

## Cloud Operator Days Tokyo 2026

# ワークライフハックセッション ITワーカーに光を! ~創る環境・変わる働き方~

Cloud Operator Days Tokyo 2026 実行委員会



@CloudOpsDaysTokyo



@cloudopsdays



## 本セッションの進め方

### 委員による簡単なハック紹介

- 実体験に基づく知見共有

### インタラクティブなディスカッション

- SOE.LでQ&A

### 参加者の知見共有

- 体験を皆で学びあう！



## 4つのテーマ

### 今年買ってよかったガジェット

### デスク紹介・テレワーク品質向上

### 自作キーボード

### 生成AIワークハック

奥村優斗  
@AXLBIT



竹本裕宣  
@GMO



島津直道  
@DENSO



山下 祐生  
@レッドハット



# 最近買ってよかったガジェット紹介

# ウェアラブルデバイスはイイぞ

スマートグラス



EvenG2

スマートウォッチ



Apple Watch

# リモートから、オフィスへ

オフィス回帰が進み、対面での会議・商談・雑談が増えている。  
オンライン会議では気にならなかった不便が、対面のコミュニケーションでは表面化してきます。



# オンラインでは、PC画面が助けてくれていた

---

会議アプリを開けば、これだけの支援が画面の中にありました。

## 台本

発表原稿を画面に表示しながら話せる

## 翻訳

外国語の会話も、リアルタイムで日本語に

## AI議事録

会話が自動で要約・整理される

**オンラインでは、画面が助けてくれていた。  
でも対面では、その機能を使えない。**

対面では、字幕も翻訳もメモも、手元から遠くなった。

# AR / VRデバイス、使ったことがありますか？

Meta Quest  
Apple Vision Pro  
XREAL  
Ray-Ban Meta  
その他スマートグラス

“仕事中ずっとかけられるARデバイス”  
はどうでしょう？

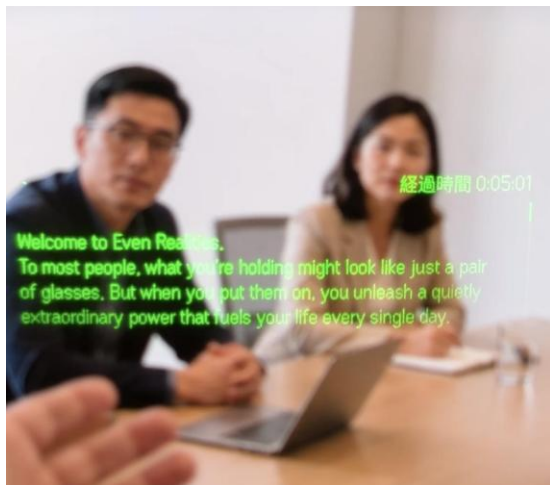
必要な情報を、相手から目を離さずに確認できるスマートグラス。



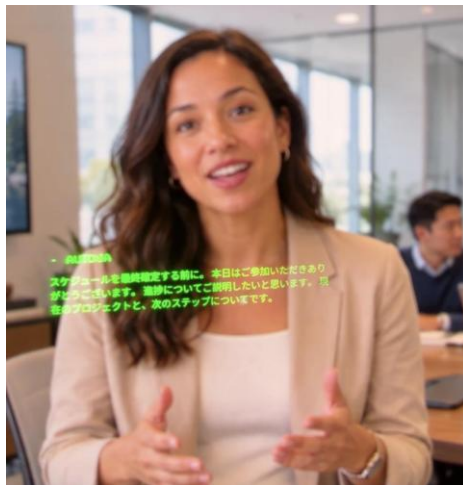
先ほどの困りごとが、視界の中で解決されていきます。

# 対面コミュニケーションを、視界の中から支援する

## テレプロンプター



## リアルタイム翻訳



## AI要約



**どれだけ便利でも、  
普段から着け続けられなければ意味がない。**

# 自然に着用できる

スマートグラスらしさが強すぎず、一般的なメガネに近い見た目です。

日常の服装やオフィス環境にも馴染みやすいデザイン



# カメラがない、という選択

---

Even G2にはカメラが搭載されていません。  
だからこそ、周囲から「撮影されているのでは」と思われにくい。

**「カメラがない」は弱点ではなく、  
対面で自然に使うためのメリットです。**

特に、オフィスや対面コミュニケーションで使うスマートグラスとして、この点は重要です。

一日持ち歩いて使うデバイスとして、無理のない軽さと駆動時間です。

36g

## 軽さ

一般的なメガネに近い  
軽さで、着け続けやすい

約48h

## バッテリー

通常使用で最大約2日間。  
毎日の充電に追われにくい

控えめ

## デザイン

スマートグラスらしさを  
主張しすぎない見た目

# Even R1で操作する

Even G2をより自然に操作するための存在——それがEven R1です。



- 指先のタップ・スクロールで操作が完結する
- スマホを取り出さない
- 相手から視線を外さない
- 視界内の情報を、そのまま操作できる

オンラインでは、PC画面が支援していた。  
対面では、その支援を視界の中へ。

# 初めてのApple Watchを買いました



画像：Apple Newsroom

2025年9月、発売日に購入

## Apple Watch Series 11

これが初めてのApple Watch

### 01 アラームの音で起こされたくない

手首の振動だけで、静かに起きたい

### 02 睡眠を記録したい

自分の睡眠を、数字で見たい

# Apple Watchでできること

腕時計というより、手首につける小さな道具のセット。今日はこのうち2つの話をします。



## 日常ツール

アラーム、タイマー

## 通知・通信

通知、電話、メッセージ

## 健康・睡眠

睡眠記録、心拍数

## 運動

アクティビティリング、ワークアウト

## 安全

転倒検出、緊急SOS

## 決済・連携

Suica、Macのロック解除

使える機能はモデル・地域・watchOSのバージョンで異なります（例：心電図アプリはSE 3では非対応）

# SE 3・Series 11・Ultra 3の違い



SE 3

|                |              |
|----------------|--------------|
| ケースサイズ／素材      | 40・44 mm アルミ |
| バッテリー（通常使用）    | 最大18時間       |
| 心電図・血中酸素・高血圧通知 | 非対応          |
| 耐水性能           | 50 m         |
| 安全・屋外          | 転倒検出、緊急SOS   |

## 私の使用モデル



Series 11

|                |                  |
|----------------|------------------|
| ケースサイズ／素材      | 42・46 mm アルミ／チタン |
| バッテリー（通常使用）    | 最大24時間           |
| 心電図・血中酸素・高血圧通知 | 対応               |
| 耐水性能           | 50 m             |
| 安全・屋外          | 転倒検出、緊急SOS       |

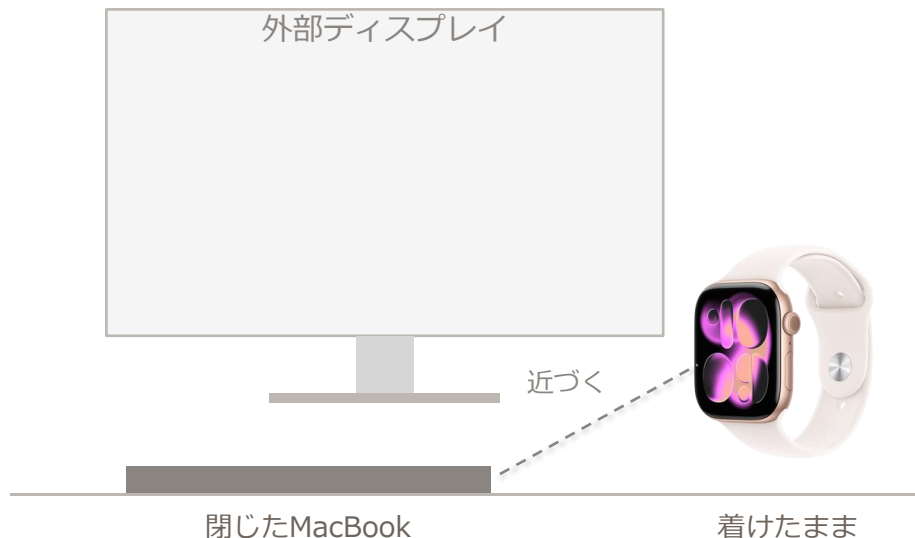


Ultra 3

|                |                        |
|----------------|------------------------|
| ケースサイズ／素材      | 49 mm チタン              |
| バッテリー（通常使用）    | 最大42時間                 |
| 心電図・血中酸素・高血圧通知 | 対応                     |
| 耐水性能           | 100 m、ダイビング対応          |
| 安全・屋外          | + 衛星通信、2周波GPS、アクションボタン |

3モデル共通：常時表示、通知・電話、Suica、Macのロック解除、睡眠スコア、手首の皮膚温  
バッテリーはApple公称値（フル充電での通常使用時）。使用条件や設定によって変わります。

# 予想外に便利だった、Macの認証



## 近づくだけでロック解除

Watchを着けてMacの前に座れば、パスワードを打たずにログインできる。本体を閉じたクラムシェルでも動く。

## パスワード入力も、Watchで承認

システム設定の変更など、管理者パスワードを求められる場面は、Watchのサイドボタンをダブルクリックで通る。

どちらもmacOS標準機能。起動直後の初回ログインなど、パスワードが要る場面は残る。

**クラムシェルのままだでも、毎日のパスワード入力が減った**

# 睡眠記録で、生活習慣が変わった

振動だけで起きられるのは期待どおり。予想外だったのは、記録を見てからの変化でした。

記録を見て分かったこと

**自分の睡眠のクセが、数字で見える**

**飲んだ日は、スコアが下がりがち**

私の記録での傾向。医学的な話ではない



睡眠スコアの表示例（Apple公式サイト）。私の記録ではない

変えた行動

**就寝時刻を、毎日できるだけ揃える**

毎朝スコアを見るようになってから、寝る時刻を意識するようになった

**お酒を飲む頻度を減らす**

スコアが下がる日の共通点が飲酒だった。それをきっかけに、飲む回数を減らした

— アクティビティリングは、消費カロリーの目安として参考程度に見ている

# 毎日つけるための充電習慣

「寝るときもつけるなら、いつ充電するの？」 私の場合は、入浴中です。

朝

**振動で起きる**

日中

**着けたまま**

夜・入浴中

**外して充電**

就寝

**また着けて寝る**

音は鳴らさず、手首の  
振動だけで起きる

通知、Macのロック解除、  
タイマー

浴室の外の充電器に置く。  
私の使い方ではこれで足り  
ている

睡眠を記録。寝ている  
間の装着は、意外と気  
にならなかった

毎日この繰り返し

— 充電の回数や残量は使い方が変わります。睡眠記録と通知が中心の、私の使い方での話です。

# 身につけるガジェット、何を使っていますか？

スマートグラス



Even Realities  
**Even G2**

スマートウォッチ



Apple Watch  
**Series 11**

スマートバンド



Xiaomi Smart Band  
**10 Pro**



Google  
**Fitbit Air**  
画面なし

スマートリング



Oura  
**Ring 5**



RingConn  
**Gen 3**

## どんなときに便利ですか？

- スマートリングは未使用ですが、気になっています

# デスク紹介・テレワーク品質向上



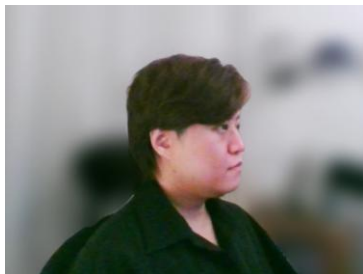
# 同じ「映る」でも、印象がまったく違う



**OBSBOT Tiny 2 Lite**  
4K対応のジンバルWebカメラ  
AIによる人物追尾・ズーム機能

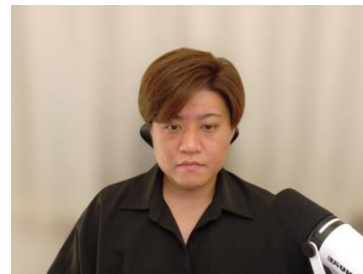
## ノートPC内蔵カメラ

- ノートPCが横にあり、横向き画角
- センサーが小さく、逆光や室内照明に弱い
- 色が沈みがちで、暗い印象になりやすい



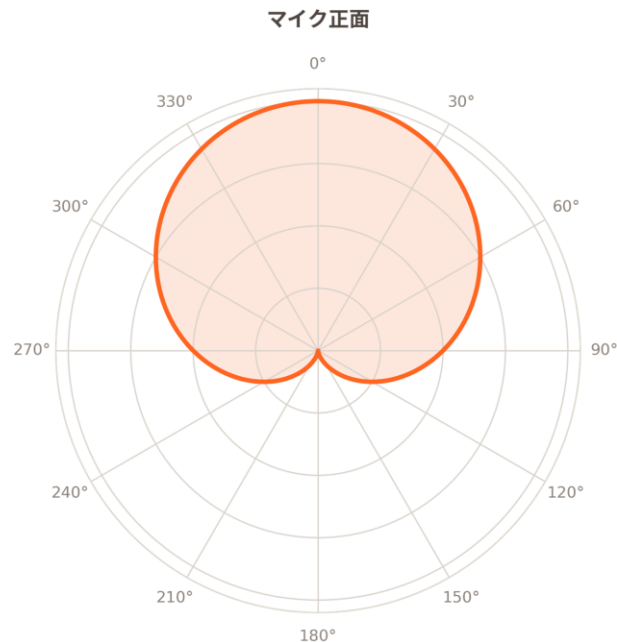
## 単体カメラ（OBSBOT）

- カメラだけを正面に独立配置
- センサーが大きく、色再現が自然に近い
- 色が安定し、明るい印象



# 指向性という仕組みで、周囲の音を拾いにくくする

Shure MV7+は単一指向性（カーディオイド）。  
マイクの正面方向の音を中心に拾い、背面や側面からの音は拾いにくくなります。



リモートでの周囲の生活音（声・エアコン）が入らなくなりました。スピーカーで会議しても音がループしないです。

# マイクアームもいいぞ

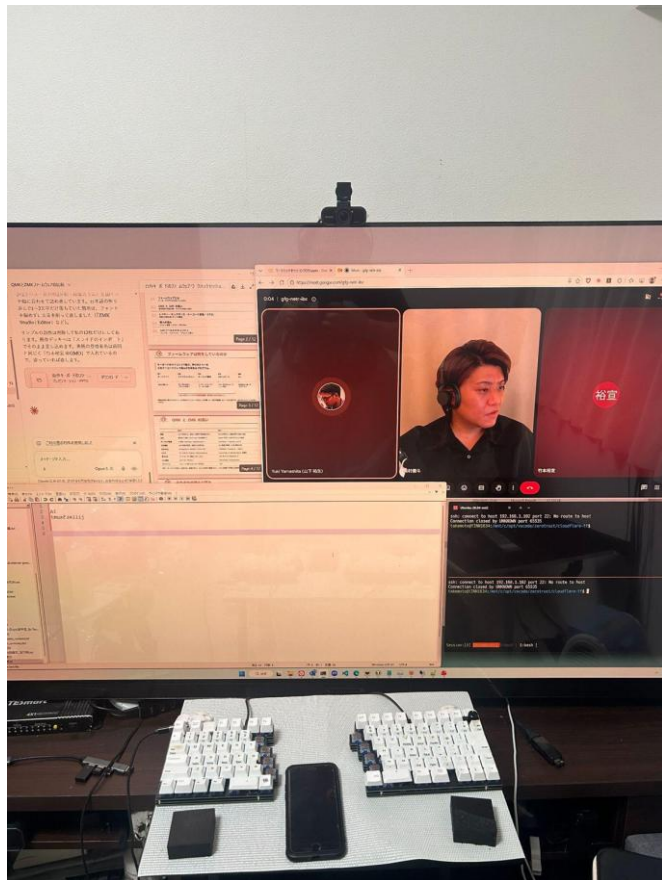


マイクアームを導入することで、  
口にマイクを近づけることができる。

Gainを下げることができ、  
さらに周囲の音を拾いにくくできる。

机もスッキリ！

# デスクの空間活用事例



65インチ有機EL REGZA!

# デスクの空間活用事例

2025年



# デスクの空間活用事例



# 分割キーボードに手を出してしまった



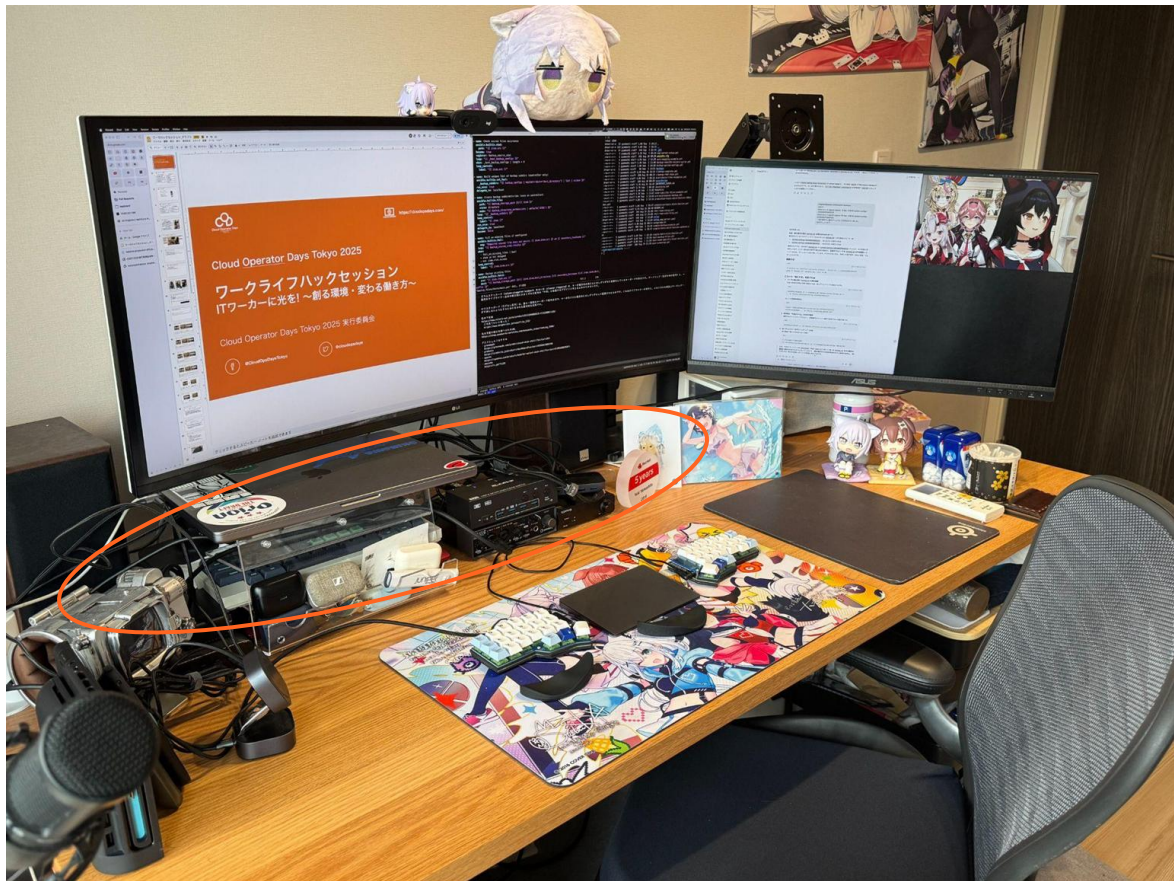
Jezailfunder  
Cornix LP



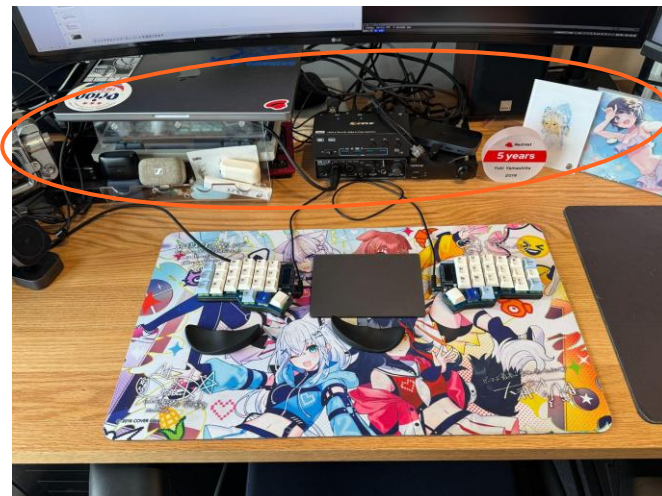
Jezailfunder  
Jiffy75 LP



Keychron × ギズモード・ジャパン  
Keychron Orca echo



2025年





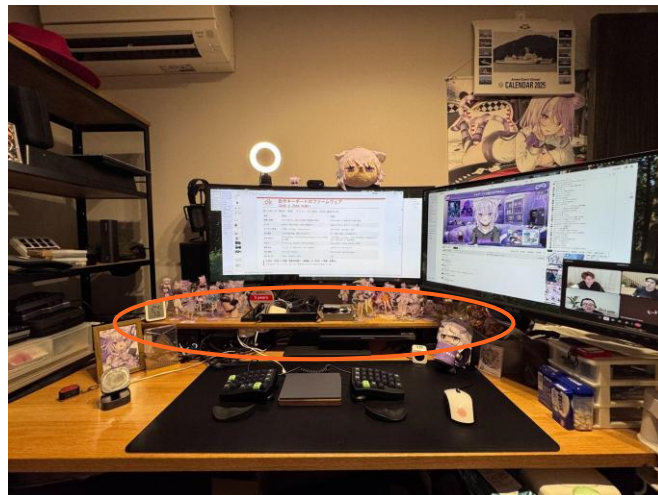
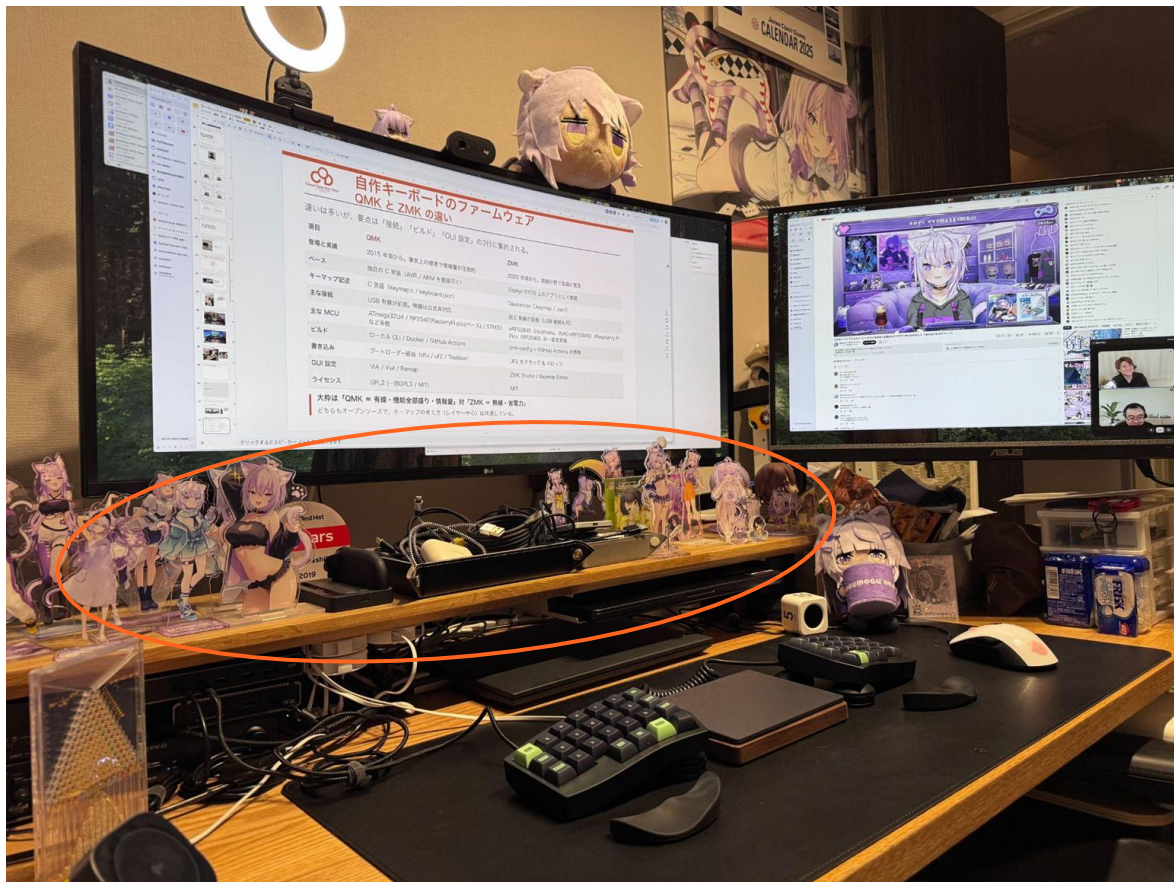
PREDUCTS

## DASHBOARD Desk Shelf

- 必要な道具を「隠さず、散らかさず」置ける
- 2本のレールで、自分好みにカスタマイズできる
- モジュールを追加して、使い方をあとから広げられる
- デスク上の作業スペースを広く保てる
- 木の質感と機能性を両立した、洗練されたデザイン
- 「棚」ではなく、仕事環境を整えるための土台
- 働き方や持ち物が変わっても、長く使い続けられる

# デスクの空間活用事例

2026年



# 自作キーボード

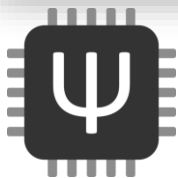
# 自作キーボードのファームウェア(ソフトウェア)



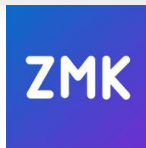
こういうキーボードって、  
数字キー、記号キー、  
F1などのキーを、  
どうやって  
タイプするのかしら？

**キーマップは、自分で書き換えられる。**

レイヤー／タップダンス／キーコード変換／マクロ。打鍵数と指の移動を減らす4つの機能と、導入の手順、GUI設定ツールとセキュリティまで。



QMK



ZMK

# 自作キーボードのファームウェア

## QMK と ZMK の違い

| 項目      | QMK                                                    | ZMK                                                                    |
|---------|--------------------------------------------------------|------------------------------------------------------------------------|
| 登場と実績   | 2015 年頃から。事実上の標準で情報量が圧倒的                               | 2020 年頃から。無線分野で急速に普及                                                   |
| ベース     | 独自の C 実装 (AVR / ARM を直接叩く)                             | Zephyr RTOS 上のアプリとして実装                                                 |
| キーマップ記述 | C 言語 (keymap.c / keyboard.json)                        | Devicetree (.keymap / .conf)                                           |
| 主な接続    | USB 有線が前提。無線は公式非対応                                     | BLE 無線が前提 (USB 接続も可)                                                   |
| 主な MCU  | ATmega32U4 / RP2040(Raspberry Pi picoベース) / STM32 など多数 | nRF52840 (nice!nano、XIAO nRF52840) / Raspberry Pi Pico (RP2040) ※一部未実装 |
| ビルド     | ローカル CLI / Docker / GitHub Actions                     | zmk-config + GitHub Actions が標準                                        |
| 書き込み    | ブートローダー経由 (dfu / uf2 / Toolbox)                        | UF2 をドラッグ & ドロップ                                                       |
| GUI 設定  | VIA / Vial / Remap                                     | ZMK Studio / Keymap Editor                                             |
| ライセンス   | GPL2 (一部GPL3 / MIT)                                    | MIT                                                                    |

大枠は「QMK = 有線・機能全部盛り・情報量」対「ZMK = 無線・省電力」

どちらもオープンソースで、キーマップの考え方 (レイヤー中心) は共通している。

次ページから4つの特殊機能を紹介

# 自作キーボードのファームウェア

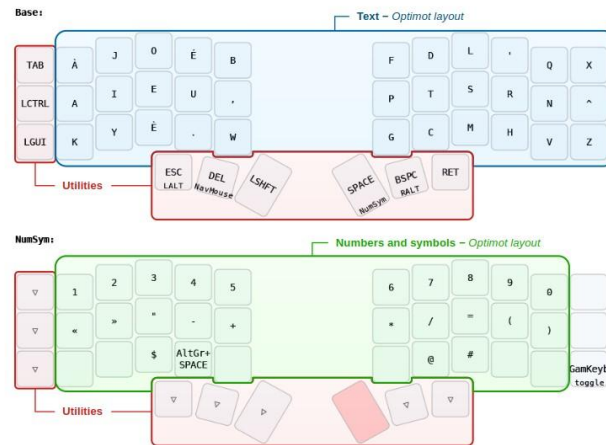
## レイヤー：キーに何枚もの「面」を持たせる

Fn キーのような「面」を自分で何枚でも定義できる。指の移動を減らす中核機能。

### 代表的な切り替え方

- 押している間だけ有効： QMK MO(1) / ZMK &mo 1
- トグルで固定： QMK TG(1) / TO(1) / ZMK &tog &to
- 長押しでレイヤー・単押しでキー： QMK LT(1,KC\_SPC) / ZMK &lt 1 SPACE
- 2 枚同時押しで 3 枚目： QMK Tri Layer / ZMK conditional-layers

- HJKL で ←↓↑→ ができます！
- グローバルホットキーでF13やF24も使えます！



同じ物理キーが、有効なレイヤーによって別のキーになる

### カラムスタガードや 40% キーボードでは必須

物理キーが 40～50 個しかない配列では、数字・記号・矢印・ファンクションを置く場所が物理的に足りない。レイヤーを重ねることで、ホームポジションから手を離さずフルサイズ相当の入力ができる。

# 自作キーボードのファームウェア タップダンス：タップ回数で動作を変える

同じキーを 1 回叩いたときと 2 回叩いたときで別の動作を割り当てられる。

## 実例：IME の ON/OFF を 1 キーに集約

1× 1 タップ → IME OFF (英数 / LANG2)

2× 2 タップ → IME ON (かな / LANG1)

トグルではないので「今どちらの状態か」を気にせず、押した回数で必ず狙った状態になる。

## 使いどころ・注意点

- 3 回以上や「2 回叩いて長押し」も定義できる
- 判定待ち時間 (tapping-term) の間は出力が確定しないため、短すぎても長すぎても打ちづらい。150~200ms 前後から調整
- 連打が多いキー (矢印など) には向かない

## QMK

```
// keymap.c
tap_dance_action_t tap_dance_actions[] = {
    [TD_IME] = ACTION_TAP_DANCE_DOUBLE(
        KC_LNG2, KC_LNG1),
};
// キーマップ側では TD(TD_IME) を配置
```

## ZMK

```
// *.keymap
td_ime: tap_dance_ime {
    compatible = "zmk,behavior-tap-dance";
    #binding-cells = <0>;
    tapping-term-ms = <200>;
    bindings = <&kp LANG2>, <&kp LANG1>;
};
```

※ キーコード名は各 firmware の keys.h / keycode 一覧を確認

# 自作キーボードのファームウェア JIS ⇔ US キーコード変換

HID が送るのは「キーの位置」だけ。文字に変換するのは OS の配列設定。

## 実例 : Windows は JIS 配列として認識している

この状態で「@」を打ちたいのに、US 配列の感覚で Shift+2 を送ると「"」が出てしまう。JIS 配列で @ が置かれている位置のキーコードを送る必要がある。

```
#include "keymap_japanese.h" // QMK
JP_AT // 実体は KC_LBRC → JIS 環境で @
```

## 運用のコツ

- QMK は keymap\_japanese.h (JP\_AT 等) と sendstring\_japanese.h で JIS 環境に対応
- ZMK は US 基準が標準。JIS 向けは自作 define や mod-morph で対応
- 「OS 側を US にする」「ファーム側で吸収する」のどちらかに統一する

同じ「@ を打つ」でも、送るキーコードで結果が変わる

US 配列の感覚で Shift + 2 を送る



× 「"」が出てしまう

いずれの場合も、OS (Windows) 側は JIS 配列として解釈する

JP\_AT (= KC\_LBRC) を送る



○ 「@」が出る

これをするには、ほぼビルドが必要。これをしたいがために、ビルドするためめっちゃ調べました(´・ω・`)

# 自作キーボードのファームウェア

## マクロ：1 キーで一連の操作を送る

定型入力やショートカットに加えて、キーボードの外の機器の操作まで 1 キーに封じ込められる。

### 実例：KVM スイッチの接続切替をワンキー化

KVM スイッチ側のホットキー（PrintScreen 2 連打がプレフィックス）をマクロに登録。切替のたびに 3 打鍵していたものが、指 1 本で終わる。

PrtSc

PrtSc

1

→ 個人 PC へ切替

PrtSc

PrtSc

2

→ PS5 へ切替

### QMK

```
case KVM_PC:
    if (record->event.pressed) {
        SEND_STRING(SS_TAP(X_PSCR)
                    SS_TAP(X_PSCR) "1");
    }
```

### ZMK

```
kvm_pc: kvm_pc {
    compatible = "zmk,behavior-macro";
    wait-ms = <40>; tap-ms = <40>;
    bindings = <&kp PSCRN &kp PSCRN &kp N1>;
};
```

OSのショートカットキーをマクロ登録するというのもあります

# 自作キーボードのファームウェア 簡易ツール：ビルドせずにキーマップを変える

ファームウェアビルド時に対応させるなら、ブラウザやアプリからリアルタイムにキーマップを書き換えられる。

## QMK 系

### VIA / Vial / Remap

- キーマップ・マクロ・レイヤーを即時変更
- Vial はコンボやタップダンスも GUI で設定可
- ファーム側で VIA/Vial 有効化と定義ファイルが必要

## QMK 系

## ZMK 系

### Keychron Launcher

- VIA ベースの Keychron 公式 Web 設定ツール
- 対応機種向けに定義が用意済みで導入が楽
- 自作機では基本的に VIA/Vial を使う

## ZMK 系

### ZMK Studio / Keymap Editor

- Studio は USB/BLE 経由でリアルタイム変更
- 利用には studio-unlock キーの設定が必要
- Keymap Editor は GUI 編集 → GitHub でビルド

参考：Remap（日本発の Web 設定ツール。QMK / VIA 対応）も同じ枠組みで使える。

## 注意点：WebHID とセキュリティ

### ブラウザ側

- 一度許可したサイトは raw HID に自由にアクセスできる。書き換えでキーロガー化・入力注入も理屈上は可能
- 既定は「サイトが要求できる」状態。必要なとき以外はブロックし、使用後は許可を取り消す
- Firefox・Safari は懸念から未実装。使えるのは Chromium 系ののみ

### ファーム側

- Vial は既定で Locked モード。解除は物理キーを数秒ホールドする操作が必要
- ロック中はマクロ変更とブートローダー起動が禁止。キーマトリクス読み出しは常時禁止
- 編集後は「Lock」が再起動で施錠。信用できない端末では解除操作をしない
- VIA 単体にロックは無い。QMK は SECURE\_ENABLE = yes（既定 60 秒で自動ロック）

# キーキャップの選び方

配列の次は、指がいちばん長く触れている面の話。キーキャップは大きく4つの軸で決まる。

01

## プロフィール

高さ・段差・上面の形。  
打鍵感はほぼここで決まる。

02

## 素材

ABS / PBT / POM。  
手触り・音・経年のテカリ。

03

## 印刷方式

文字の作り方。消えるか、  
出せる色が変わる。

04

## 互換性

MXマウント、キーサイズ。  
必要なキーが揃うか。

**決める順番は、プロフィール → 素材 → 印刷方式。互換性は最後に確認する**

打鍵感への効きがこの順番。逆から選ぶと、見た目は好きなのに指に合わない、が起きる。

同じ形でも、素材が変わると手触り・音・経年変化はまったく別物になる。

| 素材  | 手触りと音                  | 経年変化                   | 相性の良い印刷    | ひとこと                      |
|-----|------------------------|------------------------|------------|---------------------------|
| ABS | つるつと滑らか。<br>高めのカチツとした音 | 指の脂でテカる。<br>よく打つキーから光る | ダブルショット    | 発色が良く色数を出せる。<br>GMK が定番   |
| PBT | ざらつとマット。<br>低めのコトツとした音 | テカりにくく、<br>長く同じ顔のまま    | 昇華／ダブルショット | 厚く作りやすい。<br>いまのカスタムの主流    |
| POM | しっとり滑らか。<br>最も静かで丸い音   | 指紋は付くが<br>テカりはしない      | 選択肢が少ない    | 半透明でぬるつとした質感。<br>出回る数は少ない |

**PBT の強みは「テカらない」と「厚く作れる」。厚みは剛性と、底打ちの音の低さに直結する。**

標準的なキーキャップの肉厚は 1.2～1.4mm 前後。1.5mm を超えると「厚肉」と呼ばれる領域に入る。

# キーキャップ印刷方式

文字の作り方は大きく4通り。断面で見ると、何が違うのかが一目でわかる。

## パッド印刷

乗せる



表面にインクを転写しているだけ。安価だが、使ううちに擦れて消えていく。

## 昇華 (dye-sub)

染み込ませる



熱で染料を樹脂に浸透させる。消えないが、地より濃い色しか出せない。

## リバース昇華

地を染める



地の方を濃く染め、文字を地色のまま残す。黒地に白文字はこの方式。

## ダブルショット

成形する



2色の樹脂を重ねて成形。文字が裏まで貫通し、原理的に消えない。

**見分けるポイントは「消えるか」と「黒地に白を出せるか」の2つだけ**

長く使うならダブルショットか昇華。黒地に白文字が欲しいなら、地の方を染めるリバース昇華のセットを探す。

プロフィールとは、キーキャップの「高さ」と「形」。段差の付け方と、上面の削り方の2軸で決まる。

## 段差 — 行ごとに高さを変えるか

スカルプチャード Cherry / OEM / SA / MT3



ユニフォーム(ノンスカルプチャード)

DSA / XDA / KAM / PBS



スカルプチャードは指の長さに合わせて掘ってある。ユニフォームはどの段に何を置いても同じ形 = 自作・分割キーボードと相性が良い。

## 上面 — 指が触れる面の削り方

シリンダリカル

スフェリカル

フラット

左右方向



前後方向



Cherry / OEM

SA / DSA / KAM

G20 / ノートPC

シリンダリカルは左右だけ、スフェリカルは全方向に凹む。凹みが深いほど指は収まるが、隣のキーへ逃げる自由度は下がる。

## この2軸の組み合わせに名前を付けたものが「プロフィール」

Cherry = スカルプチャード × シリンダリカル、DSA = ユニフォーム × スフェリカル。

## KAM Superuser

Keyreative KAM / 9.05mm / 厚肉PBT・昇華

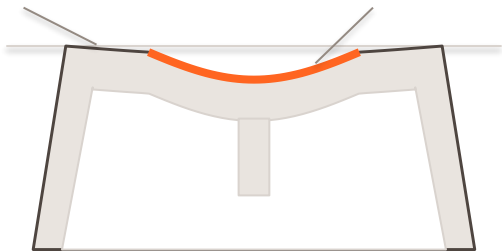
## PBS BoB

Keyreative PBS / 約7.4mm / 厚肉PBT・リバーズ昇華

### PBS の上面は、円柱と球のハイブリッド

外形は円柱の曲面

中央のくぼみは球



オレンジの部分が、実際に指が当たるスクープ。

### この2つを選んでいる理由

- | ユニフォームだから、40% でも分割でも、好きな段に好きなキーを置ける
- | 上面だけがしっかり凹み、指が吸い付くように収まる
- | 1.5～1.7mm の厚肉 PBT。たわまず、底打ちの音が低い

## ユニフォームの自由度そのままに、上面だけが指を受け止める

DSA・XDA のように置き換えが自由なまま、指の収まりはスフェリカル。そこが好き。

# 生成AIワークハック

去年は、道具の話だった。今年は、その道具を使う相手の話をします。

## あなたの仕事場、AIはどこに住んでいますか？

### GUI派

見て、選んで、任せる

### CLI派

打って、繋いで、流す

### AI最適化CLI派

AIに合わせて道具を変える

### Unix職人派

道具を変えず、AIを乗せる

**投票も質問も、全部その場の挙手でやります!!!!**

スマホは出さなくていいので、手を挙げる準備だけしておいてください。

# あなたのAIはどこに住んでいますか？

同じ「AIを使っている」でも、AIの居場所が違くと仕事の進め方まで変わる。



## ブラウザのタブ

チャット画面で相談する



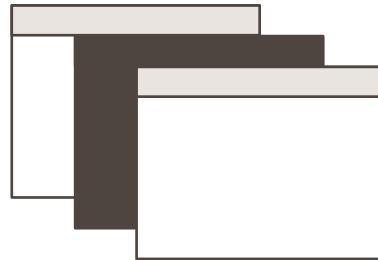
## エディタの中

コードを見ながら書かせる



## ターミナルの中

コマンドの流れに混ざる



## 気づいたら全部

住み着いて出ていかない

## AIの居場所が変わると、仕事場の景色も変わる

タブに住んでいる人は「相談」を渡し、ターミナルに住んでいる人は「作業」を渡しがち。

## まず、正直に手を挙げてください

見栄を張ると、あとのパネルで話が噛み合わなくなります。

### A

#### GUI派

AIとは画面で話す。見て、選んで、任せる

### B

#### CLI派

AIとはターミナルで話す。打って、繋いで、流す

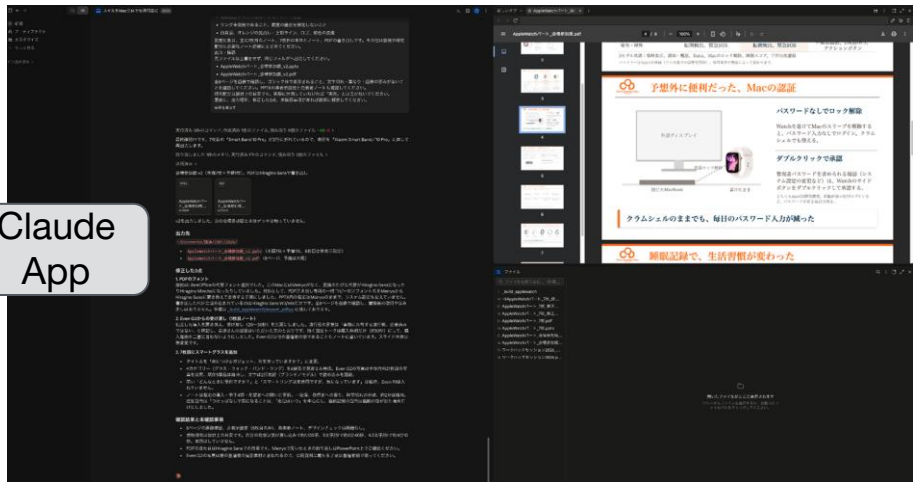
### C

#### 二刀流

場面で使い分ける。ただし比率には流儀がある

画面を見ながら、AIと一緒に考える。

Claude  
App



使っているもの

Claude / ChatGPT のデスクトップ、Cursor、ブラウザのタブ

- 差分も図もその場で目視できる。判断が速い
- チームに見せやすい。画面共有がそのまま説明になる
- 環境構築がほぼ要らない。今日から同じ環境を作れる



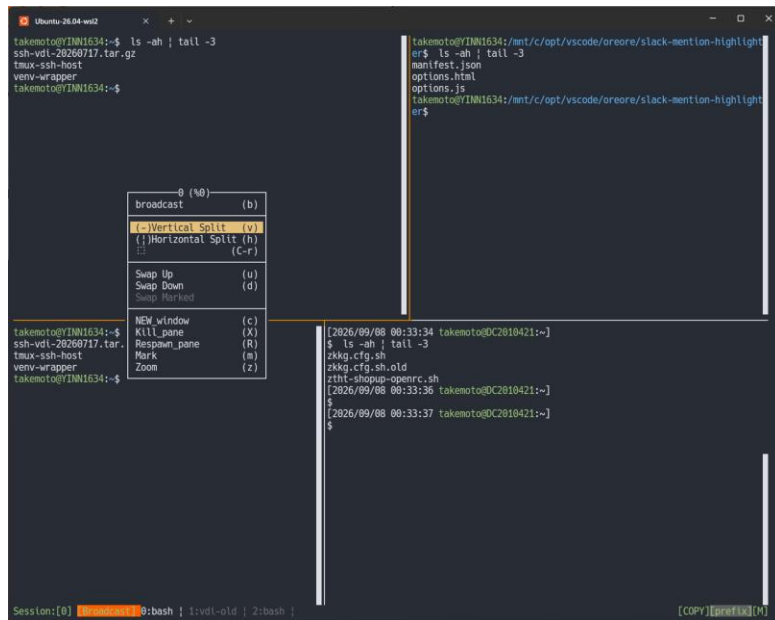
修正されたコードが見やすい！ターミナルを共有できるのはよい！  
CUIでできることはGUIでもできっから！

手を止めずに、AIへ流し込む。見通しより、手数。

使っているもの

**Claude Code などの CLI、shell、Git、エディタが  
一本の流れ**

- キーボードから手が離れない。思考が途切れない
- パイプとスクリプトで、他の道具と繋がられる
- 同じ手順を明日も再現できる。作業が資産になる



```
takemoto@YTM1634:~$ ls -ah | tail -3
ssh-vdl-20260717.tar.gz
tmux-ssh-host
venv-wrapper
takemoto@YTM1634:~$

[2026/09/08 00:33:34 takemoto@DC2010421:~]
$ ls -ah | tail -3
zkgg.cfg.sh
zkgg.cfg.sh.old
zsh -s sh.cold
[2026/09/08 00:33:36 takemoto@DC2010421:~]
$
[2026/09/08 00:33:37 takemoto@DC2010421:~]
$
```

bash、zsh、vim、git、curlコマンドを生成AIがよしなに操作してくれる  
コーディングエージェント(claudecode codex)との相性がよいと思う

ここで会場の実態を聞いてみたいと思います。当てはまるものがあれば手を挙げてください！

| コード以外の資料をメインに生成AIを使っている人

---

| コード(プログラム、IaCなど)をメインに生成AIを利用している人

---

| AIが書いたコードを、一度も読まずにマージしたことがある人

---

| AIEージェントを3体以上、同時に動かしている人

---

| マウスに手を伸ばしたら負けだと思っている人

---

同じターミナルを使っている、道具を誰に合わせるかで真っ二つに割れる。

## A

### AIに最適化されたツール

「AI時代なんだから、AIに最適化された道具を使うべき」

AI連携ターミナル / エージェント管理UI / 並列実行

## B

### Unix職人派

「Unixは50年戦える。AIが来ても、道具は変えなくていい」

Ghostty / tmux / Zellij / Neovim / shell / Git

**「CLIが速い」では一致している。割れるのは、道具を誰に合わせるか**

ワークフローにツールを合わせるのか、ツールにワークフローを合わせるのか。宗教戦争の本体はここ。

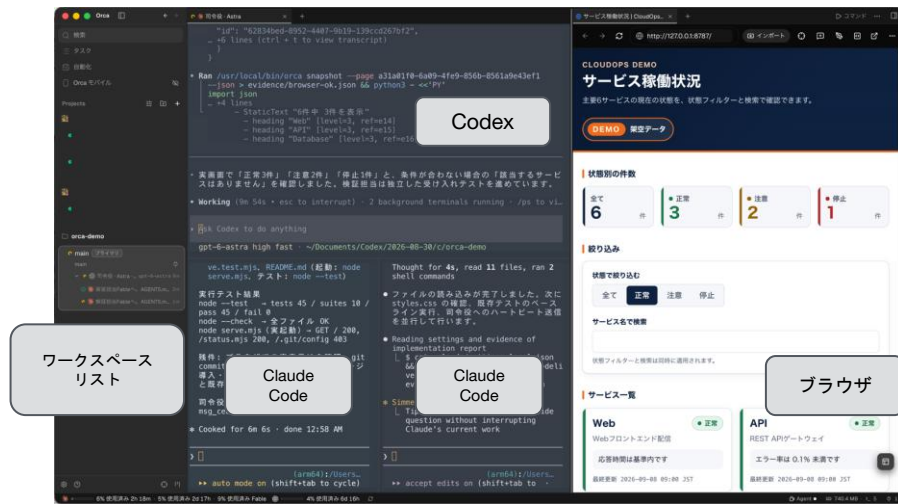
使っているもの

AI連携ターミナル (Orca / Warp / cmux など)、エージェント管理、並列実行

- エージェントを複数走らせても、どれが何をしているか見失わない
- AI向けの入出力が最初から整っている。設定に時間を使わない
- 新しい使い方が来たとき、ツール側が先に追いついてくれる



cmux



Orca

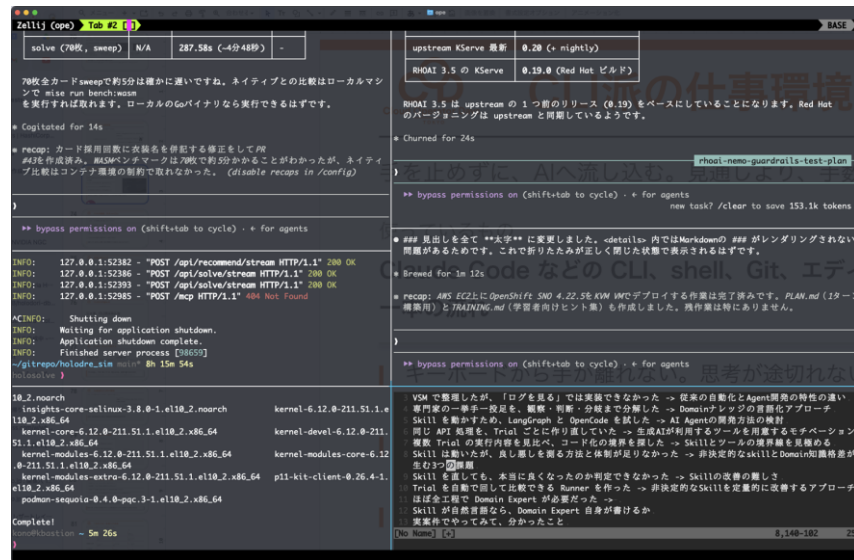
# Unix職人派の環境(awの紹介)

道具は変えない。使い慣れた組み合わせの上に、AIを一枚乗せる。

使っているもの

**Ghostty / Zellij / Neovim / zsh / gh / mise /  
aw(自作ツール)**

- どのマシンでも、dotfiles を撒けば同じ環境が戻ってくる
- AIも1プロセス。パイプとリダイレクトで他の道具と繋がる
- 10年使った操作をそのまま使える。学び直しのコストがゼロ



The screenshot shows a terminal window with multiple tabs. The active tab is titled 'Zellij (ope)' and 'Tab #2'. The terminal output includes various system logs, network activity, and command outputs. Key elements visible include:

- Terminal tabs: 'solve (70枚, sweep) N/A', '287.58s (-4分48秒)', 'upstream KServe 最新 0.20 (+ nightly)', 'RHQAI 3.5 の KServe 0.19.0 (Red Hat ビルド)'.
- System logs: '70枚全カード sweepで約5分は確かに通いですね。', 'recap: カード採用回数に衣装名を併記する修正をしてPR #43を作成済み。', 'INFO: 127.0.0.1:52382 - "POST /api/recommend/stream HTTP/1.1" 200 OK', 'INFO: 127.0.0.1:52386 - "POST /api/solve/stream HTTP/1.1" 200 OK', 'INFO: 127.0.0.1:52393 - "POST /api/solve/stream HTTP/1.1" 200 OK', 'INFO: 127.0.0.1:52385 - "POST /api HTTP/1.1" 404 Not Found', 'INFO: Shutting down', 'INFO: Waiting for application shutdown.', 'INFO: Application shutdown complete.', 'INFO: Finished server process [38659]', '~/gitrepo/holodre.shin main 8h 15m 54s', '10.2.noarch', 'insights-core-selinux-3.8.0-1.el10.2.noarch', 'kernel-core-6.12.0-211.51.1.el10.2.x86\_64', 'kernel-devel-6.12.0-211.51.1.el10.2.x86\_64', 'kernel-modules-6.12.0-211.51.1.el10.2.x86\_64', 'kernel-modules-core-6.12.0-211.51.1.el10.2.x86\_64', 'kernel-modules-extra-6.12.0-211.51.1.el10.2.x86\_64', 'p11-kit-client-0.26.4-1.el10.2.x86\_64', 'podman-sequence-0.4.0-pqc-3-1.el10.2.x86\_64', 'Complete!', 'lockdownctl - Sm 26s', 'upstream KServe 最新 0.20 (+ nightly)', 'RHQAI 3.5 の KServe 0.19.0 (Red Hat ビルド)', 'RHQAI 3.5 は upstream の 1 つ前のリリース (0.19) をベースにしていることになります。Red Hat のバージョンは upstream と同期しているようです。', 'Churned for 24s', 'bypass permissions on (shifftab to cycle) . + for agents', 'new task? /clear to save 153.1k tokens', '### 見出しを全て \*\*太字\*\* に変更しました。<details> 内ではMarkdownの ## がレンダリングされない問題があるためです。これで新行したものが正しく閉じた状態で表示されるはずです。', 'Brewed for 3m 12s', 'recap: AWS EC2上にOpenShift 4.22.55をKVM VMでデプロイする作業は完了済みです。PLAN.md (1ターニング用) と TRAINING.md (学習者向けヒント集) も作成しました。残作業は特ありません。', 'bypass permissions on (shifftab to cycle) . + for agents', 'VSM で整理したが、「ログを見る」では実験できなかった → 従来の自動化とAgent開発の特性の違い。専用APIの導入と、監視・分析・評価・改善まで分岐した → DomainPレンジの自動化アプローチ。Skill を動かすため、LangGraph と OpenCode を試した → AI Agentの開発方法の検討。同じ API 処理を、Trial ごとに作り直していた → 生成AIが利用するツールを用意するモチベーション。複数 Trial の実行内容を追比べ、コード化の境界を探した → Skillにツールの境界線を見極める。Skill と繋いだり、負し直しを調べる方法と体系化はできなかった → 非決定的な Skill と Domain知識が得意な3つの側面。Skill を直しても、本当に良くなったのか判定できなかった → Skillの改善の難しさ。Trial を自動で出して比較できる Runner を作った → 非決定的なSkillを定量的に改善するアプローチ。Skill が自然言語なら、Domain Expert 自身が書けるか。実案件でやってみて、分かったこと。', '8,148-182 23s'.

**主張 : Unixは50年戦ってきた。AIが来ても、勝つのは組み合わせ**

## 生成AI座談会

---

- | AIに任せて、一番うまくいった仕事は？
- | AIに任せて、一番失敗した仕事は？
- | 他人には勧めづらいけど、自分は好きな設定は？

# 明日から試せる生成AIワークハック

今日の話から、明日そのまま試せるものだけを抜き出しました。

## 01 AIをブラウザから出して、PCを操作させる

デスクトップからCLIに置き換えて1日過ごす

## 02 AI専用のペインを常駐させる

ターミナルを1枚増やすだけ。閉じないと決める

## 03 「相談」と「作業」を分ける

考える相手と、手を動かす相手を別セッションにする

## 04 指示は口頭よりファイルで

書いた指示は明日も使える。口頭の指示は消える

## 05 コード生成とレビューでAgentを分ける

コード生成とレビューを両方にやらせるより精度がダンチ

## 06 生成AIの得意、不得意を掴む

生成AIが苦手な仕事を任せて時間を浪費しない！

## 07 やり取りを履歴に残す

リポジトリに入れて検索対象にする。過去の自分が使える

## 08 1週間、相手の派閥で暮らす

GUI派はCLIで、CLI派はGUIで。一番学びが大きい

あなたが会場の皆さんにおすすめしたい生成AIワークハックをぜひ教えてください！

## なんか名言！

---

去年は「この椅子で世界が変わった」だった。今年は「この環境からもう戻れない」だった。

来年もまた、お互いの机を覗きに来てください。

# ありがとうございました



@CloudOpsDaysTokyo



@cloudopsdays

# 椅子選びバッドノウハウ

## 椅子の特殊要件

リモートワークの椅子選び。座面高・可動範囲・肘掛けの3つで、市販品の大半が候補から外れる。

### 01

#### 座面の高さ

35cm がベスト。

机の高さの関係で40cm以上は高すぎ。

一般的なオフィスチェアは最低40cm。

### 02

#### 可動範囲

180度まで倒せるか、折り畳めるか。

どちらかは必須条件。

でかいディスプレイでゲームするため、椅子をしまいたい。

### 03

#### 肘掛け

倒したときに肘掛けが傾かないこと。

座った姿勢では判断できない。

マウステーブルを肘掛けに置いてるので、傾くとマウスが落ちる。

トラックボールはNG(一時期ヘビーユーザ)

**確認の順番は、座面高 → 可動範囲 → 肘掛け。肘掛けは倒して初めて分かる**

座面高で候補が絞れ、可動範囲で仕様が決まる。肘掛けの傾斜はカタログに載らないので実物確認になる。

# 椅子選びバッドノウハウ

## 失敗の遍歴：フェーズ1から3

脱落したのは要件そのものではなく、毎回ちがう見落としだった。

| フェーズ     | 椅子        | 入手                    | 脱落理由                  | そこから引いた学び             |
|----------|-----------|-----------------------|-----------------------|-----------------------|
| 1        | 中華オフィスチェア | 新品・低価格帯               | リクライニング125度。倒しても低かった  | 角度は「可」ではなく実数値で確認する    |
| 2        | ゲーミングチェア  | ヤフオク・中古               | 微妙に壊れてた。毎日じわじわ効く      | 中古は「微妙に壊れている」前提で見積もる  |
| 3 ← いまここ | ゲーミングチェア  | AKRacing + 3rdシリンドー交換 | 蒸れる。合皮に通気性がない         | オフィスチェアで180度リクライニング。。 |
| 特殊       | アーロンチェア   | 試用のみ                  | 180度も折り畳みも不可。要件は満たさない | 腰が伸びる。要件の外に評価軸があった    |

3脚とも要件で落ちたわけではない。角度・コンディション・素材と、確認すべき項目が毎回ひとつ増えている。

## 最新アーロンチェアは、腰が伸びる！

- 特殊要件（180度リクライニング・折り畳み）は満たさないのに、座り続けて腰が伸びる感覚があった
  - 座面高と角度だけを追ってきた前提そのものを疑わせる結果
- 「リクライニング可」は125度でも成立する。180度とは別物と考える
  - 中古は「微妙に壊れてる」前提。差額以上に手間が乗る
- 素材の欠点は季節が変わるまで出ない。次はメッシュ系で探す
  - 次の探索軸：35cm × フルフラットか折り畳み × 通気性のある素材

# AIを操作するツール、あなたはどっち派？

議論の前に、会場の分布を見せてください。声と手をお願いします！

## CALL ①

**「AIをターミナルで動かしてる人ー？」**

RESPONSE 「はい！」 + 挙手  
Claude Code / Codex CLI など

## CALL ②

**「エディタの中で動かしてる人ー？」**

RESPONSE 「はい！」 + 挙手  
VS Code / Cursor など

## CALL ③

**「2体以上を同時に走らせてる人ー？」**

RESPONSE 「はい！」 + 両手  
並列勢は後で捕まえます

**多数派にも少数派にも、必ずマイクを回します**

挙げた手の数で、話を聞く順番を決めます

少数派こそ聞きたい。「自分だけかも」と思ったら、そこが今日いちばんおもしろい話です。

# 争点① CLI vs GUI

同じAIエージェントでも、「どこから叩くか」で運用がまるごと変わる。

## CLI — terminal

- SSH先・サーバの上でもそのまま動く
- セッションを何本でも横に並べられる
- 操作が全部テキスト=再現と共有ができる

▲ 弱点：差分のレビューが目視頼み

VS

## GUI — VS Code

- 差分がその場で色付きで読める
- ファイルツリーで文脈を渡しやすい
- 拡張・デバッガ・Gitと地続き

▲ 弱点：リモートと並列がやりづらい

コール：「ターミナル派ー？」 → レスpons：「はい！」

火種：ターミナル派に聞きたい — AIが書いた差分、本当に全部読んでますか？

## 争点② 直近ツール vs 古参ツール

AIエージェント前提の新顔と、10年以上戦ってきた汎用マルチプレクサ。

### 新顔 — cmux / Orca

- エージェント1体 = 1タブ = 1 worktree
- worktreeの作成も片付けも任せられる
- 差分ビューやブラウザまで同居している
- ▲ 弱点：設定資産も指の記憶もゼロから

VS

### 古参 — tmux / zellij

- 積み上げた設定と指の記憶がそのまま効く
- SSHの向こう側でも操作が変わらない
- AI専用ではない = 何にでも使い回せる
- ▲ 弱点：並列とworktree管理は自分で組む

コール：「cmux / Orca 触った人ー？」 → レスポンス：「はい！」

火種：新顔はどれもGUI寄り。CLI vs GUI の勝負、もう決着してませんか？

# 結局、議論の軸はこの4つ

ツールの好みではなく、この4つのどこを取るかの話に落ちる。

## 01

### 並列性

何体を同時に走らせるか。  
1体を丁寧に見るか。

## 02

### レビュー

AIの差分をどこで読むか。  
git diff か、エディタか。

## 03

### 環境

手元だけで足りるか。  
SSHの向こうでも同じか。

## 04

### 乗り換え

設定やキーバインドの  
資産を捨てられるか。

**あなたがゆずれない軸はどれ？ 指の本数（1～4）で挙げてください**

一番多かった軸から、パネルで掘っていきます

4つ全部を満たすツールは、たぶん今はまだありません。だから使い分けの話になります。

## ツール選びは「並列性」と「レビュー」で決まる

- CLI か GUI かは目的ではなく、何体を・どこで・どうレビューするかの結果
- 新顔 (cmux / Orca) は並列性が強み。古参 (tmux / zellij) は再現性が強み
- 両方に指を置いておくのが、いまのところ一番つぶしが効く
- 迷ったら「1体を丁寧に」から。並列は運用が回りはじめてから増やす

**最後のコール：「今日帰って、新しいツール入れる人ー？」**

レスポンス：「はい！」 — 入れたら #CODT2026 で結果を教えてください