d'un apareix a l'altre, lletra a lletra.

No cal res més. Un ordinador sempre encés, Tailscale per a arribar-hi des de qualsevol lloc, i una sessió de `tmux` amb nom: és el més paregut que conec a un escriptori que no es tanca mai.

2>&1

La tesi del llibre: IA i terminal parlen el mateix idioma (text), i jo he aprés a fer de tercer en la conversa.

Estava depurant —«depurant», ara ja em sona natural dir-ho— un script que havia escrit per a automatitzar l'exportació de notes del curs, i em donava un error estrany que no acabava d'entendre. Li vaig ensenyar a la IA el que veia per pantalla, copiant-ho a mà, i la resposta va ser: «pega'm ací la ixida sencera, incloent els errors». Vaig intentar-ho, i em vaig trobar amb una cosa curiosa: quan redirigia l'eixida a un arxiu per a poder copiar-la, els missatges d'error no hi apareixien. Es quedaven fora, com si el programa parlara per dos boques distintes i jo només n'estiguera escoltant una.

```
$ ./exporta_notes.sh > sortida.txt
```

L'arxiu `sortida.txt` tenia el text normal del programa. Però els errors —«no es pot obrir l'arxiu de configuració», deia la pantalla— no hi eren enlloc. Li ho vaig preguntar a la IA, i em va explicar una cosa que em va fer riure de mi mateix, per la senzillesa

de la resposta: cada programa, en el terminal, té de fet dos canals de eixida distints. Un per a la resposta normal (`stdout`), i un altre, separat, específicament per als errors (`stderr`). Quan redirigia amb >, només capturava el primer. Els errors, per defecte, continuaven eixint per pantalla, com un actor secundari que insistix a parlar encara que li hages tancat el micròfon principal.

```
$ ./exporta_notes.sh > sortida.txt 2>&1
```

Eixe estrany `2>&1` —que la primera volta que el vaig vore em va semblar un jeroglífic— resulta que és, llegit peça a peça, ben senzill: «canal 2 (els errors), redirigix-lo cap on va el canal 1 (la eixida normal)». Ara sí, tot junt, tot a l'arxiu, tot a la vista.

I ahí, escrivint eixa línia estranya, se'm va acudir una imatge que m'ha acompanyat des d'aleshores, i que crec que és, en el fons, de què va este llibre sencer: el terminal i la IA parlen exactament el mateix idioma. Text. Res més. El terminal escup text i espera rebre text. La IA llig text i escriu text. No hi ha cap traducció de per mig, cap intèrpret extern que puga perdre matisos pel camí. Quan li pegue a

la IA l'eixida d'un error, no li estic descrivint el problema —li estic passant el problema mateix, literal, exacte, amb els mateixos caràcters que veuria jo.

Açò canvia completament el meu paper en tot este afer. Durant mesos m'havia pensat que jo era, simplement, l'alumne, i la IA la mestra. Però eixa nit, davant eixe `2>&1`, vaig entendre que la relació era una altra: jo era, més aïna, el traductor de necessitats. La IA no sap què vull aconseguir amb l'exportació de notes, ni per què m'importa. Jo sí. El terminal no sap explicar per què un error m'afecta a mi personalment, en la meua faena real, amb els meus alumnes reals. Jo sóc qui posa el context, qui trau la pregunta correcta de dins d'un embolic confús, qui reconeix quan una resposta tècnicament perfecta no encaixa amb el que de veres necessite.

En eixe triangle —jo, la IA, el terminal— cadascú fa una cosa que els altres dos no poden fer sols. El terminal executa amb una precisió absoluta i sense cap comprensió del perquè. La IA entén patrons, recorda sintaxi, tradueix intencions a comandos, sense poder, ella mateixa, prémer mai la tecla. I jo, que abans em pensava l'element més feble del trio —

el que no sabia res, el que preguntava tot—, resulta que sóc l'únic dels tres que sap per què estem fent açò, i l'únic que pot decidir, al final, si la resposta és bona o no ho és.

No és, doncs, que la IA m'haja substituït davant del terminal. És que hem aprés, entre els dos, a fer de tercer l'un per a l'altre: jo li pose paraules i intencions a allò que el terminal escup en fred; la IA em posa comandos exactes a allò que jo sé descriure només amb paraules normals; el terminal executa, sense trair mai el que se li demana, exactament allò que entre tots dos hem decidit que calia fer.

Apunt tècnic: `stdout`, `stderr`, i com veure'ls (o amagar-los) a voluntat

Tot programa de terminal té, per defecte, tres canals numerats:

- `0` — **`stdin`:** l'entrada (el que li arriba, per exemple des d'una pipa `|`, com al capítol de les canonades).
- `1` — **`stdout`:** l'eixida normal, el resultat.

- **`2` — `stderr`:** els missatges d'error, separats a posta perquè no es barregen amb el resultat.

Redirigir només l'eixida normal a un arxiu (els errors continuen eixint per pantalla):

```
$ comanda > sortida.txt
```

Redirigir també els errors, junts amb l'eixida:

```
$ comanda > sortida.txt 2>&1
```

Redirigir els errors a un arxiu apart (útil per a revisar només els problemes):

```
$ comanda > sortida.txt 2> errors.txt
```

Descartar completament una eixida que no t'interessa, enviant-la a un «forat negre» especial del sistema:

```
$ comanda 2> /dev/null
```

Este comportament —dos canals separats per a resultat i error— és idèntic en Linux, macOS i, amb sintaxi pràcticament igual, en PowerShell de Windows (2> funciona també ahí). No és una peculiaritat d'un sistema concret. És una decisió de disseny vella de dècades, i que continua tenint sentit

avui: separar «el que ha passat» de «el que ha fallat» perquè cadascú puga anar on li cal, sense que es trepitgen.

exit 0

Epíleg: on he arribat, tot el que encara no sé, i per què eixir amb codi zero no vol dir haver-ho fet tot bé.

Vaig aprendre, cap al final d'estos mesos, una cosa que hauria d'haver preguntat el primer dia i no vaig preguntar fins ara: què vol dir eixe número que a vegades apareix quan un script acaba, eixe `exit 0` o `exit 1` que havia vist passar per pantalla desenes de voltes sense fer-hi cas.

```
$ ./exporta_notes.sh; echo $?
0
```

Un `0` vol dir que tot ha anat bé. Qualsevol altre número —`1`, `127`, els que siguen— vol dir que alguna cosa ha fallat, i quin número exacte és, de vegades, ni tan sols importa massa: només importa que no era zero.

Em va agradar eixa metàfora més del que esperava, i he decidit acabar el llibre amb ella perquè resumix, més bé que qualsevol altra cosa que puga dir, com em sent ara mateix, a punt de tancar estes pàgines.

Perquè jo no acabe este llibre sabent-ho tot. Encara em bloquege amb expressions regulars complicades. Encara no domine `tmux` com el company que me'l va descobrir. Encara hi ha comandos que li pregunte a la IA cada volta perquè no me'ls he acabat de gravar al cap, i probablement no me'ls gravaré mai, i ara sé que això no passa res. Si haguera d'esperar a saber-ho tot per a donar-me per satisfet, no acabaria mai —ni este llibre, ni, sospite, cap aprenentatge de veres.

Isc amb `exit 0`. No perquè ho haja fet tot bé —he esborrat una carpeta sencera sense voler, he confós `Baixades` amb `Downloads`, he estat a punt d'executar un `find /` que podria haver-me donat un mal de cap seriós, i encara he perdut quatre hores darrere d'una línia de `PATH` que faltava. Isc amb `exit 0` perquè, mirat de conjunt, el procés ha acabat com havia d'acabar: un professor que va deixar el terminal fa vint anys, i que ara torna a sentir-lo com allò que era quan tenia setze anys i un Amstrad PC1512 a la taula del menjador. Casa.

```
$ exit 0
```

Durant molt de temps vaig pensar que este seria el final del llibre —eixa comanda, eixe símbol net de tot ha anat bé, tancant-ho tot amb la mateixa precisió amb què va obrir-se al capítol primer. Però no ho serà. Perquè si tanque ací, encara estaria deixant que el lector es pensara que allò important d'este any ha sigut el terminal. I no ho ha sigut.

He aprés `pwd` i `chmod` i `grep` i `2>&1`. Sé obrir una connexió SSH, sé llegir un `git merge` sense entrar en pànic, sé quan confiar en la IA i quan aturar-me a llegir dos voltes abans de prémer Intro. Tot açò és cert, i tot açò em servirà. Però si ho reduïsc a una llista de comandos apresos, m'estic deixant la part que de veres importa.

El que de veres ha passat este any no és que un professor de cinquanta anys haja aprés terminal. És que un professor de cinquanta anys, que portava dècades sent l'expert de la sala —el que sap, el que explica, el que corregix—, va acceptar tornar a ser el més novell de la classe. Tornar a equivocar-se en públic. Tornar a preguntar coses que un xiquet de setze anys respon sense pensar-hi. Tornar, en definitiva, a ser aprenent.

I eixa lliçó, ara ho veig clar, no és de Linux ni de terminals ni de cap tecnologia concreta. És la mateixa lliçó que se li oblida a molta gent, passats els quaranta o els cinquanta, quan ja s'han acomodat en allò que dominen i deixen, sense adonar-se'n, de començar coses noves de zero. El terminal no era el tema d'este llibre. Era, només, l'excusa que la vida em va posar davant —una pantalla negra, un fill amb una idea, una nit a la cuina— perquè jo tornara a sentir, a cinquanta anys, allò que quasi havia oblidat que se sentia: la vergonya xicoteta i l'orgull immens de no saber alguna cosa, i decidir aprendre-la igualment.

Això és el que espere que et quede, si has arribat fins ací. No una llista de comandos —per a això ja tens internet, i una IA que te'ls explicarà millor que jo. El que espere que et quede és la pregunta que a mi em va canviar l'any: què tens tu encara pendent d'aprendre, no perquè et calga per a la faena, sinó perquè fa massa temps que has decidit que ja eres massa gran per a començar?

Perquè no ho eres. Ni jo ho era. `command not found` no vol dir que no pugues. Vol dir, només, que encara no t'han entés. Insistix.

Apunt tècnic: els codis d'eixida i per què val la pena mirar-los

Cada comanda o script que s'executa al terminal acaba amb un **codi d'eixida** (*exit code*), un número que indica si ha anat bé o no:

- `0` significa èxit. Sempre. És l'únic número reservat exclusivament per a «tot correcte».
- **Qualsevol altre número** (de l'1 al 255) significa que alguna cosa ha fallat. Cada programa decideix el seu propi significat per a cada número, i sovint no cal saber-los tots: n'hi ha prou de saber que no era zero.

Consultar el codi d'eixida de l'última comanda:

```
$ ./script.sh
$ echo $?
0
```

En PowerShell (Windows), l'equivalent és `$LASTEXITCODE`.

Per què importa: és la manera que tenen els scripts —i les IA, quan encadenen comandos per tu— de saber si poden continuar o si han d'aturar-se. Molts scripts fan servir `&&` per a executar una comanda només si l'anterior ha acabat amb `exit 0`:

```
$ cd capitulos && ls
```

Si el `cd` fallara (carpeta inexistent, per exemple), el `ls` mai s'executaria —una xicoteta xarxa de seguretat, feta de números, que evita que un error en cadena se'n vaja encara més lluny.

I amb això tanque l'apunt tècnic. Però no el diari, no de veres —eixe ja s'ha tancat unes pàgines arrere, amb l'única lliçó que valia la pena aprendre.

Annex

Ací acaba el diari. El que ve tot seguit no és cronologia ni anècdota.

Xuleta ràpida

Per a quan ja no cal la història, només la comanda.

No l'havia planejada, esta secció. Però mentre repassava els vint-i-cinc capítols per a tancar el llibre, em vaig adonar que jo mateix, mesos després d'escriure'ls, encara tornava a obrir-los sencers per a buscar un sol comando enmig d'una anècdota. Ací teniu, agrupats per tema i sense diari pel mig, els comandos que més repetisc. Entre parèntesis, el capítol on cadascun té la seua història, per si un dia voleu tornar-hi.

Moure's i mirar (cap. 4)

Comanda	Què fa
`pwd`	On estic ara (ruta completa)
`ls`	Què hi ha en esta carpeta

Comanda	Què fa
`ls -l`	Detall llarg: permisos, mida, data
`cd carpeta`	Moure's amb ruta relativa
`cd /ruta/completa`	Moure's amb ruta absoluta
`cd ~`	Tornar a la carpeta d'usuari
`cd ..`	Pujar un nivell
`cd -`	Tornar a la carpeta on eres just abans

Shell i configuració (cap. 2, 3, 5, 14)

Comanda	Què fa
`echo $SHELL`	Quina shell tinc ara
`cat /etc/shells`	Quines shells hi ha instal·lades
`chsh -s /bin/zsh`	Canviar la shell per defecte

Comanda	Què fa
`echo $PATH`	On busca el terminal els executables
`which comanda`	On és exactament una comanda instal·lada
`alias ll='ls -la'`	Crear una drecera per a una comanda llarga
`source ~/.bashrc`	Recarregar la configuració sense obrir terminal nou

Arxius i permisos (cap. 6, 8)

Comanda	Què fa
`ls -l arxiu`	Vore els permisos
`chmod +x script.sh`	Fer un arxiu executable
`chmod 755 script.sh`	Permisos rwx per al propietari, r-x per a la resta
`sudo chown usuari:grup arxiu`	Canviar el propietari

Comanda	Què fa
`sudo comanda`	Executar com a root (amb compte)
`rm -rf carpeta`	Esborrar sense confirmació — sense marxa arrere
`alias rm='rm -i'`	Fer que `rm` pregunte sempre abans d'esborrar

Gestor de contrasenyes (cap. 23)

Comanda	Què fa
`pass insert -m ruta`	Guardar una contrasenya nova, amb metadades a sota
`pass generate ruta 20`	Generar-ne una a l'atzar i guardar-la directament
`pass show ruta`	Consultar-la (només per a ulls humans, mai per a pantalla d'IA)

Comanda	Què fa
`variable=$(pass show ruta | head -n1)`	Llegir-la a una variable, sense imprimir-la
`comanda <<< "$variable"`	Passar-la per l'entrada estàndard, sense que aparega en text

Buscar i combinar amb pipes (cap. 9, 12)

Comanda	Què fa
`comanda1 | comanda2`	Envia l'eixida de la primera a l'entrada de la segona
`cat arxiu`	Mostrar el contingut d'un arxiu
`grep "text" arxiu`	Buscar línies amb «text»
`grep -r "text" .`	Buscar en tota una carpeta i subcarpetes

Comanda	Què fa
`grep -c` / `grep -l`	Comptar coincidències / mostrar només els noms d'arxiu
`wc -l` / `wc -w`	Comptar línies / paraules
`sort` / `sort -n`	Ordenar (alfabèticament / numèricament)
`uniq`	Eliminar línies repetides consecutives
`find ~ -name "*.tmp"`	Buscar arxius per nom (sense tocar-los)

Demanar ajuda (cap. 10)

Comanda	Què fa
`man comanda`	Obri el manual complet (s'ix amb `q`)

`comanda --help`	Resum ràpid de les opcions

Editar text (cap. 21)

`nano`	`vim`
`nano arxiu.txt` — obri, sense modes	`vim arxiu.txt` — obri, per modes
`Ctrl+O` guardar	`i` entrar en mode inserció
`Ctrl+X` eixir	`Esc` tornar al mode normal
`Ctrl+K` tallar la línia	`:w` guardar
`Ctrl+U` enganxar	`:wq` guardar i eixir
`Ctrl+W` buscar text	`:q!` eixir **sense** guardar

Controlar l'execució (cap. 12, 22, 25)

Comanda	Què fa
`Ctrl+C`	Interrompre una comanda en marxa
`comanda > arxiu`	Guardar l'eixida normal a un arxiu
`comanda 2> arxiu`	Guardar només els errors
`comanda > eixida 2>&1`	Guardar eixida i errors junts
`comanda 2> /dev/null`	Descartar els errors
`echo $?`	Vore si l'última comanda ha anat bé (`0`) o no
`comanda1 && comanda2`	Executar la segona només si la primera ha anat bé

Automatització amb cron (cap. 13)

Fixar el `PATH` dins de l'script, a la primera línia, perquè `cron` no falle per no trobar les comandes:

```
PATH=/usr/local/bin:/usr/bin:/bin
```

Comanda	Què fa
`journalctl -u cron`	Vore per què ha fallat un `cron` (Linux amb systemd)
`env -i script.sh`	Provar un script amb un entorn tan buit com el de `cron`

Git (cap. 15, 16)

Comanda	Què fa
`git add arxiu`	Preparar un canvi
`git commit -m "descripció"`	Guardar-lo amb un missatge
`git push`	Pujar-lo al repositori remot
`git status`	Vore l'estat (i els conflictes)
`git merge --abort`	Rendir-se i tornar arrere una fusió

Terminal remot i xarxa (cap. 16, 18, 20, 24)

Comanda	Què fa
`ssh usuari@servidor`	Obrir una sessió remota
`tmux`	Obrir una sessió persistent nova
`tmux new -s nom`	Obrir-ne una amb nom, per a distingir-ne diverses
`Ctrl+b`, després `d`	Desconnectar-se de `tmux` sense tancar-la
`tmux ls`	Vore les sessions vives en eixa màquina
`tmux attach`	Tornar a connectar-se a l'última sessió
`tmux attach -t nom`	Tornar exactament a la sessió que toca
`sudo tailscale up`	Activar Tailscale al dispositiu

Comanda	Què fa
`ssh usuari@nom-del-dispositiu`	Connectar per la xarxa Tailscale, sense IP a memoritzar

Eines TUI per al dia a dia (cap. 17)

Eina	Per a què serveix
`neomutt`	Client de correu
`newsboat`	Lector de feeds RSS/Atom
`nchat`	Client de xat (WhatsApp, Telegram)
`btop` / `htop`	Monitor de recursos del sistema
`ranger` / `nnn`	Explorador d'arxius per teclat
`lazygit`	Interfície visual per a `git`

Cap d'esta llista cal memoritzar-la de colp —jo mateix encara en consulte la meitat cada volta que en necessite alguna. Val més tindre-la a mà que tindre-la al cap.