SHADOW SUITE · PAPER 1.26.2

ShadowCommand

Manuale dell'amministratore

Comandi personalizzati, menu, trigger e automazioni scritti in YAML. Se sai scrivere un file di configurazione, sai programmare il tuo server.

84	47	158	2	0
AZIONI	REQUISITI	SEGNAPOSTO	LINGUE	DIPENDENZE OBBLIGATORIE

Versione 1.0.0 · per Paper / Purpur / Folia 1.26.2 · Java 25
Ogni esempio di questo manuale e' pronto da incollare in un file e usare.

Indice

1. Cos'e' ShadowCommand

Un motore per costruire il tuo server scrivendo file YAML: comandi, menu, automazioni, premi, condizioni. Niente Java, niente ricompilare, niente riavviare.

Ogni server ha bisogno di comandi che non esistono: `/discord`, `/kit`, `/regole`, un menu di teletrasporto, un annuncio ogni dieci minuti, un premio al primo accesso. ShadowCommand serve a scriverli in pochi minuti, senza toccare una riga di codice.

Il comando piu' corto possibile

File `plugins/ShadowCommand/commands/base.yml`

```
discord:
  run:
    - "message: &bEntra nel Discord:
<click:open_url:'https://discord.gg/iltuoserver'>&f&ndiscord.gg/iltuoserver</click>"
```

Tre righe e `/discord` esiste davvero: registrato nel server, con il suo permesso, con il completamento automatico del client, con i colori. Da qui in avanti e' tutta questione di aggiungere righe.

Le funzionalita', in breve

Comandi personalizzati

Alias, descrizione, uso, permesso automatico, sotto-comandi annidati, argomenti tipizzati con completamento vero (giocatori, mondi, materiali, scelte), comandi da console, comandi riservati, comandi nascosti.

Un vero linguaggio

`if` / `elseif` / `else`, `while`, `repeat`, `for`, `break`, funzioni riutilizzabili con parametri e valore di ritorno, attese che non bloccano nulla, blocchi asincroni.

84 azioni

Messaggi, titoli, bossbar, libri, suoni, particelle, effetti, teletrasporti, inventario, denaro, permessi, gruppi, mondo, meteo, variabili, GUI, cooldown, e le azioni dedicate agli altri plugin della suite.

47 requisiti

Permesso, denaro, livello, oggetto in mano, mondo, regione, bioma, altezza, ora reale, meteo, distanza, variabile, espressione libera, probabilita', piu' quelli del battle pass e dei minigiochi.

GUI complete

Menu a piu' pagine, icone che cambiano aspetto in base a una condizione, contenuti dinamici (giocatori, mondi, liste), animazioni, aggiornamento dal vivo senza sfarfallio, menu che accettano oggetti, menu annidati con il tasto indietro.

Automazioni

Trigger su circa 30 eventi di gioco, cartelli cliccabili, ingressi e uscite dalle regioni, schedulazioni `every` / `at` / `cron`, e un trigger che ascolta l'evento di **qualunque** altro plugin installato.

Database vero

H2 locale al primo avvio, senza configurare niente. Metti le credenziali MySQL e il plugin crea da solo tutte le tabelle: variabili e cooldown diventano condivisi fra piu' server dello stesso network.

Cooldown che non si perdono

Un kit da 24 ore resta da 24 ore anche se il server riparte dieci volte. La scrittura avviene subito, non al salvataggio periodico: un crash non regala un secondo utilizzo.

Conferme a pulsanti

Nessuna conferma si scrive a mano. La domanda arriva in chat con due pulsanti cliccabili, con un codice monouso che nessun altro puo' usare al posto tuo.

Multilingua

Italiano e inglese inclusi, un file per lingua, e ogni giocatore vede la lingua del proprio client. Aggiungere una lingua vuol dire copiare una cartella e tradurla.

Diagnostica

Gli errori si scoprono all'avvio, non in gioco: file, riga e suggerimento. Un comando con errori non viene registrato. `/scmd debug` esegue passo passo mostrando i valori veri di ogni segnaposto.

Migrazione da MyCommand

`/scmd import mycommand` converte i comandi esistenti, traduce `$player` e `$$if` nella sintassi nuova e scrive un rapporto di cosa e' passato e cosa va guardato a mano.

Cosa serve per usarlo

Requisito	Dettaglio
Server	Paper, Purpur o Folia **1.26.2**
Java	Java 25 o superiore
Database	Nessuno: H2 locale incluso. MySQL / MariaDB opzionale
Altri plugin	**Nessuno obbligatorio.** Vedi il capitolo 12 per cosa si guadagna installando gli opzionali
Licenza	La chiave ricevuta con l'acquisto, da incollare in `config.yml`

Come leggere questo manuale

I capitoli 3, 4 e 5 sono un ricettario: per ogni tipo di cosa che puoi creare trovi un esempio **facile**, pronto da incollare, e dove ha senso anche uno **avanzato** che mostra fin dove si puo' arrivare. I capitoli dal 6 in poi sono il riferimento da consultare quando serve.

2. Installazione e primo avvio

Cinque minuti, di cui quattro sono il riavvio del server.

1. Metti il jar

Copia `ShadowCommand-1.0.0.jar` nella cartella `plugins/` e avvia il server. Al primo avvio il plugin crea la sua cartella, tutti i file di configurazione, i due file di lingua, il database locale e qualche esempio funzionante.

2. Incolla la licenza

Apri `plugins/ShadowCommand/config.yml`: la sezione della licenza e' la prima.

```yaml
licence:
  key: "SHADOW-la-tua-chiave-qui"
  # Giorni di tolleranza dopo la scadenza di un abbonamento. 0 per non concederne.
  grace-days: 3
```

Riavvia. In console deve comparire la riga della licenza dentro il riquadro di avvio:

```
+-- ShadowCommand 1.0.0 --------------------
|  Server     26.2 | Java 25.0.4.1 | Folia: no
|  Licenza    standard - Nome Cliente
|  Storage    H2 file locale (data/commands.mv.db)
|  Comandi    9 | trigger 13 | menu 1 | schedulazioni 4 | funzioni 3
|  Vocabolario 84 azioni, 47 requisiti, 158 segnaposto
|  Economia   ShadowEconomy
|  Regioni    ShadowRegions
|  Lingue     2 | predefinita it_IT
|  Problemi   nessuno
|  Avvio      435 ms
+---------------------------------------------
```

Se la licenza non va

Il plugin non parte e scrive in console il motivo esatto e l'id di questo server, che serve per l'assistenza.

Non tocca nulla

: file, mondi e database restano esattamente come erano. Dopo aver incollato la chiave serve un riavvio, non un `/scmd reload`.

3. Guarda cosa c'e' dentro

```
plugins/ShadowCommand/
├── config.yml            impostazioni generali e licenza
├── overrides.yml         blocco, intercettazione e alias di comandi altrui
├── schedules.yml         comandi a tempo
├── panel.yml             aspetto del pannello /scmd
├── commands/             i tuoi comandi (quanti file vuoi, anche in sottocartelle)
│   └── esempi.yml
├── triggers/             gli script che partono dagli eventi
│   ├── esempi.yml
│   └── suite.yml         esempi per gli altri plugin della suite (spenti)
├── menus/                le GUI
│   └── esempio.yml
├── functions/            funzioni riutilizzabili
│   └── esempi.yml
├── lang/
│   ├── it_IT.yml
│   └── en_US.yml
└── data/commands.mv.db   database locale
```

Il nome dei file dentro `commands/`, `triggers/` e `menus/` non conta: serve solo a te per ritrovarli. Puoi tenere un file unico o farne venti, e puoi usare le sottocartelle.

4. Il tuo primo comando

Crea `commands/miei.yml` e scrivi:

```yaml
sito:
  description: "Il sito del server"
  run:
    - "message: &7Il nostro sito: &b www.ilmioserver.it"
    - "sound: entity.experience_orb.pickup"
```

In gioco: `/scmd reload` e prova `/sito`. Non serve riavviare.

5. Se arrivi da MyCommand

```
/scmd import mycommand
```

Legge `plugins/MyCommand/commands/`, converte i comandi, traduce i segnaposto `$player` e le condizioni `$$if` nella sintassi nuova, e scrive un rapporto di cosa e' passato pulito e cosa conviene guardare a mano. Con `--dry` mostra il risultato senza scrivere niente, con `--disabled` importa tutto spento.

Il database: quando conviene MySQL

Di fabbrica il plugin usa un database H2 dentro la sua cartella e non devi fare niente. Passa a MySQL solo se hai **piu' server** che devono condividere variabili e cooldown (un lobby e due survival, per esempio).

```yaml
storage:
  mysql:
    enabled: true
    host: "127.0.0.1"
    port: 3306
    database: "shadow"
    user: "shadow"
    password: "..."
    table-prefix: "sc_"
```

Le tabelle le crea il plugin da solo al primo avvio: non devi importare nessuno script SQL. Se il database non risponde, il plugin lo scrive e **non parte**, invece di partire a meta' e perdere i dati dei giocatori.

I colori

I colori si scrivono come si e' sempre fatto, con la e commerciale: `&a` verde, `&c` rosso, `&l` grassetto, `&#FF8800` per un colore esatto. In piu' funzionano i gradienti e un paio di comportamenti del testo:

```yaml
- "message: &6Ciao &e{player}&6, benvenuto!"
- "message: &#FF8800Arancione esatto"
- "message: <gradient:#ff0000:#ffaa00>Titolo sfumato</gradient>"
- "message: <rainbow>Arcobaleno</rainbow>"
- "message: <hover:show_text:'&7Clicca!'><click:run_command:'/kit'>&a[ RITIRA IL KIT ]</click>
</hover>"
```

> **Per scrivere una e commerciale vera**
> Raddoppiala: `&&` diventa un semplice `&` nel messaggio.

3. Ricettario: i comandi

Ogni tipo di comando che puoi creare, con un esempio pronto da incollare. Dove la cosa si presta, c'e' anche la versione avanzata.

Tutti i comandi vivono in `plugins/ShadowCommand/commands/`. Il nome del file non conta. Dopo ogni modifica: `/scmd reload`.

> **L'unica chiave obbligatoria**
> Un comando ha bisogno solo di `run:` (oppure di `menu:`, oppure di `text:`). Tutto il resto - permessi, alias, descrizione, cooldown - ha un valore di fabbrica sensato e si aggiunge solo quando serve.

3.1 Comando di solo testo FACILE

Il caso piu' comune in assoluto: `/regole`, `/discord`, `/staff`, `/vote`. Serve a stampare delle righe e basta.

```yaml
# commands/informazioni.yml

regole:
  description: "Le regole del server"
  aliases: [ "rules", "regolamento" ]
  text:
    - "&8&m                                        "
    - "&6&lREGOLE DEL SERVER"
    - ""
    - "&e1. &7Rispetta gli altri giocatori."
    - "&e2. &7Niente cheat, niente client modificati."
    - "&e3. &7Non chiedere oggetti allo staff."
    - "&e4. &7Divertiti."
    - "&8&m                                        "
```

`text:` e' una scorciatoia per una serie di `message:`. Una riga vuota resta vuota. Puoi usare i segnaposto: `&7Ciao {player}, sei online da {playtime}`.

Variante: con i pulsanti

Se una riga deve essere cliccabile, si scrive cosi'. Il giocatore clicca invece di copiare.

```yaml
discord:
  text:
    - "&bSiamo anche su Discord."
    - "<click:open_url:'https://discord.gg/xxx'><hover:show_text:'&7Apre il browser'>&f&n[ Entra nel server Discord ]</hover></click>"
```

3.2 Comando che esegue altri comandi FACILE

Un comando che ne lancia altri: i tuoi, quelli di Essentials, quelli della console. E' il mattone con cui si costruisce quasi tutto il resto.

```yaml
# commands/utili.yml

spawn:
  description: "Torna allo spawn"
  run:
    - "message: &7Ti riporto allo spawn..."
    - "console: minecraft:tp {player} 0 100 0"
    - "sound: entity.enderman.teleport"
```

Azione	Chi esegue il comando
`command:`	Il giocatore, con i suoi permessi
`console:`	La console, quindi con tutti i permessi
`op:`	Il giocatore, con l'operatore concesso per un istante
`bypass:`	Il giocatore, con **un solo** permesso concesso per un istante
`runas:`	Un altro giocatore, indicato da te

Versione avanzata AVANZATO

Un `/premio` per lo staff che da' un rango a un altro giocatore, lo annuncia, lo registra e non richiede che chi lo usa sia operatore.

```yaml
premio:
  description: "Assegna un rango premio a un giocatore"
  permission: "server.staff.premio"
  arguments:
    - { name: "chi", type: player, required: true }
    - { name: "rango", type: choice, values: [ "vip", "vip+", "mvp" ], required: true }
  confirm: true
  confirm-text: "&7Stai per dare &e{arg2} &7a &f{arg1}&7."
  run:
    # bypass: concede un permesso solo per questa riga, poi lo toglie
    - "bypass: luckperms.user.parent.add | lp user {arg1} parent add {arg2}"
    - "tell: {arg1} &6Complimenti! &7Hai ricevuto il rango &e{arg2}&7."
    - "broadcast: &e{arg1} &7e' ora &6{arg2}&7!"
    - "runas: {arg1} me festeggia!"
    - "log: [premi] {player} ha dato {arg2} a {arg1}"
    - "var-of: {arg1} set rango {arg2}"
```

> **Perche' `bypass:` e non `op:`**
>
> `op:` da' **tutti** i permessi per un istante: se il comando che lanci ha un effetto collaterale, lo ha con i poteri massimi. `bypass:` concede solo il permesso che scrivi tu. Entrambi rimettono le cose com'erano anche se il comando fallisce, ed entrambi si possono limitare o spegnere in `config.yml`.

3.3 Comando con argomenti FACILE

Gli argomenti sono **tipizzati**: il plugin controlla che siano validi prima di eseguire, e il client completa da solo mentre il giocatore scrive.

```yaml
saluta:
  description: "Saluta un giocatore online"
  arguments:
    - name: "chi"
      type: player            # completa con i giocatori online
      required: true
  run:
    - "tell: {arg1} &e{player} &7ti saluta!"
    - "message: &7Saluto mandato a &f{arg1}&7."
```

Se scrivi `/saluta Pippo` e Pippo non c'e', il comando non parte e il giocatore riceve un messaggio chiaro. Non devi controllarlo tu.

I tipi disponibili

type	Accetta	Completamento del client
`word`	una parola	niente
`text`	tutto il resto della riga	niente
`int`	numero intero, con `min` e `max`	niente
`number`	numero con decimali	niente
`bool`	true/false, si/no	i due valori
`choice`	uno dei `values`	i valori
`player`	giocatore online	i nomi online
`offline-player`	qualunque giocatore conosciuto	i nomi visti
`world`	mondo caricato	i mondi
`material`	materiale valido	i materiali
`command`	un comando di ShadowCommand	i tuoi comandi
`variable`	nome di variabile	le variabili esistenti

Versione avanzata AVANZATO

Un `/kit` con scelta obbligatoria, quantita' facoltativa con limiti, messaggi d'errore su misura e contenuto diverso per ogni scelta.

```yaml
kit:
  description: "Ritira il tuo kit"
  usage: "/kit <legno|ferro|diamante> [quantita]"
  aliases: [ "kits", "starter" ]

  arguments:
    - name: "tipo"
      type: choice
      values: [ "legno", "ferro", "diamante" ]
      required: true
      error: "&cScegli fra &elegno&c, &eferro&c e &ediamante&c."
    - name: "quantita"
      type: int
      min: 1
      max: 5
      default: 1
      error: "&cLa quantita' va da 1 a 5."

  # il permesso viene composto con l'argomento: server.kit.diamante
  requirements:
    - "permission: server.kit.{arg1}"

  run:
    - "message: &aEcco il kit &e{arg1} &7(x{arg2})."
    - "repeat: {arg2}"
    - "  if: {arg1} == diamante"
    - "    give: DIAMOND_SWORD 1 name:'&bSpada del kit' enchant:SHARPNESS:3 unbreakable"
    - "    give: DIAMOND_HELMET 1"
    - "  elseif: {arg1} == ferro"
    - "    give: IRON_SWORD 1 name:'&fSpada di ferro'"
    - "  else:"
    - "    give: WOODEN_SWORD 1"
    - "  end"
    - "end"
    - "var: add kit_ritirati {arg2}"
    - "sound: entity.player.levelup"
```

Come si leggono gli argomenti

Segnaposto	Vale
`{arg1}` `{arg2}`	il primo, il secondo, ...
`{arg:tipo}`	per nome, se l'argomento e' dichiarato
`{args}`	tutto quello che e' stato scritto
`{args:2-}`	dal secondo in poi (comodo per i motivi e i messaggi)
`{args:2-4}`	dal secondo al quarto
`{argslength}`	quanti ne sono stati scritti
`{arg2|nessuno}`	il secondo, oppure `nessuno` se manca
`{label}`	con quale alias e' stato scritto il comando

3.4 Comando con sotto-comandi FACILE

Quando un comando ha piu' funzioni: `/clan crea`, `/staff warn`, `/casa vai`. Ogni figlio ha permesso, argomenti e script propri, e il client li completa.

```yaml
staff:
  description: "Strumenti dello staff"
  permission: "server.staff"
  run:
    - "message: &7Usa &e/staff lista &7o &e/staff warn <chi> <motivo>&7."
  subcommands:
    lista:
      run:
        - "message: &7Online adesso: &f{server_online}&7/&f{server_max}"
        - "message: &7Nomi: &f{server_online_names}"
```

Il permesso dei figli si compone da solo

Se non scrivi `permission:` in un sotto-comando, riceve `permesso del padre` + `.` + `nome del figlio`. Qui `/staff lista` chiede `server.staff.lista`.

Versione avanzata AVANZATO

Un sistema di clan completo con quattro sotto-comandi, argomenti tipizzati, permessi separati e variabili condivise fra i giocatori.

```yaml
clan:
  description: "Gestione del tuo clan"
  aliases: [ "gilda" ]
  player-only: true
  run:
    - "if: {var:clan|nessuno} == nessuno"
    - "  message: &7Non sei in nessun clan. Creane uno con &e/clan crea <nome>&7."
    - "else:"
    - "  message: &6Il tuo clan: &e{var:clan} &8(&7{var:clan_ruolo}&8)"
    - "end"

  subcommands:

    crea:
      arguments:
        - { name: "nome", type: word, required: true }
      requirements:
        - "condition: {var:clan|nessuno} == nessuno"   # non deve gia' averne uno
        - "money: >= 5000"
      cost:
        money: 5000
      run:
        - "var: set clan {arg1}"
        - "var: set clan_ruolo capo"
        - "svar: set clan_{arg1}_capo {player}"
        - "svar: set clan_{arg1}_membri 1"
        - "broadcast: &6E' nato il clan &e{arg1}&6, fondato da &f{player}&6!"

    invita:
      arguments:
        - { name: "chi", type: player, required: true }
      requirements:
        - "variable: clan_ruolo == capo"
        - "player-online: {arg1}"
      run:
        - "tell: {arg1} &6{player} &7ti invita nel clan &e{var:clan}&7."
        - "tell: {arg1} <click:run_command:'/clan accetta {var:clan}'>&a[ ACCETTA ]</click>
<click:run_command:'/clan rifiuta'>&c[ RIFIUTA ]</click>"
        - "message: &7Invito mandato a &f{arg1}&7."

    accetta:
      arguments:
        - { name: "nome", type: word, required: true }
      requirements:
        - "condition: {var:clan|nessuno} == nessuno"
      run:
        - "var: set clan {arg1}"
        - "var: set clan_ruolo membro"
        - "svar: add clan_{arg1}_membri 1"
        - "broadcast: &e{player} &7si e' unito al clan &6{arg1}&7."

    esci:
      confirm: true
      confirm-text: "&7Stai per uscire dal clan &e{var:clan}&7."
      run:
        - "svar: sub clan_{var:clan}_membri 1"
        - "message: &7Hai lasciato il clan &e{var:clan}&7."
```

```
      - "var: delete clan"
      - "var: delete clan_ruolo"
```

I sotto-comandi si annidano quanto vuoi: un figlio puo' avere a sua volta i suoi `subcommands:`.

3.5 Comando con requisiti, costi e attese FACILE

Il caso classico: un premio che si puo' ritirare una volta al giorno.

```
giornaliero:
  description: "Il premio quotidiano"
  cooldown: "24h"
  cooldown-message: "&cHai gia' ritirato il premio. Torna fra &e{cooldown_left}&c."
  run:
    - "money: give 500"
    - "give: DIAMOND 1"
    - "message: &aPremio ritirato! &7Torna domani."
    - "sound: entity.player.levelup"
```

Il cooldown sopravvive al riavvio

Di fabbrica finisce nel database e viene riletto all'avvio: un premio da 24 ore resta da 24 ore anche se il server riparte dieci volte. La riga viene scritta **subito**, non al salvataggio periodico, quindi nemmeno un crash regala un secondo ritiro. Con MySQL vale su tutti i server che condividono il database.

Versione avanzata AVANZATO

Un teletrasporto a casa che costa, richiede condizioni, chiede conferma a pulsanti, ha un'attesa durante la quale non ci si puo' muovere, e reagisce in modo diverso a ogni tipo di rifiuto.

```yaml
casa:
  description: "Torna a casa tua"
  player-only: true

  # 1. Le condizioni. Se una non passa, non parte niente.
  requirements:
    - "world: !world_the_end"
    - type: condition
      value: "{var:casa_x|vuoto} != vuoto"
      deny: "&cNon hai ancora impostato una casa. Usa &e/setcasa&c."
    - type: condition
      value: "{health} > 4"
      deny: "&cSei troppo ferito per teletrasportarti."
      sound: "block.note_block.bass"
  requirements-mode: all           # all (di fabbrica) | any | none

  # 2. La conferma, con due pulsanti cliccabili in chat
  confirm: true
  confirm-text: "&7Costo: &e250 monete&7. Vuoi procedere?"

  # 3. L'attesa prima di partire
  warmup: "5s"
  warmup-bossbar: true
  warmup-cancel-on-move: true
  warmup-cancel-on-damage: true
  warmup-message: "&7Non muoverti per &e{warmup}&7..."
  warmup-cancelled: "&cTeletrasporto annullato: ti sei mosso."

  # 4. Il prezzo. O si paga tutto, o non si preleva niente.
  cost:
    money: 250
    message: "&7Hai pagato &e{cost_money} monete&7."

  # 5. L'attesa fra un uso e l'altro
  cooldown: "10m"
  cooldown-bypass: "server.casa.nocooldown"
  cooldown-persistent: true

  run:
    - "teleport: {var:casa_x} {var:casa_y} {var:casa_z} {var:casa_mondo}"
    - "message: &aBentornato a casa."
    - "particle-at-player: PORTAL 60"

  # Code alternative: cosa fare quando qualcosa nega
  on-deny:
    - "sound: block.note_block.bass"
  on-cooldown:
    - "message: &cAncora &e{cooldown_left}&c prima di poter tornare a casa."
  on-error:
    - "message: &cQualcosa e' andato storto, riprova."
    - "money: give 250"
```

L'ordine esatto dei controlli

Serve saperlo per capire chi paga e chi no: **un warmup non parte mai se non puoi pagare**, e **il cooldown si mette solo dopo un'esecuzione riuscita**.

```
1. il comando esiste ed e' acceso
2. visibilita' nel completamento
3. permesso
4. giocatore o console
5. argomenti: numero, tipo, obbligo
6. requisiti
7. cooldown
8. conferma
9. warmup
10. costi
11. run
```

3.6 Comando con logica FACILE

Un comando che si comporta in modo diverso a seconda di com'e' messo il giocatore.

```yaml
vita:
  run:
    - "if: {health} >= 18"
    - "  message: &aStai benissimo. &7({health}/{maxhealth})"
    - "elseif: {health} >= 8"
    - "  message: &eSei un po' malmesso. &7({health}/{maxhealth})"
    - "else:"
    - "  message: &cSei messo male! &7({health}/{maxhealth})"
    - "  effect: REGENERATION 10 1"
    - "end"
```

Versione avanzata AVANZATO

Una cassa dei premi: un'animazione che gira, una probabilita' diversa per ogni premio, una funzione riutilizzabile, un annuncio solo per i premi rari e un contatore persistente.

```yaml
# functions/premi.yml
functions:
  consegna:
    args: [ "oggetto", "quanti", "nome" ]
    run:
      - "give: {oggetto} {quanti}"
      - "message: &aHai vinto: &e{nome} &7x{quanti}"
      - "var: add premi_vinti 1"
      - "return: {nome}"
```

```yaml
# commands/cassa.yml
cassa:
  description: "Apri una cassa dei premi"
  cooldown: "1h"
  cost:
    items:
      - "TRIPWIRE_HOOK 1"          # la chiave della cassa

  run:
    # --- animazione: cinque giri sempre piu' lenti -------------------
    - "repeat: 5"
    - "  actionbar: &7Estrazione in corso &e{rand:mela,ferro,oro,diamante,smeraldo}"
    - "  sound: block.note_block.hat"
    - "  wait: {math:{loop}*4}t"
    - "end"
    - "sound: ui.toast.challenge_complete"

    # --- il premio: si scende di rarita' finche' uno non passa -------
    - "chance: 2"
    - "  call: consegna NETHER_STAR 1 'Stella del Nether'"
    - "  broadcast: &d{player} &7ha estratto &d{result}&7 dalla cassa!"
    - "  stop"
    - "end"
    - "chance: 10"
    - "  call: consegna DIAMOND 8 'Otto diamanti'"
    - "  broadcast: &b{player} &7ha estratto &b{result}&7!"
    - "  stop"
    - "end"
    - "chance: 35"
    - "  call: consegna IRON_INGOT 16 'Sedici lingotti di ferro'"
    - "  stop"
    - "end"
    - "money: give {rand:50-250}"
    - "message: &7Premio di consolazione: &a{result|monete}&7."

    # --- ogni dieci casse, un bonus ----------------------------------
    - "if: {math:{var:premi_vinti} % 10} == 0"
    - "  message: &6Decima cassa: bonus!"
    - "  give: EXPERIENCE_BOTTLE 16"
    - "end"
```

Costrutto	Cosa fa
`if / elseif / else / end`	La condizione. `end` chiude sempre l'ultimo blocco aperto
`repeat: 5 ... end`	Ripete 5 volte. Dentro, `{loop}` vale 1, 2, 3...
`while: cond ... end`	Ripete finche' la condizione e' vera
`for: x in lista ... end`	Un giro per ogni elemento, in `{x}`
`chance: 10 ... end`	Esegue il blocco il 10% delle volte
`call: nome args`	Chiama una funzione; il valore torna in `{result}`
`wait: 3s`	Sospende senza bloccare nulla: non e' un `sleep`
`stop`	Termina lo script qui

3.7 Comando che apre una GUI FACILE

Se il comando serve solo ad aprire un menu, salti del tutto lo script.

```yaml
negozio:
  menu: "negozio"                # apre menus/negozio.yml
  permission: "server.negozio"
  description: "Apre il negozio del server"
```

Equivale a `run: [ "menu: negozio" ]`, ma il pannello lo riconosce come comando di menu e ti porta direttamente all'editor della GUI. Il contrario funziona altrettanto bene: nel file del menu puoi scrivere `open-command: "negozio"` e il comando nasce da solo.

Come si costruisce un menu e' tutto nel **capitolo 5**.

3.8 Comando che parla con gli altri plugin FACILE

Denaro, permessi, regioni, battle pass, minigiochi: se il plugin c'e', lo script lo usa; se non c'e', la riga non fa niente e lo scrive una volta in console.

```yaml
saldo:
  run:
    - "message: &7Hai &a{money_formatted}&7."
    - "if: {money} >= 10000"
    - "  message: &6Sei ricco!"
    - "end"
```

Versione avanzata AVANZATO

Una missione giornaliera che paga in monete, fa salire il battle pass, richiede di essere in una certa regione e tiene conto del minigioco in corso.

```yaml
missione:
  description: "Consegna la missione del giorno"
  cooldown: "20h"

  requirements:
    - "region: villaggio"                # ShadowRegions o WorldGuard
    - "battlepass-tier: >= 5"            # ShadowBattlePass
    - "in-game: false"                   # ShadowGames: non durante una partita
    - "item: EMERALD 16"

  cost:
    items:
      - "EMERALD 16"

  run:
    - "message: &6Missione consegnata!"
    - "money: give 2500 coins"                    # ShadowEconomy, multi-valuta
    - "battlepass: xp 400 missione-giornaliera"   # con i moltiplicatori attivi
    - "battlepass: event consegna-smeraldi"       # fa avanzare le missioni del pass
    - "permission: add server.mercante 24h"       # permesso a tempo (LuckPerms)
    - "game: coins give 100"                      # monete dei minigiochi

    - "if: {battlepass_premium} == true"
    - "  message: &d+50% bonus premium."
    - "  battlepass: xp 200 bonus-premium"
    - "end"

    - "broadcast: &e{player} &7ha completato la missione del giorno &8(&f{battlepass_tier}&8)"
```

Uno script scritto per la suite completa gira anche a meta'

I segnaposto dei plugin assenti rispondono `0`, `false` o vuoto invece di rompersi; le azioni non fanno niente. I **requisiti**, invece, rispondono **falso**: un requisito serve a fermare qualcosa, quindi quando non sa decidere ferma. Un server senza battle pass non regala il premio che avevi messo dietro al livello 20.

4. Ricettario: le automazioni

Script che partono da soli: quando succede qualcosa, oppure a orario. Vivono in `triggers/` e in `schedules.yml`.

4.1 Trigger su evento FACILE

Il benvenuto al primo accesso: si scrive una volta e vale per sempre.

```yaml
# triggers/benvenuto.yml
config-version: 1

triggers:
  primo-accesso:
    event: player-join
    first-join-only: true
    run:
      - "broadcast: &6Benvenuto a &e{player}&6, e' il suo primo accesso!"
      - "give: BREAD 16"
      - "money: give 500"
      - "title: &6Benvenuto"
      - "subtitle: &7Leggi le regole con /regole"
```

Versione avanzata AVANZATO

Un premio per chi trova i diamanti, ma solo in un mondo, solo con un permesso, solo una volta ogni cinque secondi e solo una volta su quattro. Con un annuncio a tutto il server quando la fortuna e' davvero grossa.

```yaml
triggers:
  minatore:
    event: block-break
    block: "DIAMOND_ORE,DEEPSLATE_DIAMOND_ORE"   # filtro dell'evento
    world: "world"
    priority: NORMAL                             # LOWEST .. MONITOR
    cancel: false                                # true = annulla la rottura
    cooldown: "5s"
    conditions:                                  # gli stessi requisiti dei comandi
      - "permission: server.minatore"
      - "y: < 16"
    run:
      - "money: give 50"
      - "actionbar: &a+50 monete &7(diamante a y={block_y})"
      - "var: add diamanti_trovati 1"
      - "chance: 5"
      - "  give: DIAMOND_BLOCK 1"
      - "  broadcast: &b{player} &7ha trovato un filone eccezionale!"
      - "end"
      - "if: {var:diamanti_trovati} == 100"
      - "  broadcast: &e{player} &7ha estratto il suo &bcentesimo &7diamante!"
      - "  permission: add server.minatore.esperto"
      - "end"
```

Gli eventi disponibili

event	Quando scatta	Segnaposto in piu'

`player-join`	entra nel server	`{first_join}`
`player-first-join`	solo al primo accesso	
`player-quit`	esce	
`player-death`	muore	`{death_cause}`, `{killer}`, `{death_message}`
`player-kill`	uccide un altro giocatore	`{victim}`
`player-respawn`	rinasce	`{respawn_bed}`
`player-world-change`	cambia mondo	`{from_world}`, `{to_world}`
`player-teleport`	si teletrasporta	`{from_x}`, `{to_x}`, ...
`player-level-up`	sale di livello	`{old_level}`, `{new_level}`
`player-chat`	scrive in chat	`{message}`
`player-command`	scrive un comando	`{command}`
`player-damage`	subisce danno	`{damage}`, `{damage_cause}`
`player-move-block`	cambia blocco, non ogni pixel	`{from_x}`, ...
`player-sneak`	si abbassa	`{sneaking}`
`player-item-held`	cambia oggetto in mano	`{item}`
`player-consume`	mangia o beve	`{item}`
`player-fish`	pesca	`{catch}`
`player-craft`	costruisce	`{item}`
`player-drop` / `player-pickup`	lascia / raccoglie	`{item}`, `{amount}`
`player-bed-enter`	va a dormire	
`player-portal`	entra in un portale	`{portal_type}`
`block-break` / `block-place`	rompe / piazza un blocco	`{block}`, `{block_x}`, `{block_y}`, `{block_z}`
`block-interact`	click su un blocco	`{block}`, `{hand}`, `{action}`
`sign-click`	click su un cartello	`{line1}` .. `{line4}`
`item-click`	click con un oggetto in mano	`{item}`, `{action}`
`entity-click`	click su un'entita'	`{entity}`, `{entity_name}`
`region-enter` / `region-exit`	entra / esce da una regione	`{region}`
`server-start`	ad avvio completato	
`server-stop`	allo spegnimento (script brevi: niente attese)	
`plugin-reload`	dopo `/scmd reload`	
`server-ping`	qualcuno guarda il server dalla lista	

`bukkit-event`	l'evento di qualunque altro plugin (vedi 4.5)	tutte le proprieta' dell'evento

> **Un listener che non serve non esiste**
> I listener vengono registrati solo se almeno un trigger li usa. Se nessuno usa `player-move-block`, quel listener non viene nemmeno creato e non costa niente. `/scmd triggers` mostra quali sono attivi e quante volte sono scattati.

4.2 Cartelli cliccabili FACILE

Un cartello che apre il negozio. Il confronto sulla prima riga ignora colori e maiuscole, e il plugin colora il cartello da solo quando viene piazzato.

```yaml
triggers:
  cartello-negozio:
    event: sign-click
    line1: "[Negozio]"
    create-permission: "server.staff"     # solo lo staff puo' scriverlo
    run:
      - "menu: negozio"
```

Le quattro righe del cartello sono disponibili come `{line1}` ... `{line4}`, quindi un solo trigger puo' servire cento cartelli diversi:

```yaml
triggers:
  cartello-warp:
    event: sign-click
    line1: "[Warp]"
    run:
      - "message: &7Ti porto a &e{line2}&7..."
      - "console: minecraft:tp {player} {line3}"
```

4.3 Comandi a tempo FACILE

L'annuncio periodico, il classico dei classici. Sta in `schedules.yml`.

```yaml
# schedules.yml
config-version: 1

schedules:
  annuncio:
    every: "10m"
    run:
      - "broadcast: &7[&bInfo&7] &fVisita il nostro sito: &bwww.ilmioserver.it"
```

Versione avanzata AVANZATO

Quattro schedulazioni diverse: a orario fisso, con espressione cron, una per ogni giocatore online e una che gira fuori dal thread di gioco.

```yaml
schedules:

  # tutti i giorni alle 4 del mattino, ora reale del server
  backup-notturno:
    at: "04:00"
    run:
      - "console: save-all"
      - "log: backup fatto"

  # ogni ora esatta: minuto ora giorno mese giorno-della-settimana
  stipendio:
    cron: "0 * * * *"
    for-each-player: true          # una volta per ogni giocatore online
    conditions:
      - "permission: server.stipendio"
      - "world: !world_the_end"
    run:
      - "money: give 100"
      - "actionbar: &a+100 monete &7(stipendio orario)"

  # ogni mezz'ora, fuori dal thread principale
  pulizia:
    every: "30m"
    async: true
    run:
      - "console: minecraft:kill @e[type=item]"
      - "broadcast: &7Gli oggetti a terra sono stati puliti."

  # ogni due ore, ma anche subito all'avvio del server
  meteo:
    every: "2h"
    on-start: true
    run:
      - "weather: clear 6000 world"
```

Chiave	Significato
`every`	Ogni tot: `30s`, `10m`, `2h`, `1d`
`at`	A un'ora precisa, ogni giorno
`cron`	Espressione cron a cinque campi
`on-start`	Esegue anche subito all'avvio
`for-each-player`	Ripete per ogni giocatore online, con `{player}` valorizzato
`async`	Fuori dal thread di gioco (solo azioni sicure)
`enabled`	Spegne la schedulazione senza cancellarla

`/scmd schedules` elenca tutto con il prossimo orario, permette di eseguire subito una schedulazione per provarla e di accenderla o spegnerla.

4.4 Bloccare e sostituire comandi altrui FACILE

Nascondere `/plugins` ai curiosi. Sta in `overrides.yml`.

```yaml
# overrides.yml
block:
  - command: "plugins"
    aliases-too: true                    # blocca anche /pl e /?
    bypass-permission: "server.vedipl"
    message: "&cComando non disponibile."
```

Versione avanzata AVANZATO

Bloccare, intercettare e rinominare. L'intercettazione sostituisce il comando di un altro plugin con uno script tuo, senza toccare quel plugin.

```yaml
# overrides.yml

block:
  - command: "me"
    message: "&cDisattivato su questo server."
  - command: "pl"
    aliases-too: true
    bypass-permission: "server.vedipl"

intercept:
  # /spawn resta il comando di sempre per i giocatori, ma ora fa quello che vuoi tu
  - command: "spawn"
    cancel-original: true
    run:
      - "message: &7Ti porto allo spawn..."
      - "wait: 2s"
      - "if: {var:in_combattimento|no} == si"
      - "  message: &cNon puoi scappare durante un combattimento!"
      - "  stop"
      - "end"
      - "teleport: 0 100 0 world"

alias:
  ci: "msg"
  gm0: "gamemode survival"
  gm1: "gamemode creative"
```

Funziona anche sui plugin caricati dopo di noi
Il controllo avviene sull'evento del comando, non sull'albero dei comandi del server: quindi blocca anche i comandi registrati da un plugin che si carica dopo ShadowCommand. Il permesso `shadowcommand.bypass.block` lascia passare chi deve.

4.5 Reagire all'evento di un plugin qualsiasi AVANZATO

Questo e' il trigger che non deve essere previsto nel plugin per funzionare: gli dai il nome di una classe di evento e lui la cerca fra **tutti** i plugin accesi.

```yaml
triggers:
  pass-livello:
    event: bukkit-event
    class: "net.shadowmc.battlepass.api.event.BattlePassTierUpEvent"
    run:
      - "broadcast: &e{player} &7e' salito al livello &f{new_tier} &7del pass!"
      - "if: {new_tier} == 100"
      - "  broadcast: &6&l{player} HA COMPLETATO IL BATTLE PASS!"
      - "  firework: gold star 2"
      - "end"
```

Ogni proprieta' leggibile dell'evento diventa un segnaposto, con il nome tradotto:

Metodo dell'evento	Segnaposto
`getPlayer()`	`{player}`
`getNewTier()`	`{new_tier}`
`isPremium()`	`{premium}`
`getAmount()`	`{amount}`

Ci sono sempre anche `{event}`, il nome completo della classe, e `{event_name}`. Se non sai quali proprieta' ha un evento, scrivi un trigger che stampa tutto e guarda il risultato:

```yaml
triggers:
  scopri:
    event: bukkit-event
    class: "com.qualcuno.plugin.event.QualcosaEvent"
    run:
      - "log: evento {event_name} -> giocatore={player} quantita={amount}"
```

> **Due cose da sapere**
> Se il plugin che definisce quella classe non e' installato, il trigger resta spento e il nome che hai scritto compare in `/scmd errors`: un pacchetto sbagliato altrimenti non si noterebbe. E `cancel: true` funziona solo se quell'evento e' davvero annullabile.

In `triggers/suite.yml` ne trovi sei gia' pronti, spenti, per gli altri plugin della suite: bastano da esempio per capire la forma.

5. Ricettario: le GUI

Un menu e' un file in `menus/`. Si apre con `menu: nome` da uno script, con `menu:` in un comando, o con `/scmd menu nome`.

5.1 Il menu minimo FACILE

Tre righe, tre icone. Da qui si parte sempre.

```yaml
# menus/principale.yml
config-version: 1

title: "&8Menu principale"
rows: 3
open-command: "menu"              # crea /menu da solo

items:
  regole:
    slot: 11
    material: BOOK
    name: "&6Regole"
    lore:
      - "&7Leggi le regole del server."
    click:
      - "command: regole"
      - "menu-close:"

  negozio:
    slot: 13
    material: EMERALD
    name: "&aNegozio"
    lore:
      - "&7Compra e vendi."
      - ""
      - "&8Saldo: &a{money_formatted}"
    click:
      - "menu: negozio"

  chiudi:
    slot: 15
    material: BARRIER
    name: "&cChiudi"
    click:
      - "menu-close:"
```

Gli slot si contano da **0**, da sinistra a destra e dall'alto in basso: la prima riga e' 0-8, la seconda 9-17, e cosi' via. `rows` va da 1 a 6.

5.2 Il menu completo AVANZATO

Icone che compaiono solo a chi ha il permesso, icone che cambiano faccia in base a una condizione, click diversi per tasto destro e sinistro, prezzo del click, cornice, suoni, aggiornamento dal vivo.

```yaml
# menus/negozio.yml
config-version: 1

title: "&8Negozio &7({page}/{pages})"
rows: 6
open-command: "negozio"
open-permission: "server.negozio"
open-requirements:
  - "world: !world_the_end"
open-sound: "block.chest.open"
close-sound: "block.chest.close"
update-interval: 20              # tick fra un aggiornamento e l'altro, 0 = mai
close-on-click: false
cancel-clicks: true              # false = il giocatore puo' prendere gli oggetti
open-actions:
  - "var: set ultima_gui negozio"

filler:
  material: GRAY_STAINED_GLASS_PANE
  name: " "
  slots: "border"                # border | all | 0-8,45-53

items:

  # --- una icona informativa che si aggiorna da sola ------------------
  info:
    slot: 4
    material: PLAYER_HEAD
    head: "{player}"
    name: "&eCiao {player}"
    lore:
      - "&7Saldo: &a{money_formatted}"
      - "&7Rango: &f%vault_group%"
      - "&7Online da: &f{playtime}"
    glow: true

  # --- una icona che si vede solo a chi ha il permesso ----------------
  vip:
    slot: 20
    material: DIAMOND
    name: "&bZona VIP"
    view-requirements:
      - "permission: server.vip"       # se non passa, l'icona non compare
    lore:
      - "&7Teletrasporto riservato."
      - ""
      - "&eSinistro &7per andare"
      - "&eDestro &7per le informazioni"
    left-click:
      - "teleport: 100 64 100 world"
      - "menu-close:"
    right-click:
      - "message: &7La zona VIP si sblocca con /vip."
    click-requirements:
      - "money: >= 100"
    click-deny: "&cTi servono 100 monete."
    click-sound: "ui.button.click"
```

```
    click-cooldown: "3s"

  # --- una icona che cambia aspetto: il primo stato che passa vince ---
  missione:
    slot: 22
    states:
      - view-requirements: [ "variable: quest == completata" ]
        material: LIME_DYE
        name: "&aMissione completata"
        lore: [ "&7Complimenti!" ]
      - view-requirements: [ "variable: quest == iniziata" ]
        material: ORANGE_DYE
        name: "&6Missione in corso"
        lore: [ "&7Torna quando hai finito." ]
      - material: GRAY_DYE                      # l'ultimo e' quello di riserva
        name: "&7Missione non iniziata"
        click:
          - "var: set quest iniziata"
          - "message: &6Missione accettata!"
          - "menu-refresh:"

  # --- contenuto dinamico: una icona per ogni giocatore online --------
  giocatori:
    slots: "28-34,37-43"
    source: players
    source-sort: "name"
    material: PLAYER_HEAD
    head: "{entry}"
    name: "&f{entry}"
    lore:
      - "&7Mondo: &f{entry_world}"
      - "&7Ping: &f{entry_ping}ms"
    left-click:
      - "teleport-player: {entry}"
      - "menu-close:"

navigation:
  previous: { slot: 48, material: ARROW, name: "&ePagina precedente" }
  next:     { slot: 50, material: ARROW, name: "&ePagina successiva" }
  close:    { slot: 49, material: BARRIER, name: "&cChiudi" }
```

Le sorgenti dinamiche

Con `source:` una sola icona ne genera quante servono, e le pagine si calcolano da sole sugli `slots` che hai dato.

source	Elenca	Segnaposto
`players`	i giocatori online	`{entry}`, `{entry_uuid}`, `{entry_world}`, `{entry_ping}`, `{entry_health}`
`worlds`	i mondi caricati	`{entry}`, `{entry_players}`, `{entry_time}`
`commands`	i comandi visibili a chi guarda	`{entry}`, `{entry_description}`
`list`	una variabile di tipo lista, con `source-value: {var:amici}`	`{entry}`, `{entry_index}`

`range`	numeri, con `source-value: "1-64"`	`{entry}`

I tipi di click

`click` (qualunque), `left-click`, `right-click`, `shift-left-click`, `shift-right-click`, `middle-click`, `drop-click`, `double-click`, `number-key-click`. Se c'e' `click` **e** un tipo specifico, viene eseguito solo il piu' specifico.

5.3 Un menu che accetta oggetti AVANZATO

Una cassetta delle consegne: il giocatore ci mette dentro qualcosa e alla chiusura viene pagato. Con `cancel-clicks: false` gli oggetti si possono spostare davvero.

```yaml
# menus/consegna.yml
config-version: 1

title: "&8Consegna il grano"
rows: 3
cancel-clicks: false
input-slots: "10-16"

filler:
  material: BLACK_STAINED_GLASS_PANE
  name: " "
  slots: "0-9,17-26"

close-actions:
  - "message: &7Hai consegnato &e{menu_items} &7oggetti."
  - "money: give {math:{menu_items}*5}"
  - "menu-return-items:"          # rende quello che non e' stato accettato
```

> **Gli oggetti non si perdono mai**
> Quello che resta negli `input-slots` alla chiusura viene sempre restituito, anche se il server si spegne in quel momento: viene salvato e riconsegnato al login successivo.

5.4 Animazioni e menu annidati AVANZATO

```yaml
items:
  caricamento:
    slot: 13
    frames:
      - { material: WHITE_STAINED_GLASS_PANE, name: "&f|" }
      - { material: LIGHT_GRAY_STAINED_GLASS_PANE, name: "&7/" }
      - { material: GRAY_STAINED_GLASS_PANE, name: "&8-" }
    frame-interval: 5             # tick fra un fotogramma e l'altro
```

Le animazioni richiedono `update-interval` maggiore di zero. L'aggiornamento riscrive **solo le icone cambiate**: il menu non si chiude e non sfarfalla, quindi va bene anche per contatori e saldi che cambiano di continuo.

Aprire un menu da dentro un altro menu ricorda quello precedente: l'icona `back` della `navigation` torna indietro da sola. La profondita' massima e' 8, cosi' una catena sbagliata non diventa infinita.

Costruire le GUI senza scrivere YAML

`/scmd menus` elenca i menu, li apre in anteprima, ne crea di nuovi da un modello e permette di spostare un'icona da uno slot all'altro trascinandola. Le modifiche vengono scritte nel file, e la versione precedente resta conservata per poter tornare indietro.

6. Il linguaggio

Uno script e' una lista di righe. Ogni riga e' `azione: argomenti`. Non serve sapere altro per iniziare, e questo capitolo e' quello che serve per finire.

```yaml
run:
  - "message: &aCiao {player}!"
  - "sound: entity.experience_orb.pickup"
```

L'indentazione dentro le stringhe e' solo estetica: serve a te per leggere i blocchi, viene tolta prima di compilare e non ha nessun significato.

Le tre sintassi

Si scrive	Chi lo risolve	Esempio
`{...}`	ShadowCommand	`{player}`, `{arg1}`, `{var:punti}`
`%...%`	PlaceholderAPI, se installato	`%vault_eco_balance%`
`&x`	Colori e stili	`&a`, `&l`, `&#FF8800`

Per scrivere una graffa vera: `\{`. Per una percentuale vera: `\%`. Per una e commerciale vera: `&&`.

Il controllo di flusso

if / elseif / else / end

```yaml
- "if: {level} >= 30"
- "  message: &6Sei un veterano."
- "elseif: {level} >= 10"
- "  message: &eTe la cavi."
- "else:"
- "  message: &7Sei all'inizio."
- "end"
```

while

```yaml
- "var: set i 0"
- "while: {var:i} < 5"
- "  message: &7Giro {var:i}"
- "  var: add i 1"
- "end"
```

repeat

```yaml
- "repeat: 3"
- "  particle-at-player: FLAME 20"
- "  wait: 10t"
- "end"
```

for

```yaml
- "for: colore in rosso,verde,blu"
- "  message: &7Colore: {colore}"
- "end"
```

Dentro un ciclo, `{loop}` vale 1, 2, 3... e `{loop_index}` vale 0, 1, 2... `break` esce dal ciclo piu' interno, `continue` salta al giro successivo. La lista di `for` puo' essere scritta a mano, una variabile di tipo lista o un segnaposto che restituisce valori separati da virgola.

Parola	Cosa fa
`chance: 10`	Esegue il blocco il 10% delle volte

`stop`	Termina lo script qui
`return: valore`	Esce dalla funzione restituendo un valore
`goto:` / `label:`	Salti, per chi arriva da MyCommand. Funzionano, ma `if` e `while` sono piu' chiari
`call: nome args`	Chiama una funzione
`wait: 3s`	Sospende e riprende dopo
`async:` ... `end`	Blocco fuori dal thread di gioco

Le funzioni

Stanno in `functions/`, oppure in qualunque file sotto la chiave `functions:`. Servono a non riscrivere dieci volte le stesse righe.

```yaml
functions:
  premia:
    args: [ "quanti", "motivo" ]
    run:
      - "give: DIAMOND {quanti}"
      - "message: &a+{quanti} diamanti &7({motivo})"
      - "var: add premi 1"
      - "return: {quanti}"
```

```yaml
run:
  - "call: premia 3 'primo accesso'"
  - "message: &7La funzione ha restituito {result}."
```

Gli argomenti si citano per nome e sono **locali**: non toccano le variabili del giocatore. La ricorsione e' permessa fino al limite di profondita' impostato in `config.yml`.

Le attese

```yaml
- "wait: 20t"        # tick
- "wait: 3s"         # secondi
- "wait: 1m30s"      # minuti e secondi
```

> **`wait` non blocca niente**
> Non e' una pausa del server: la sessione dello script viene sospesa e ripresa piu' tardi dallo scheduler. Mille giocatori con uno script in attesa non occupano mille thread, ne occupano zero. Se il giocatore esce, la sessione si chiude da sola.

Le espressioni

Ovunque ci sia una condizione - `if`, `while`, `chance`, i requisiti, i `view-requirements` delle GUI - vale la stessa grammatica.

Gruppo	Operatori
Logici	`&&` `||` `!`
Confronto	`==` `!=` `>` `>=` `<` `<=`

Testo	`contains` `startsWith` `endsWith` `matches` (regex) `in`
Matematica	`+` `-` `*` `/` `%` `^`

Fra due numeri il confronto e' numerico, fra un numero e un testo e' testuale, e di fabbrica non distingue maiuscole e minuscole.

Le funzioni delle espressioni

```
abs(x)   min(a,b)   max(a,b)   round(x[,cifre])   floor(x)   ceil(x)   sqrt(x)   pow(a,b)
rand(a,b)   randf(a,b)   sign(x)   clamp(x,a,b)   avg(...)   sum(...)
len(s)   upper(s)   lower(s)   cap(s)   trim(s)   reverse(s)
sub(s,da,a)   replace(s,cerca,sostituisci)   index(s,cerca)   split(s,sep,n)   count(s,cerca)
number(s)   text(x)   bool(x)   ifelse(cond,a,b)   default(x,y)
now()   time(formato)   date(formato)   since(millis)
```

```
- "if: max({var:punti}, 0) >= 100 && upper({world}) contains NETHER"
- "  message: &6Condizione complicata soddisfatta."
- "end"
```

Quando sbagli a scrivere

Ogni file viene compilato all'avvio. Un problema produce una riga come questa, con file, riga e suggerimento:

```
[ShadowCommand] commands/kit.yml -> kit.run:7   azione sconosciuta 'mesage'
                                                forse intendevi: message
[ShadowCommand] commands/kit.yml -> kit.run:12  'if' aperto alla riga 4 non chiuso da 'end'
```

Il comando che contiene l'errore **non viene registrato**: meglio un comando che manca di uno che si interrompe a meta' davanti ai giocatori. Gli errori restano consultabili con `/scmd errors` e nel pannello.

> **Tre strumenti che fanno risparmiare un'ora**
> `/scmd test <riga>` esegue una riga sola, subito. `/scmd parse <testo>` mostra come vengono risolti i segnaposto, con i valori veri. `/scmd debug <comando>` esegue passo passo mostrando cosa fa ogni riga.

Quando un'azione sbaglia in gioco

Un'azione che riceve argomenti impossibili - un materiale che non esiste, un giocatore offline, un numero che non e' un numero - **non ferma lo script**: registra l'errore, lo scrive in console una volta al minuto per non intasare, e prosegue. Se preferisci il contrario, nel comando si scrive:

```
strict: true      # il primo errore ferma lo script ed esegue on-error
```

7. Tutte le azioni

Ogni riga di uno script e' un'azione. Qui ci sono tutte, con la forma esatta. In gioco lo stesso elenco e' in `/scmd actions`, con la ricerca.

Legenda: `<obbligatorio>`, `[opzionale]`. Fra parentesi gli alias.

Messaggi

Azione	Forma	Cosa fa
`message` (msg, m)	`message: <testo>`	Manda un messaggio a chi ha avviato lo script
`tell`	`tell: <giocatore> <testo>`	Manda un messaggio a un altro
`broadcast` (bc)	`broadcast: <testo>`	A tutto il server
`broadcast-world`	`broadcast-world: <mondo> <testo>`	A un mondo solo
`broadcast-perm`	`broadcast-perm: <permesso> <testo>`	A chi ha quel permesso
`broadcast-radius`	`broadcast-radius: <raggio> <testo>`	A chi e' vicino
`actionbar`	`actionbar: <testo>`	Sopra la barra degli oggetti
`title`	`title: <titolo>`	Titolo grande a schermo
`subtitle`	`subtitle: <testo>`	Sottotitolo
`title-full`	`title-full: <entra> <resta> <esce> | <titolo> | <sottotitolo>`	Titolo con i tempi
`bossbar`	`bossbar: <id> <durata> <colore> <stile> <testo>`	Barra in alto
`bossbar-remove`	`bossbar-remove: <id>`	Toglie la barra
`book`	`book: <titolo> | <pagina1> | <pagina2>`	Apre un libro
`chat`	`chat: <testo>`	Fa parlare il giocatore in chat
`log`	`log: <testo>`	Scrive in console
`clear-chat`	`clear-chat: [righe]`	Pulisce la chat

Comandi

Azione	Forma	Cosa fa
`command` (player)	`command: <comando>`	Eseguito dal giocatore
`console`	`console: <comando>`	Eseguito dalla console
`op`	`op: <comando>`	Come giocatore, OP per un istante
`bypass`	`bypass: <permesso> | <comando>`	Con quel permesso concesso per un istante
`runas`	`runas: <giocatore> <comando>`	Come un altro giocatore
`cancel`	`cancel`	Annulla l'evento che ha avviato lo script

`script`	`script: <comando>`	Esegue lo script di un altro comando tuo
`trigger`	`trigger: <nome>`	Esegue un trigger a mano

Suoni, particelle, effetti

Azione	Forma
`sound`	`sound: <suono> [volume] [tono] [giocatore]`
`sound-all`	`sound-all: <suono> [volume] [tono]`
`sound-at`	`sound-at: <suono> <x> <y> <z> [mondo] [volume] [tono]`
`stop-sound`	`stop-sound: [suono]`
`particle`	`particle: <tipo> <x> <y> <z> [quantita] [dx] [dy] [dz] [velocita]`
`particle-at-player`	`particle-at-player: <tipo> [quantita]`
`effect`	`effect: <effetto> <durata> <livello> [nascondi]`
`effect-clear`	`effect-clear: [effetto]`
`firework`	`firework: <colore> [tipo] [potenza]`
`lightning`	`lightning: [x y z] [danno]`

Il giocatore

Azione	Forma
`teleport` (tp)	`teleport: <x> <y> <z> [mondo] [yaw] [pitch]` oppure `teleport: <nome-warp>`
`teleport-player`	`teleport-player: <giocatore>`
`heal`	`heal: [quantita]`
`feed`	`feed: [quantita]`
`damage`	`damage: <quantita>`
`kill`	`kill: [giocatore]`
`gamemode`	`gamemode: <survival|creative|adventure|spectator>`
`fly`	`fly: <on|off|toggle>`
`speed`	`speed: <walk|fly> <0..10>`
`exp`	`exp: <give|take|set> <quantita>`
`level`	`level: <give|take|set> <quantita>`
`health`	`health: <set|add|take> <quantita>`
`food`	`food: <set|add|take> <quantita>`
`fire`	`fire: <secondi>`
`velocity` (push)	`velocity: <x> <y> <z>`
`invulnerable`	`invulnerable: <on|off> [durata]`
`glow`	`glow: <on|off> [durata]`

`kick`	`kick: [giocatore] | <motivo>`

Inventario

Azione	Forma
`give`	`give: <materiale> [quantita] [opzioni]`
`take` (remove)	`take: <materiale> [quantita]`
`clear`	`clear: [materiale]`
`drop`	`drop: <materiale> [quantita]`
`equip`	`equip: <helmet|chestplate|leggings|boots|hand|offhand> <materiale> [opzioni]`
`open-inventory`	`open-inventory: <enderchest|crafting|anvil|workbench>`

Le opzioni di `give` ed `equip`, in qualunque ordine:

```
name:'&bSpada'        lore:'riga1|riga2'      amount:3          model:1001
enchant:SHARPNESS:5   enchant:UNBREAKING:3    unbreakable       glow
flags:HIDE_ATTRIBUTES head:{player}           color:#FF0000     nbt-safe
```

```
- "give: DIAMOND_SWORD 1 name:'&bLama d ombra' lore:'&7Forgiata nel buio|&8Unica'
enchant:SHARPNESS:5 unbreakable glow"
```

Denaro, permessi, gruppi

Azione	Forma	Serve
`money`	`money: <give|take|set> <quantita> [valuta]`	ShadowEconomy o Vault
`pay`	`pay: <giocatore> <quantita> [valuta]`	ShadowEconomy o Vault
`permission`	`permission: <add|remove> <nodo> [durata]`	ShadowPermissions, LuckPerms o Vault
`group`	`group: <add|remove|set> <gruppo> [durata]`	ShadowPermissions, LuckPerms o Vault

Variabili

Azione	Forma
`var`	`var: <set|add|sub|mul|div|append|remove|delete|expire> <nome> [valore] [durata]`
`gvar`	come `var`, ma globale (stessa variabile per tutti i giocatori)
`svar`	come `var`, ma di server (una sola copia, non legata a nessuno)
`var-of`	`var-of: <giocatore> <set|add|...> <nome> [valore]` - anche su chi e' offline
`local`	`local: <nome> <valore>` - vive solo dentro questa esecuzione

GUI

Azione	Forma

`menu`	`menu: <nome> [giocatore]`
`menu-close`	`menu-close:`
`menu-refresh`	`menu-refresh:`
`input`	`input: <domanda> | <variabile>` - chiede un testo sull'incudine
`confirm`	`confirm: <domanda>` - due pulsanti in chat, prosegue solo con si'

`input:` e `confirm:` sospendono lo script e lo riprendono con la risposta. Un `confirm:` con risposta negativa esegue il blocco `else:` se c'e' subito dopo.

Mondo

Azione	Forma
`time`	`time: <set|add> <valore> [mondo]`
`weather`	`weather: <clear|rain|thunder> [durata] [mondo]`
`setblock`	`setblock: <x> <y> <z> <materiale> [mondo]`
`breakblock`	`breakblock: <x> <y> <z> [mondo] [drop]`
`spawn-mob`	`spawn-mob: <tipo> [quantita] [x y z] [nome]`
`cooldown`	`cooldown: <set|clear> <chiave> [durata] [giocatore]`

Battle pass (richiede ShadowBattlePass)

Forma	Cosa fa
`battlepass: xp <quantita> [sorgente]`	Esperienza, con i moltiplicatori attivi
`battlepass: xp-raw <quantita>`	Esperienza esatta, senza moltiplicatori
`battlepass: tier <quanti>`	Fa salire di livello
`battlepass: coins <give|take> <quantita>`	Monete del pass
`battlepass: booster <molt> <durata>`	Moltiplicatore per un giocatore
`battlepass: global-booster <molt> <durata>`	Moltiplicatore per tutti
`battlepass: event <nome> [quantita]`	Fa avanzare le missioni che guardano quel nome
`battlepass: unlock <track> [retroattivo]`	Sblocca un percorso

> **`battlepass: event` e' quella che cambia le cose**
> Una missione del pass puo' essere "fai quello che fa il mio comando", e il comando lo dice al pass con una riga sola. Non serve che il pass sappia niente del tuo comando.

Minigiochi (richiede ShadowGames)

Forma	Cosa fa
`game: join <gioco> [arena]`	Mette il giocatore in coda
`game: leave`	Lo tira fuori

`game: spectate <idPartita>`	Lo fa entrare da spettatore
`game: start [idPartita]`	Forza l'avvio
`game: stop [idPartita] [motivo]`	Forza la fine
`game: xp <quantita>`	Esperienza dei minigiochi
`game: coins <give|take> <quantita>`	Monete dei minigiochi
`game: stat <gioco> <chiave> <quantita>`	Somma a una statistica

Quando il motore rifiuta, il motivo finisce in `{game_error}` e lo script prosegue: cosi' puoi mostrare "arena piena" invece di un silenzio.

> **Senza il plugin corrispondente**
> Tutte le azioni `battlepass:` e `game:` non fanno niente e lo scrivono una volta sola in console. Lo script non si rompe.

8. Requisiti, costi, cooldown

Tutto quello che decide **se** un comando parte, **quanto** costa e **quando** si potra' riusare.

I requisiti

Un requisito e' una condizione che deve essere vera **prima** che lo script parta. Se una non lo e', lo script non parte affatto, il giocatore riceve il messaggio del requisito e viene eseguito l'eventuale blocco `on-deny`.

Due forme: la corta e la completa.

```yaml
requirements:
  - "money: >= 500"                    # corta
  - type: level                        # completa
    value: ">= 10"
    deny: "&cTi serve il livello 10."
    sound: "block.note_block.bass"
```

Elenco completo

Requisito	Forma corta	Note
`permission`	`permission: nodo`	`!nodo` per "NON deve averlo"
`money`	`money: >= 500`	ShadowEconomy o Vault
`level`	`level: >= 10`	livelli di esperienza
`exp`	`exp: >= 1000`	punti esperienza totali
`health`	`health: > 10`	
`food`	`food: >= 6`	
`world`	`world: world,world_nether`	`!world_the_end` per escludere
`gamemode`	`gamemode: SURVIVAL,ADVENTURE`	
`item`	`item: DIAMOND 5`	deve averli nell'inventario
`item-in-hand`	`item-in-hand: DIAMOND_PICKAXE`	
`empty-slots`	`empty-slots: >= 3`	spazio libero
`y`	`y: > 60`	altezza
`biome`	`biome: PLAINS,FOREST`	
`region`	`region: spawn`	ShadowRegions o WorldGuard
`distance`	`distance: <= 100 0 64 0 world`	dal punto indicato
`time`	`time: 0-12000`	tick del mondo
`real-time`	`real-time: 20:00-23:59`	ora reale del server
`weather`	`weather: clear`	
`sneaking`	`sneaking: true`	
`flying`	`flying: false`	

`riding`	`riding: false`	
`online`	`online: >= 5`	giocatori collegati
`player-online`	`player-online: {arg1}`	il bersaglio deve essere collegato
`sender`	`sender: player`	`player`, `console`, `any`
`cooldown`	`cooldown: kit-vip`	vero solo se quel cooldown e' scaduto
`variable`	`variable: quest == completata`	una variabile del giocatore
`condition`	`condition: {level} * 2 > {var:soglia}`	espressione libera
`placeholder`	`placeholder: %vault_group% == vip`	confronto su PlaceholderAPI
`chance`	`chance: 50`	passa il 50% delle volte
`battlepass-tier`	`battlepass-tier: >= 10`	livello nel pass
`battlepass-premium`	`battlepass-premium: true`	deve avere il premium
`battlepass-track`	`battlepass-track: vip`	percorso sbloccato
`battlepass-quest`	`battlepass-quest: <id>`	missione completata
`battlepass-coins`	`battlepass-coins: >= 100`	monete del pass
`battlepass-streak`	`battlepass-streak: >= 7`	giorni consecutivi
`battlepass-season`	`battlepass-season: estate2026`	stagione in corso
`in-game`	`in-game: true`	dev'essere in partita
`game`	`game: bedwars,skywars`	quale minigioco
`game-state`	`game-state: RUNNING`	stato della partita
`game-arena`	`game-arena: arena1`	quale arena
`spectating`	`spectating: false`	sta guardando
`game-level`	`game-level: >= 5`	livello nei minigiochi
`game-coins`	`game-coins: >= 500`	monete dei minigiochi

Come si combinano

Di fabbrica devono essere veri **tutti**. Per cambiare:

```
requirements-mode: any        # all (di fabbrica) | any | none
```

Oppure si scrive tutto in una condizione sola:

```
requirements:
  - "condition: {level} >= 10 || {var:vip} == si"
```

> **I requisiti della suite rispondono falso quando il plugin manca**
>
> E' voluto: un requisito esiste per fermare qualcosa, quindi quando non sa decidere ferma. Un server senza battle pass non regala il kit che avevi messo dietro al livello 20.

I costi

Si pagano **dopo** che i requisiti sono passati e **prima** che lo script parta. Se qualcosa non basta non viene prelevato niente: e' tutto o niente.

```yaml
cost:
  money: 250
  exp: 100
  levels: 2
  hunger: 4
  health: 2
  items:
    - "DIAMOND 3"
    - "IRON_INGOT 16"
  message: "&7Hai pagato &e{cost_money} monete."
```

Il permesso `shadowcommand.bypass.cost`, oppure `cost-bypass:` scritto nel comando, salta il pagamento. Se lo script si ferma per un errore il costo **non** viene restituito: se lo vuoi, rimborsalo tu in `on-error`.

I cooldown

```yaml
cooldown: "30s"                 # per giocatore
global-cooldown: "5m"           # per tutto il server
cooldown-key: "kit"             # condiviso fra piu' comandi con la stessa chiave
cooldown-persistent: true       # sopravvive al riavvio (di fabbrica: true)
cooldown-bypass: "server.nocd"
cooldown-message: "&cAspetta ancora &e{cooldown_left}&c."
cooldown-on-deny: false         # true = paga l'attesa anche quando un requisito nega
```

`{cooldown_left}` esce come `2m 14s` nella lingua del giocatore; `{cooldown_left_seconds}` e' il numero secco. Da script:

```yaml
- "cooldown: set kit-vip 1h"
- "cooldown: clear kit-vip"
```

Non si perdono al riavvio

Di fabbrica un cooldown finisce nel database - quello locale, o il tuo MySQL - e viene riletto all'avvio. La scrittura avviene **subito**, non al salvataggio periodico: se aspettasse i trenta secondi del salvataggio in blocco, un server ucciso male subito dopo tornerebbe su senza quel cooldown, e prendere due volte lo stesso kit sarebbe questione di aspettare un crash.

Con `cooldown-persistent: false` resta solo in memoria: va bene per le attese di pochi secondi, dove scrivere su disco a ogni uso sarebbe sprecato. Le righe scadute vengono cancellate da sole all'avvio e ogni ora, quindi la tabella non cresce.

Il warmup

Un'attesa **prima** dell'esecuzione, con barra e possibilita' di annullamento.

```
warmup: "5s"
warmup-bossbar: true
warmup-cancel-on-move: true
warmup-cancel-on-damage: true
warmup-message: "&7Non muoverti per &e{warmup}&7..."
warmup-cancelled: "&cAnnullato: ti sei mosso."
```

Il permesso `shadowcommand.bypass.warmup` lo salta.

La conferma

```
confirm: true
confirm-text: "&7Stai per spendere &e500 monete&7."
```

La domanda compare in chat con due pulsanti cliccabili, uno verde e uno rosso: non c'e' niente da riscrivere. E' pensata per i comandi che costano o che sono irreversibili. Da script si fa lo stesso con `confirm: <domanda>`.

> **Il codice dietro ai pulsanti e' monouso**
> Ogni domanda genera un codice casuale legato a chi l'ha ricevuta, valido 60 secondi e per una volta sola. Nessun altro puo' rispondere al posto tuo, nemmeno vedendo il comando in uno screenshot.

9. Segnaposto e variabili

I segnaposto si scrivono fra graffe e funzionano **ovunque**: messaggi, condizioni, nomi delle icone, titoli delle GUI, argomenti delle azioni.

Le quattro portate delle variabili

Portata	Chi la vede	Dove vive	Come si scrive
locale	solo questa esecuzione	memoria	`local: x 5` → `{x}`
giocatore	un giocatore, ovunque	database	`var: set punti 10` → `{var:punti}`
globale	tutti, una copia per giocatore	database	`gvar: ...` → `{gvar:nome}`
server	una sola copia in tutto	database	`svar: set jackpot 500` → `{svar:jackpot}`

> `gvar` e `svar` **si confondono facilmente**
>
> `gvar` e' una variabile che **esiste per ogni giocatore** con lo stesso nome e lo stesso valore di partenza (comoda per "tutti partono da 3 vite"). `svar` e' una variabile **del server**, unica: il montepremi, il numero di eventi fatti.

Le operazioni

```
- "var: set nome valore"          # imposta
- "var: add punti 10"             # somma
- "var: sub punti 3"              # sottrae
- "var: mul punti 2"              # moltiplica
- "var: div punti 4"              # divide
- "var: append lista mela"        # aggiunge a una lista
- "var: remove lista mela"        # toglie da una lista
- "var: delete punti"             # cancella
- "var: expire punti 7d"          # la fa scadere fra 7 giorni
- "var-of: {arg1} add punti 5"    # su un altro giocatore, anche offline
```

Una variabile che non esiste vale stringa vuota nei testi e `0` nei calcoli: non serve dichiararla. Il tipo si deduce dal valore. Per forzarlo: `{var:punti|number}`, `{var:nome|text}`.

Valori di riserva. `{var:punti|0}` vale `0` se la variabile non esiste. Vale per qualunque segnaposto: `{arg2|nessuno}`, `{world|world}`.

Segnaposto del giocatore

{player}	nome	{uuid}	UUID
{displayname}	nome visualizzato	{sender}	nome, o CONSOLE
{world}	mondo	{gamemode}	modalita'
{x} {y} {z}	posizione intera	{x_exact} ...	con i decimali
{yaw} {pitch}	rotazione	{block_x} ...	blocco sotto i piedi
{health}	vita	{maxhealth}	vita massima
{food}	fame	{saturation}	saturazione
{level}	livello	{exp}	esperienza totale
{ip}	indirizzo	{ping}	ritardo in ms
{money}	saldo	{money_formatted}	saldo formattato
{group}	gruppo principale	{prefix} {suffix}	da LuckPerms / ShadowPermissions
{item}	oggetto in mano	{item_name}	il suo nome
{item_amount}	quanti	{item_lore}	la sua descrizione
{target}	giocatore inquadrato	{target_block}	blocco inquadrato
{biome}	bioma	{light}	luce
{first_join}	true/false	{playtime}	tempo di gioco
{flying}	true/false	{sneaking}	true/false

Segnaposto del server

{server_online}	giocatori collegati	{server_max}	massimo
{server_tps}	TPS	{server_tps_1m}	media a un minuto
{server_ram_used}	MB usati	{server_ram_max}	MB totali
{server_uptime}	da quanto e' acceso	{server_version}	versione
{server_online_names}	elenco separato da virgole		
{server_world_time}	ora del mondo	{server_weather}	meteo

Argomenti, calcolo, testo, tempo

Argomenti

{arg1} {arg2} ...	i singoli argomenti
{arg:nome}	per nome
{args}	tutti
{args:2-}	dal secondo in poi
{args:2-4}	dal secondo al quarto
{argslength}	quanti sono
{label}	con quale alias

Calcolo e caso

{math:1+2*3}	7
{math:{var:punti}/2|0}	con riserva
{rand:1-100}	numero a caso
{rand:mela,pera,uva}	elemento a caso
{randplayer}	un giocatore a caso
{uuidgen}	un UUID nuovo

Testo

```
{upper:{player}}    MAIUSCOLO
{lower:...}         minuscolo
{cap:...}           Prima Maiuscola
{trim:...}          senza spazi ai lati
{len:...}           lunghezza
{rev:...}           al contrario
{sub:testo,0,3}     sottostringa
{rep:testo,a,b}     sostituisce
{strip:...}         toglie i colori
{esc:...}           protegge i simboli
```

Tempo e stato

```
{time}              ora del server
{date}              data
{time:HH:mm:ss}     formato libero
{date:EEEE d MMMM}  nella lingua giusta
{timestamp}         millisecondi
{cooldown_left}     quanto manca
{cooldown:kit}      di un cooldown a
scelta
{uses:kit}          quante volte usato
{uses_player:kit}   da questo giocatore
{last_used:kit}     quando
```

Dentro cicli, funzioni e GUI

```
{loop} {loop_index}      dentro un ciclo
{result}                 valore restituito dall'ultima funzione
{menu_page} {menu_pages} dentro una GUI
{entry} {entry_*}        dentro una sorgente dinamica di una GUI
```

Battle pass e minigiochi

```
{battlepass_tier}          livello nel pass       {battlepass_maxtier}   livello massimo
{battlepass_xp}            esperienza             {battlepass_premium}   true/false
{battlepass_progress}      avanzamento in %       {battlepass_coins}     monete del pass
{battlepass_pending}       premi da ritirare      {battlepass_streak}    giorni consecutivi
{battlepass_rank}          posizione              {battlepass_season}    id stagione
{battlepass_season_name}   nome stagione          {battlepass_time_left} quanto manca
{battlepass_multiplier}    moltiplicatore attivo  {battlepass_available} true se installato
{battlepass_stat:kills}    una statistica         {battlepass_quest:id}  avanzamento missione

{game}                     minigioco in corso     {in_game}              true/false
{game_arena}               arena                  {game_state}           stato partita
{game_players}             giocatori              {game_max}             posti
{game_alive}               ancora vivi            {game_id}              numero partita
{game_level}               livello                {game_xp}              esperienza
{game_coins}               monete                 {games_running}        partite in corso
{games_list}               elenco minigiochi      {games_available}      true se installato
{spectating}               sta guardando
```

Senza il plugin corrispondente rispondono `0`, `false` o vuoto: una scoreboard scritta per la suite completa non si riempie di errori su un server che ne ha installata solo meta'.

PlaceholderAPI

Li leggiamo. Ogni `%...%` nei tuoi testi passa da PlaceholderAPI. Se non e' installato, il testo resta com'e' e la cosa viene segnalata una volta sola in console.

Li forniamo. Con PlaceholderAPI presente, gli altri plugin - scoreboard, tab, chat - possono leggere i tuoi dati:

```
%shadowcommand_var_&lt;nome&gt;%             variabile del giocatore
%shadowcommand_gvar_&lt;nome&gt;%            variabile globale
%shadowcommand_svar_&lt;nome&gt;%            variabile di server
%shadowcommand_cooldown_&lt;chiave&gt;%      quanto manca, formattato
%shadowcommand_cooldown_raw_&lt;chiave&gt;%  quanto manca, in secondi
%shadowcommand_uses_&lt;comando&gt;%         quante volte e' stato usato
%shadowcommand_commands%               quanti comandi personalizzati esistono
%shadowcommand_can_&lt;comando&gt;%          true/false: puo' eseguirlo adesso
```

Compatibilita' MyCommand

Con `compatibility.dollar-placeholders: true` in `config.yml` funzionano anche `$player`, `$world`, `$arg1`, `$args`, `$x $y $z`, `$money`, `$online`, `$displayname`, `$uuid`. Vengono tradotti prima della compilazione, quindi non costano niente mentre il server gira.

Pulizia e scadenze

Le variabili con una scadenza vengono cancellate da un lavoro periodico ogni ora, e comunque ignorate se lette dopo la scadenza. Le variabili di un giocatore che non entra da piu' di `variables.purge-after-days` giorni (0 = mai) vengono rimosse all'avvio.

`/scmd vars` sfoglia tutto, filtra per giocatore o per nome, mostra il valore e permette di cambiarlo o cancellarlo.

10. I comandi del plugin

Tutta l'amministrazione vive sotto `/scmd`. Gli alias sono `/shadowcommand`, `/sc` e `/scommand`.

Sono i comandi **di ShadowCommand**: quelli che crei tu hanno il nome che gli dai e non c'entrano con questo elenco.

Elenco completo

Comando	Cosa fa	Permesso
`/scmd`	Apre il pannello	`shadowcommand.admin`
`/scmd help [pagina]`	Guida in gioco	`shadowcommand.admin`
`/scmd reload [file]`	Ricarica tutto, o un file solo	`shadowcommand.reload`
`/scmd list [filtro]`	Elenca i comandi personalizzati	`shadowcommand.admin`
`/scmd info <comando>`	Dettaglio di un comando	`shadowcommand.admin`
`/scmd edit <comando>`	Apre l'editor del comando	`shadowcommand.edit`
`/scmd new <nome>`	Crea un comando da un modello	`shadowcommand.edit`
`/scmd delete <comando>`	Cancella un comando (con conferma)	`shadowcommand.edit`
`/scmd toggle <comando>`	Accende o spegne un comando	`shadowcommand.edit`
`/scmd run <comando> [args]`	Esegue lo script su te stesso	`shadowcommand.run`
`/scmd runas <giocatore> <comando>`	Lo esegue su un altro	`shadowcommand.runas`
`/scmd test <riga>`	Esegue una riga sola, subito	`shadowcommand.run`
`/scmd debug <comando>`	Esecuzione passo passo, con i valori	`shadowcommand.debug`
`/scmd parse <testo>`	Mostra come vengono risolti i segnaposto	`shadowcommand.debug`
`/scmd errors`	Errori di compilazione e di esecuzione	`shadowcommand.admin`
`/scmd stats [uses|time]`	Classifica di usi e tempi	`shadowcommand.admin`
`/scmd menus`	Elenca e apre i menu	`shadowcommand.admin`
`/scmd menu <nome> [giocatore]`	Apre un menu	`shadowcommand.menu`
`/scmd triggers`	Elenca i trigger e i listener attivi	`shadowcommand.admin`
`/scmd schedules`	Elenca le schedulazioni	`shadowcommand.admin`
`/scmd schedule run <nome>`	Esegue subito una schedulazione	`shadowcommand.admin`
`/scmd vars [giocatore]`	Sfoglia le variabili	`shadowcommand.vars`
`/scmd vars set <giocatore> <nome> <valore>`	Imposta una variabile	`shadowcommand.vars.edit`
`/scmd vars del <giocatore> <nome>`	Cancella una variabile	`shadowcommand.vars.edit`
`/scmd cooldown clear <giocatore> [chiave]`	Azzera i cooldown	`shadowcommand.admin`

`/scmd actions [cerca]`	Elenco delle azioni con la sintassi	`shadowcommand.admin`
`/scmd requirements [cerca]`	Elenco dei requisiti	`shadowcommand.admin`
`/scmd placeholders [cerca]`	Elenco dei segnaposto	`shadowcommand.admin`
`/scmd import mycommand [cartella]`	Importa da MyCommand	`shadowcommand.import`
`/scmd export <comando>`	Stampa il comando in YAML	`shadowcommand.admin`
`/scmd suite`	Stato delle integrazioni	`shadowcommand.admin`
`/scmd lang [lingua]`	Cambia la tua lingua	`shadowcommand.admin`
`/scmd licenza`	La licenza installata e l'id di questo server	`shadowcommand.admin`
`/scmd version`	Versione e ambiente	`shadowcommand.admin`

Il pannello

`/scmd` senza argomenti apre l'interfaccia. Da ogni voce si scende nel dettaglio: sfogliare, cercare, accendere e spegnere, modificare le righe dello script una per una, provare il comando su te stesso, tornare a una versione precedente.

```
+-----------------------------------------------------+
|  ShadowCommand                                      |
|                                                     |
|   [Comandi]     24 definiti, 23 attivi              |
|   [Menu]         5 definiti                         |
|   [Trigger]     11 attivi, 4 listener registrati    |
|   [Schedulaz.]   4, prossima fra 3m 12s             |
|   [Variabili]   1.204 righe                         |
|   [Statistiche] piu' usato: /kit (3.201)            |
|   [Errori]      nessuno                             |
|   [Azioni]      84 disponibili                      |
|                                                     |
|   [Ricarica]    [Importa]     [Lingua]     [Chiudi] |
+-----------------------------------------------------+
```

I quattro comandi che userai di piu'

Comando	Quando
`/scmd reload`	Dopo ogni modifica ai file. Rilegge tutto: comandi, lingue, menu, trigger. Gli script gia' in esecuzione finiscono con la versione vecchia, senza rompersi
`/scmd errors`	Quando un comando non compare: quasi sempre la risposta e' qui
`/scmd test <riga>`	Per provare una riga senza creare un comando. `/scmd test message: &aciao {player}`
`/scmd parse <testo>`	Per capire perche' un segnaposto non esce come credevi: mostra il valore vero

> **Ricaricare un file solo**
> Mentre stai scrivendo, `/scmd reload commands/kit.yml` ricarica soltanto quello: piu' veloce, e non tocca il resto.

11. I permessi

Tre famiglie: quelli dell'amministrazione, quelli che scavalcano i controlli, e quelli che nascono da soli per ogni comando che scrivi.

Amministrazione

Nodo	Di fabbrica	Cosa da'
`shadowcommand.admin`	op	Pannello, elenchi, informazioni
`shadowcommand.reload`	op	Ricaricare
`shadowcommand.edit`	op	Creare, modificare, cancellare comandi
`shadowcommand.run`	op	Eseguire script a mano
`shadowcommand.runas`	op	Eseguirli su altri giocatori
`shadowcommand.debug`	op	Debug passo passo
`shadowcommand.vars`	op	Vedere le variabili
`shadowcommand.vars.edit`	op	Cambiarle
`shadowcommand.import`	op	Importare da MyCommand
`shadowcommand.menu`	op	Aprire un menu a mano
`shadowcommand.*`	op	Tutto quanto sopra

Scavalcamenti

Nodo	Di fabbrica	Cosa da'
`shadowcommand.bypass.cooldown`	false	Nessuna attesa fra un uso e l'altro
`shadowcommand.bypass.cost`	false	Non paga i costi
`shadowcommand.bypass.warmup`	false	Nessun tempo di caricamento
`shadowcommand.bypass.requirements`	false	Salta i requisiti
`shadowcommand.bypass.block`	false	Usa i comandi bloccati da `overrides.yml`

Nessuno di questi e' concesso agli operatori

E' voluto: uno staff che prova un comando deve vederlo **come lo vedono i giocatori**, se non chiede il contrario. Un cooldown che non scatta perche' chi provava era OP e' un bug scoperto dai giocatori.

I permessi dei tuoi comandi

Ogni comando che **non** dichiara `permission:` riceve automaticamente `shadowcommand.command.<nome>`, con `permission-default` a `true`: cioe' tutti possono usarlo, se non dici il contrario.

```yaml
kit:
  permission: "server.kit"        # se lo ometti: shadowcommand.command.kit
  permission-default: op          # true | false | op | not-op
  run:
    - "give: BREAD 16"
```

permission-default	Chi puo' usarlo senza che tu dia niente
`true`	tutti (valore di fabbrica)
`false`	nessuno: va concesso a mano
`op`	solo gli operatori
`not-op`	tutti tranne gli operatori

Un sotto-comando che non dichiara un permesso riceve `<permesso del padre>.<nome del figlio>`: se `/clan` chiede `server.clan`, `/clan crea` chiede `server.clan.crea`.

> **I permessi generati sono permessi veri**
> Vengono registrati davvero nel server, quindi compaiono nel completamento di LuckPerms e negli editor di permessi: non devi ricordarteli a memoria per scriverli.

Un esempio di configurazione tipica

Come si distribuiscono di solito su un server con LuckPerms:

```
gruppo default    shadowcommand.command.regole
                  shadowcommand.command.discord
                  server.kit.legno

gruppo vip        server.kit.ferro
                  server.negozio

gruppo staff      shadowcommand.admin
                  shadowcommand.reload
                  server.staff.*

gruppo admin      shadowcommand.*
```

Nota che `shadowcommand.*` **non** include gli scavalcamenti: chi amministra il plugin non salta automaticamente i cooldown dei comandi.

12. Dipendenze e integrazioni

La regola, senza eccezioni: **nessuna dipendenza e' obbligatoria**. Un plugin che manca degrada una funzione, lo scrive in console una volta sola, e il server parte lo stesso.

Cosa serve davvero

Requisito	Obbligatorio	Nota
Paper / Purpur / Folia 1.26.2	**si**	Non funziona su Spigot puro: usiamo le API di Paper
Java 25	**si**	Lo richiede gia' Paper 1.26.2
Un database	no	H2 locale incluso nel jar. MySQL / MariaDB solo se vuoi condividere fra piu' server
Qualunque altro plugin	no	Tutto quello che segue e' facoltativo

Integrazioni opzionali

Plugin	Cosa sblocca	Se manca
PlaceholderAPI	Legge i `%...%` di chiunque, e pubblica i tuoi dati per scoreboard, tab e chat	I `%...%` restano scritti come sono
Vault	Denaro e permessi con qualunque plugin che stia dietro Vault	Si prova la suite, poi si rinuncia
LuckPerms	`permission:`, `group:`, permessi a scadenza, `{prefix}` e `{suffix}`	Con il solo Vault la durata viene ignorata
WorldGuard	Trigger di ingresso e uscita dalle regioni, requisito `region:`	I trigger di regione non vengono registrati
ShadowEconomy	Denaro **multi-valuta**: `money: give 100 gemme`	Si passa a Vault
ShadowPermissions	Permessi e gruppi, anche a tempo	Si passa a LuckPerms, poi a Vault
ShadowRegions	Regioni, come WorldGuard ma della suite	Si passa a WorldGuard
ShadowBattlePass	8 azioni, 7 requisiti e 16 segnaposto del pass	Le azioni non fanno niente, i requisiti negano
ShadowGames	8 azioni, 7 requisiti e 15 segnaposto dei minigiochi	Idem
ShadowBasics	Ci annunciamo al suo elenco della suite	Nulla cambia
ShadowCrazyChest	Un premio della cassa puo' eseguire uno script tuo	Nulla cambia

ShadowSkyBlock	Trigger sull'isola, comandi di isola scritti in YAML	Nulla cambia

Come viene scelto chi usare

Per ogni servizio c'e' un ordine di preferenza: vince il primo che risponde.

Economia

1. **ShadowEconomy** - multi-valuta
2. **Vault** - valuta unica
3. **Nessuna** - `money:` non fa niente, i requisiti `money:` sono sempre veri e i costi in denaro sono zero

Permessi

1. **ShadowPermissions**
2. **LuckPerms** - con permessi a scadenza
3. **Vault** - permessi e gruppi
4. **Nessuno** - `permission:` e `group:` non fanno niente

Regioni

1. **ShadowRegions**
2. **WorldGuard**
3. **Nessuna** - i trigger di regione non esistono e il requisito `region:` e' sempre falso

Segnaposto

1. **PlaceholderAPI** - letti e forniti
2. **Nessuno** - i `%...%` restano testo

Il banner di avvio dice sempre com'e' andata, e `/scmd suite` lo ripete in gioco:

```
|  Economia   ShadowEconomy
|  Permessi   nessun gestore di permessi
|  Regioni    ShadowRegions
|  BattlePass ShadowBattlePass 1.0.0
|  Minigiochi ShadowGames
|  Papi       assente — i segnaposto %...% non vengono sostituiti
```

Essentials, CMI e gli altri

Non serve nessuna integrazione dedicata: i loro comandi si eseguono con `console:` o `command:`, e i loro dati si leggono con PlaceholderAPI. ShadowCommand si dichiara da caricare **dopo** di loro, cosi' `overrides.yml` puo' bloccare o sostituire i loro comandi.

Per chi sviluppa plugin

ShadowCommand pubblica un'API nel `ServicesManager`: un altro plugin puo' eseguire i tuoi script, leggere e scrivere le variabili e aggiungere azioni e requisiti propri, senza compilare contro di noi.

```
ShadowCommandAPI api = Bukkit.getServicesManager().load(ShadowCommandAPI.class);
if (api != null) {
    api.run(player, List.of("message: &aCiao", "give: DIAMOND 1"));
    api.execute(player, "kit", new String[]{ "diamante" });
    api.variables().set(player.getUniqueId(), "punti", "10");
    api.actions().register(miaAzione);
    String testo = api.resolve(player, "Ciao {player}, hai {var:punti} punti");
}
```

Un'azione registrata da fuori compare subito in `/scmd actions` e nel completamento dell'editor, come se fosse nostra.

Eventi che lanciamo, tutti annullabili

Evento	Quando
`ShadowScriptPreExecuteEvent`	Prima di eseguire uno script
`ShadowScriptPostExecuteEvent`	Dopo
`ShadowCommandDeniedEvent`	Requisito, costo o cooldown negato
`ShadowVariableChangeEvent`	Una variabile cambia

13. Configurazione e problemi

Ogni valore in `config.yml` ha gia' il numero giusto per un server normale: si puo' copiare il jar e non aprire mai quel file. Qui c'e' cosa cambiare quando serve.

Le sezioni di config.yml

Sezione	A cosa serve
`licence`	La tua chiave e i giorni di tolleranza
`messages`	Lingua predefinita, lingua del client, prefisso in console
`storage`	H2 locale o MySQL, prefisso delle tabelle, salvataggio periodico
`engine`	Confronti, cooldown di fabbrica, suono di rifiuto, **limiti degli script**
`commands`	Permessi automatici, completamento, messaggi d'uso
`security`	Cosa possono fare `op:`, `bypass:` e `console:`, comandi vietati
`triggers`	Interruttore generale e frequenza dei controlli
`menus`	Aggiornamento di fabbrica, anti doppio click, profondita' massima
`variables`	Cache degli offline, pulizia dei giocatori inattivi
`logging`	Registro delle esecuzioni, debug, avviso sulle azioni lente
`compatibility`	Segnaposto con il dollaro, colori con il paragrafo, tipi di MyCommand
`integrations`	Accende o spegne singolarmente le integrazioni

I limiti degli script

Servono a garantire che un file scritto male non porti giu' il server. Alzali solo se sai perche'.

```yaml
engine:
  limits:
    max-steps: 10000            # righe eseguibili da un singolo script
    max-depth: 16               # profondita' delle funzioni che si chiamano fra loro
    max-sessions-per-player: 8  # script in attesa contemporanei per giocatore
    max-wait: 600000            # durata massima di un wait
    max-command-chain: 8        # comandi che ne lanciano altri, a catena
    action-warn-millis: 25      # oltre questa soglia l'azione lenta finisce nel log
```

La sicurezza

```yaml
security:
  op-action:
    enabled: true
    allowed-commands: [ ]       # vuoto = qualunque comando; altrimenti solo questi
    log: true                   # ogni uso finisce in console con chi, cosa e da dove
  bypass-action:
    enabled: true
  console-action:
    allowed: true

  # Il freno di emergenza: comandi che nessuno script puo' eseguire, in nessun modo
  forbidden-commands: [ "stop", "op", "deop", "reload" ]

  max-broadcast-per-second: 10
  escape-player-input: true
```

Lascia acceso `escape-player-input`
Disinnesca quello che scrivono i giocatori prima che finisca in un messaggio. Senza, un giocatore che si chiama `<click:run_command:'/kill @a'>PREMI` trasformerebbe ogni messaggio che contiene `{player}` in una trappola per chi lo legge.

Le lingue

Un file per lingua in `lang/`. Di fabbrica ogni giocatore vede la lingua del proprio client, con ritorno alla lingua predefinita quando manca. Per aggiungerne una: copia `it_IT.yml`, chiamalo con il codice giusto e traducilo.

```yaml
messages:
  default-locale: "it_IT"
  follow-client-locale: true
  prefix-in-console: true
```

I file di lingua vengono aggiornati automaticamente quando una versione nuova del plugin ne porta una piu' recente, e la tua copia viene salvata accanto con estensione `.old`. I file che scrivi tu - comandi, menu, trigger - **non vengono mai sovrascritti**.

Se qualcosa non va

Sintomo	Cosa guardare
Il plugin non si avvia	La licenza: il motivo esatto e' scritto in console dentro un riquadro. Dopo aver incollato la chiave serve un **riavvio**, non un reload
Un comando non esiste in gioco	`/scmd errors`: quasi sempre e' un errore di compilazione. Un comando con errori non viene registrato apposta
Il comando esiste ma non fa niente	Un requisito nega in silenzio: dai un `deny:` a ogni requisito, oppure guarda `/scmd debug <comando>`
Un segnaposto resta scritto com'e'	Se e' fra `%...%` manca PlaceholderAPI. Se e' fra graffe, `/scmd parse` dice cosa vale davvero

I colori non si vedono	Controlla di aver usato la e commerciale e non il paragrafo, e che il testo non sia dentro un'azione che vuole un valore numerico
Il cooldown non scatta su di te	Sei operatore? No: gli scavalcamenti non sono dati agli OP. Controlla `cooldown-bypass:` nel comando
Un trigger non parte	`/scmd triggers` mostra i listener attivi e quante volte sono scattati: se il contatore resta a zero, il filtro non passa mai
Il server rallenta	`/scmd stats time` ordina i comandi per tempo speso. Le azioni lente finiscono gia' nel log da sole
Una GUI si apre vuota	Le `view-requirements` delle icone non passano, oppure gli `slots` sono fuori dalle `rows` dichiarate
Dopo un aggiornamento manca qualcosa	Le lingue e i menu di sistema vengono aggiornati e la tua copia salvata come `.old`: confronta i due file

Prima di chiedere assistenza

Con queste tre cose il problema si risolve quasi sempre al primo messaggio:

1. l'output di `/scmd version` e di `/scmd errors`;
2. il pezzo di file YAML che non funziona, per intero;
3. l'id del server, che `/scmd licenza` mostra con il pulsante per copiarlo.

ShadowCommand 1.0.0 · parte della Shadow Suite
Se sai scrivere un file YAML, sai programmare il tuo server.