@ ? d: -> =>

Lonelog

A Standard Notation for Solo RPG Session Logging

Version 1.7.0



Table of contents

1. Introduction **1**
1.1 What Lonelog Does 1
1.2 How to Use This Notation 2
1.3 Quick Start: Your First Session 3
2. Digital vs Analog Formats **4**
2.1 Digital Format (Markdown) 4
2.2 Analog Format (Notebooks) 4
2.3 Indentation 5
2.4 Format Examples 5
3. Core Notation **6**
3.1 Actions 7
3.2 Resolutions 9
3.3 Consequences 14
3.4 Complete Action Sequences 14
4. Optional Layers **16**
4.1 Persistent Elements 16
4.2 Progress Tracking 24
4.3 Random Tables & Generators 25
4.4 Narrative Excerpts 33
4.5 Meta Notes 34
5. Optional Structure **37**
5.1 Campaign Header 38
5.2 Session Header 40
5.3 Scene Structure 41
6. Complete Examples **53**
6.1 Minimal Shorthand Log 53
6.2 Hybrid Format: Digital and Analog 54
6.3 Complete Campaign Log (Digital) 55
6.4 Complete Campaign Log (Analog) 58
7. Best Practices **60**
7.1 Good Practices ✓ 60
7.2 Bad Practices ✗ 61
8. Templates **63**
8.1 Campaign Template (Digital YAML) 64

8.2 Campaign Template (Analog) 65
8.3 Session Template 65
8.4 Quick Scene Template 66
9. Adapting to Your System 67
9.1 System-Specific Roll Notation 68
9.2 Oracle Adaptations 69
9.3 Handling Edge Cases 71
10. Add-ons 73
10.1 What Add-ons Are 73
10.2 Why Separate Files 74
10.3 How to Use Add-ons 74
10.4 Available Add-ons 75
10.5 A Note for Add-on Authors 76
Appendices 77
A. Notation Legend 77
B. FAQ 82
C. Symbol Design Philosophy 84
D. Specification Summary 84
E. History and Migration 88
Credits & License 91

1. Introduction

If you've ever played a solo RPG, you know the challenge: you're deep in an exciting scene, dice are rolling, oracles are answering questions, and suddenly you realize: how do I capture all this without breaking the flow?

Maybe you've tried free-form journaling (gets messy), pure prose (loses the mechanics), or bullet points (hard to parse later). This notation system offers a different approach: a **lightweight shorthand** that captures the essential game elements while leaving room for as much (or as little) narrative as you want.

1.1 What Lonelog Does

Lonelog is a shared language for solo play. Whether you're playing *Ironsworn*, *Thousand Year Old Vampire*, a non-solo RPG using Mythic GME, or your own homebrew system, it helps you:

- **Record what happened** without slowing down play
- **Separate mechanics from fiction**, so a reader can see how rules and oracles shaped the story
- **Track ongoing elements** like NPCs, locations, and plot threads, and find them again sessions later
- **Share your sessions** with other solo players, who can read them at a glance because the symbols are standard
- **Scale** from a one-line shorthand to a full narrative log, in a Markdown file or a paper notebook

(If you knew this system as *Solo TTRPG Notation*, see *Appendix E: History and Migration*.)

1.2 How to Use This Notation

Lonelog is a **toolbox, not a rulebook**. The system is fully modular: grab what works for you and leave the rest behind.

At its core are just **five symbols** (see *Section 3: Core Notation*). They are carefully chosen to avoid conflicts with markdown formatting and comparison operators. These are the minimal language of play:

- `@` for player actions
- `?` for oracle questions
- `d:` for mechanics rolls
- `->` for oracle/dice results
- `=>` for consequences

That's it. **Everything else is optional.**

Scenes, campaign headers, session headers, threads, clocks, narrative excerpts—these are all enhancements you can add when they serve your play. Want to track a long campaign? Add campaign headers. Need to follow complex plots? Use thread tags. Playing a quick one-shot? Stick to the five core symbols.

The system works in concentric circles:

- **Core Notation** (required): Actions, Resolutions, Consequences
- **Optional Layers** (add as needed): Persistent Elements, Progress tracking, Notes, etc.
- **Optional Structure** (for organization): Campaign Header, Session Header, Scenes

Start small. Try the core notation for one scene. If it clicks, great—keep going. If you need more, layer in what helps. Your notes should serve your play, not the other way around.

A note on licensing: This work is released under the CC BY-SA 4.0 license, that covers the Lonelog specification itself — this document. Session logs, actual plays, and other content you create using Lonelog notation are your own work and are not subject to this license. Write, publish, and license your sessions however you like.

1.3 Quick Start: Your First Session

Never used notation before? Here's everything you need:

```
S1 *Your starting scene*
@ Action you take
d: your roll result -> Success or Fail
=> What happens as a result

? Question you ask the oracle
-> Oracle's answer
=> What that means in the story
```

That's it! Everything else is optional. Try this for one scene and see how it feels.

Quick Start Example

```
S1 *Dark alley, midnight*
@ Sneak past the guard
d: Stealth 4 vs TN 5 -> Fail
=> I kick a bottle. Guard turns!

? Does he see me clearly?
-> No, but...
=> He's suspicious, starts walking toward
the noise
```

2. Digital vs Analog Formats

This notation works in **both digital markdown files and analog notebooks**. Choose the format that suits your play style.

2.1 Digital Format (Markdown)

In digital markdown files:

- **Campaign metadata** → YAML front matter (top of file)
- **Campaign Title** → Level 1 heading
- **Sessions** → Level 2 headings (`## Session 1`)
- **Scenes** → Level 3 headings (`### S1`)
- **Core notation and tracking** → Code blocks for easy copying/parsing
- **Narrative** → Regular prose between code blocks

> ***Note:*** *Always wrap notation in code blocks (```) when using digital markdown. This prevents conflicts with Markdown syntax and ensures symbols like* `=>` *render correctly. Some Markdown extensions (Mermaid, Obsidian plugins) may interpret* `=>` *outside of code blocks.*

2.2 Analog Format (Notebooks)

In paper notebooks:

- Write headers and metadata directly as shown
- Core notation works identically but without code fences
- Use the same symbols and structure
- Brackets and tags help scanning paper pages

2.3 Indentation

Indentation is optional in both digital and analog formats. It carries no structural meaning — the notation's symbols and markers do that work. But indenting related lines within a sequence can make structure scannable at a glance, and is encouraged when it helps.

```
@ Search the archives
    d: Investigation d6=5 vs TN 4 ->
Success
    => I find the shipping manifests.
[L:Archive|searched]
    [N:Viktor|implicated]
    [Thread:The Shipment|active]
```

On paper, use whatever indentation feels natural. It's a readability aid, not a rule.

2.4 Format Examples

Digital markdown

````
## Session 1
*Date: 2025-09-03 | Duration: 1h30*

### S1 *School library after hours*

```
@ Sneak inside to check the archives
d: Stealth d6=5 vs TN 4 -> Success
````
````

```
=> I slip inside unnoticed. [L:Library|
dark|quiet]
```
```

**Analog notebook**

```
=== Session 1 ===
Date: 2025-09-03 | Duration: 1h30

S1 *School library after hours*
@ Sneak inside to check the archives
d: Stealth d6=5 vs TN 4 -> Success
=> I slip inside unnoticed. [L:Library|
dark|quiet]
```

Both formats use identical notation — only the wrapping differs.

3. Core Notation

This is the heart of the system—the symbols you'll use in nearly every scene. Everything else in this document is optional, but these core elements are what make the notation work.

There are only five symbols to remember, and they mirror the natural flow of solo play: you take an action or ask a question, you resolve it with mechanics or an oracle, then you record what happens as a result.

Where symbols go. In the **standard form**, each element (`@`, `?`, `d:`, `tbl:`, `->`, `=>`) starts its own line, as in every example below. In the
```

compact form, elements follow one another on a single line, provided each symbol is preceded by a space; the space *after* a symbol is optional. Compact form is for fast play and is shown in §6.1.

3.1 Actions

In solo play, uncertainty comes from two distinct sources: **you don't know if your character can do something** (that's mechanics), or **you don't know what the world does** (that's the oracle).

This distinction is fundamental. When you swing a sword, you use mechanics to see if you hit. When you wonder whether guards are nearby, you ask the oracle. Both create uncertainty, but they're resolved differently.

The notation reflects this with two different symbols—one for each type of action.

The `@` symbol represents you, the player, acting in the game world. Think of it as 'at this moment, I…' It's visually distinct from comparison operators, making your logs clearer and avoiding confusion when recording dice rolls.

Player-facing actions (mechanics):

```
@ Pick the lock
@ Attack the guard
@ Convince the merchant
```

World / GM questions (oracle):

```
? Is anyone inside?
? Does the rope hold?
? Is the merchant honest?
```

3.1.1 Multiple Actors

When you play more than one character—a party of PCs, a PC with a companion, or a scene where you want to log NPC actions explicitly—use `@(Name)` to attribute an action to a specific actor. The bare `@` always means the primary (or default) PC.

Actor	Format	Example
Primary PC	`@ Action`	`@ Pick the lock`
Named PC	`@(Name) Action`	`@(Elara) Covers the door`
Companion / ally NPC	`@(Name) Action`	`@(Jonah) Distracts the guard`
Lead/Assist	`@(A > B) Action`	`@(Elara > Jonah) Boosts the climb`
Joint action	`@(A+B) Action`	`@(Elara+Jonah) Force the door`

```
@ Slip into the archive
@(Jonah) Keeps watch at the door
d: Stealth d6=5 vs TN 4 -> Success
=> We're inside without raising the
alarm.
```

This convention requires no new symbols—it simply extends `@` with a parenthetical to identify who is acting.

Lead/Assist — use `@(A > B)` when one character helps another and the game has a formal assistance mechanic (PbtA's *Help* move,

Ironsworn's *Aid your Ally*, etc.). `A` is the helper; `B` makes the roll. The `>` means "assists," not a comparison operator.

```
@(Elara > Jonah) Elara steadies the rope
while Jonah climbs
d: Athletics d6+1 vs TN 4 -> Success
=> Jonah reaches the ledge. Elara
follows.
```

Joint action — use `@(A+B)` when two characters contribute to a single roll together, with no clear lead/assist distinction.

```
@(Elara+Jonah) Force open the barred door
d: Might 2d6 (take highest) vs TN 5 ->
Success
=> The bar splinters. They're through.
```

When to use it: Any time ambiguity about who is acting would require prose clarification anyway: co-op solo play, games with a party, or scenes where a companion's action has its own mechanical resolution.

Note: The Combat Add-on (§4 Actor Actions) expands this convention for tactical encounters, adding enemy and group attributions and integrating it with round tracking.

3.2 Resolutions

Once you've declared an action (`@`) or asked a question (`?`), you need to resolve the uncertainty. This is where the game system or oracle gives you an answer.

There are two types of resolutions: **mechanics** (when you roll dice or apply rules) and **oracle answers** (when you ask the game world a question).

3.2.1 Mechanics Rolls

The `d:` prefix marks a roll or rule that **gives a verdict**: success or failure, yes or no, a degree of either. Every part of the line is optional except `d:` and the outcome after `->`; the outcome can be left out only when the total is itself the result, as when you generate a stat. The usual order is:

```
d: [Label] <dice>[+mod]=<total> [vs
<target>] -> <outcome>
```

The label says what you rolled for, the dice expression says what you rolled, `=` gives the total, `vs` gives what you rolled against, and `->` gives the verdict.

```
d: Stealth d6=5 vs TN 4 -> Success
d: Lockpicking d20+3=17 vs DC 15 ->
Success
d: Strike 2d6+2=9 -> Hit
d: d100=42 -> Partial success
d: Hack the terminal (spend 1 Gear) ->
Success
```

The last line has no dice at all: `d:` covers any resolution mechanic, including spending resources or drawing cards. For keep/drop, exploding dice, dice pools and individual die results, see the **Dice Notation Add-on**.

Comparison shorthand

When the comparison is the point, you can write it with operators instead of `vs`:

Operator	Meaning	Example
`≥` or `>=`	meets or beats the target	`d: 5≥4 -> Success`
`<`	falls short of the target	`d: 2<4 -> Fail`
`vs`	neutral: "against"; the outcome says who won	`d: 5 vs TN 4 -> Success`

Write the comparison so it is true: `4≥5` is never a way of writing "4 against 5". Roll-under systems do the same: `d: d20=8 ≤ 12 -> Success`.

The outcome can be abbreviated to `S` (Success) or `F` (Fail). The abbreviation still goes after `->`:

```
d: 5≥4 -> S
d: 2<4 -> F
```

3.2.2 Oracle Answers

The `->` symbol declares an outcome: "this is what it came to." On a `d:` line it carries the verdict of a roll. On its own line it carries the answer to an oracle question.

An oracle answer can be recorded in three equivalent forms. Pick the one that fits how much the roll matters:

Form	Example	Use when
Full	`d: d100=60 vs 35 -> No`	The roll and the target both matter, e.g. in shared logs or likelihood-based oracles
Compact	`-> No, but... (d6=3)`	You want the roll on record but not the target
Bare	`-> No, but...`	Only the answer matters

```
? Do they see Phoebe? (Unlikely)
d: d100=60 vs 35 -> No
=> Distracted, but one guard lingers.
[N:Guard|watchful]

? Is anyone home?
-> Yes (d6=5)
=> Lights are on inside.
```

Likelihood goes on the question, because it describes what you asked, not the roll: `? Does the rope hold? (Likely)`.

A parenthesis directly after an answer, with no label, is always a roll reference.

Answer abbreviations

For compact logs, these codes stand for the common oracle answers:

Code	Meaning	Code	Meaning
`Y`	Yes	`N`	No
`Ya`	Yes, and…	`Na`	No, and…
`Yb`	Yes, but…	`Nb`	No, but…

Code	Meaning	Code	Meaning
`Yx`	Exceptional yes	`Nx`	Exceptional no

```
? Seen? -> Nb (3)
```

Answers that don't fit a yes/no scale are written out: `-> Partially`, `-> With a cost`, `-> Strong hit`.

Verdicts and content: `d:` or `tbl:`?

Random results do one of two jobs, and the prefix tells the reader which:

- **`d:` gives a verdict.** Did it work? Is it so? The outcome is success or failure, yes or no, or a degree of either.
- **`tbl:` gives content.** What is it? Who? What happens? The outcome is a thing, a name, an event, a trait.

The number of dice doesn't matter to either. One roll or five, if what comes back is content, it's `tbl:`.

```
? Does the guard notice me? (Unlikely)
d: d100=60 vs 35 -> No

? What disrupts the scene?
tbl: Mythic Event
  Focus: d100=78 -> NPC Action
  Meaning: d100=34 -> Betray
```

`tbl:` is covered in full in §4.3.

3.3 Consequences

Record the narrative result after rolls using `=>`. The symbol shows consequences flowing forward from actions and resolutions. The double arrow visualizes how events cascade through your story.

```
=> The door creaks open, but the noise
echoes through the hall.
=> The guard spots me and raises the
alarm.
=> I find a hidden diary with a crucial
clue.
```

Multiple consequences

You can chain multiple consequence lines for cascading effects:

```
d: Lockpicking 5≥4 -> Success
=> The door opens
=> But the hinges squeal loudly
=> Guards in the next room hear it
[Clock:Alert 1/6]
```

Most actions have one clear consequence. Chain lines when a major success or failure ripples outward.

3.4 Complete Action Sequences

Here's how the core elements combine:

Mechanics-driven sequence

```
@ Pick the lock
d: d20+Lockpicking=17 vs DC 15 -> Success
=> The door creaks open, but the noise
echoes through the hall.
```

Oracle-driven sequence

```
? Is anyone inside?
-> Yes, but... (d6=4)
=> Someone is here, but they're
distracted.
```

Combined sequence

```
@ Sneak past the guards
d: Stealth 2<4 -> Fail
=> My foot kicks a barrel. [Clock:Alert
2/6]

? Do they see me?
-> No, but... (d6=3)
=> Distracted, but one guard lingers
nearby. [N:Guard|watchful]
```

4. Optional Layers

You've got the basics—actions, rolls, and consequences. That's enough for simple play. But longer campaigns often need more: NPCs who reappear, plot threads that weave through sessions, progress that accumulates over time.

This section covers the **tracking elements** that help you manage complexity. They're all optional. If you're playing a one-shot mystery, you might not need any of this. If you're running a sprawling campaign with dozens of NPCs and multiple plot threads, you'll probably want most of it.

Pick and choose based on what your campaign needs.

4.1 Persistent Elements

As your campaign grows, certain things stick around: NPCs who reappear, locations you return to, ongoing threats, story questions that span sessions. These are your **persistent elements**.

Tags let you track them consistently across scenes and sessions. The format is simple: brackets, a type prefix, a name, and optional details. Like this: `[N:Jonah|friendly|wounded]`.

Why use tags?

- **Searchability**: Find every scene where Jonah appears
- **Consistency**: Reference NPCs the same way every time
- **Status tracking**: See how elements change over time
- **Memory aid**: Remind yourself of details weeks later

You don't need to tag everything—only what matters to your campaign. A random merchant you'll never see again? Just call them "the merchant" in prose. A recurring villain? Definitely tag them.

Every tag has a name after the type prefix, player characters included: `[PC:Alex|HP 6]`, never `[PC:HP 6]`.

Tags accumulate. A later tag adds to what's already known about the element; you never need to restate everything. Four rules cover every change:

1. **Plain values add up.** Use `-value` to remove one and `old>new` to replace one.
2. **A category holds one state.** A value written as `category:value` (§4.1.7) replaces whatever that category held before. Use it for anything that can only be one thing at a time: a status, a position, a mood.
3. **Numbers are set or adjusted.** `HP 6` sets a value; `HP-2` or `HP+3` adjusts it.
4. **Counts are numbers too.** `×12` is a count: a new count replaces the old one, and `×-1` or `×+2` adjusts it.

- `[N:Jonah|friendly|injured]` — Establishes Jonah as friendly and injured
- `[N:Jonah|captured]` — Adds *captured*; he is still friendly and injured
- `[N:Jonah|-injured]` — Removes *injured*
- `[N:Jonah|friendly>hostile]` — Replaces *friendly* with *hostile*
- `[N:Guard|status:alert]` then `[N:Guard|status:unconscious]` — The guard is now unconscious, and no longer alert
- `[PC:Alex|HP-2|Gold-5]` — Lowers HP by 2 and Gold by 5
- `[N:Bandits|×6]` then `[N:Bandits|×4]` — Six bandits, then four; the name stays the same

`>` and `→` mean the same thing inside a tag; use whichever you can type. Keep counts out of names: `[N:Bandits|×6]`, not `[N:Bandits×6]`, so the element keeps one identity as its numbers change. To reset an element entirely, restate it as a multi-line tag (§4.1.8), which reads as a full status block.

Here are the main types of persistent elements you might track:

4.1.1 NPCs

```
[N:Jonah|friendly|injured]
[N:Guard|watchful|armed]
[N:Merchant|suspicious]
```

Update NPCs with the accumulation rules above. A leading `+` (`[N:Jonah|+captured]`) is allowed when you want to stress that something is new, but it means the same as `[N:Jonah|captured]`.

4.1.2 Locations

```
[L:Lighthouse|ruined|stormy]
[L:Library|dark|quiet]
[L:Tavern|crowded|noisy]
```

4.1.3 Events

An event tag marks **something notable that happened**: a turning point you'll want to find again.

```
=> The bridge gives way behind me.
[E:Bridge Collapse]
=> The duke is dead. [E:Duke
Assassinated]
```

Search a campaign for `[E:` and you get its milestones. Anything that *builds up* toward a result is a clock, track or timer instead (§4.2).

(Logs written before v1.7 may use `[E:Name X/Y]` with a counter. Read it as `[Clock:Name X/Y]`.)

4.1.4 Story Threads

```
[Thread:Find Jonah's Sister|Open]
[Thread:Discover the Conspiracy|Open]
[Thread:Escape the City|Closed]
```

Threads track major story questions or goals. Common states:

- `Open` — active thread
- `Closed` — resolved thread
- `Abandoned` — dropped thread
- Custom states allowed (e.g., `Urgent`, `Background`)

A thread holds one state at a time, so change it by replacing it: `[Thread:Escape the City|Open>Closed]`.

4.1.5 Player Character

```
[PC:Alex|HP 8|Stress 0|
Gear:Flashlight,Notebook]
[PC:Elara|HP 15|Ammo 3|Status:Wounded]
```

Updating PC stats:

- `[PC:Alex|HP 8]` — Sets HP to 8
- `[PC:Alex|HP-2]` — Lowers HP by 2, to 6
- `[PC:Alex|HP 6]` — Sets HP to 6, the same result written explicitly
- `[PC:Alex|HP+3|Stress-1]` — Several changes at once

4.1.6 Reference Tags

To reference a previously established element without restating tags, put `#` before the type. The first mention establishes the element:

```
[N:Jonah|friendly|injured]
```

Later in the log, a reference points back to it:

```
[#N:Jonah]
```

The `#` works with every tag type: `[#L:Lighthouse]`, `[#Thread:Lighthouse Mystery]`, `[#Clock:Alert]`. It tells you this element was defined earlier. Use it to:

- Keep later mentions concise
- Signal to readers they should look back for context
- Maintain searchability (the ID "Jonah" still appears)

When to use reference tags:

- First mention: Full tag with details `[N:Name|tags]`
- Later mentions in same scene: Optional, use judgment
- Later mentions in different scenes/sessions: Use `[#N:Name]` to signal reference
- Status changes: Drop the `#` and show new tags `[N:Name|new_tags]`

4.1.7 Tag Categories

When a tag holds values across distinct types, group them with category prefixes. The category name is followed by `:` and then a comma-separated list of values that belong to it:

```
```

```
[PC:Jonah|trait:friendly,curious|
status:wounded|stat:HP 8]
[N:Guard|role:watchful|
status:armed,alert]
```

Categories are freeform — use whatever labels your game's vocabulary provides. This is especially useful for games where tags have explicit types (like Power Tags and Weakness Tags in City of Mist, or Aspects and skills in Fate). Plain tags without a category prefix work exactly as before. These two lines hold the same information, the second one grouped:

```
[N:Jonah|friendly|injured]
[N:Jonah|trait:friendly|status:injured]
```

A category holds one state: writing it again replaces what it held (§4.1). `[N:Jonah|status:captured]` means Jonah is now captured and no longer injured. When a category holds a list, rewrite the list, or add and remove single entries with `+` and `-` inside it:

```
[PC:Jonah|trait:friendly,curious]
[PC:Jonah|trait:+reckless]
friendly, curious, reckless
[PC:Jonah|trait:-curious]
friendly, reckless
```

4.1.8 Multi-Line Tags

For characters with many tags, the single-line form can become hard to read. Break a tag across lines using the same `|` separator, with the closing `]` on its own line:

```
[PC:Jonah
  | trait: friendly, curious, reckless
  | weakness: naive, easily distracted
  | status: wounded
  | stat: HP 8, Stress 2
]
```

Multi-line and single-line form are equivalent — use whichever is readable at the current level of complexity. Multi-line tags work naturally at session start (character sheets and status blocks) or whenever a character's state becomes complex enough to warrant it.

Analog format: Indent the continuation lines and draw a closing bracket at the end of the last line:

```
[PC:Jonah
  trait: friendly, curious | status:
wounded
  stat: HP 8, Stress 2]
```

4.1.9 Roll Context

When a roll draws on specific tags, traits, or situational modifiers — especially in tag-heavy games like PbtA or City of Mist — list them **before the dice**: `+[…]` for what helps, `-[…]` for what hinders.

```
```

```
d: Investigate +[Be kind to others,
Naive] 2d6=8 -> Mixed
d: Stealth +[cover] -[injured] d6=3 vs TN
4 -> Fail
d: Persuade +[silver tongue] -
[suspicious] 2d6=9 -> Strong Hit
```

Roll context means "these tags are **active for this roll**." It is temporary: it records what contributed to the roll without changing the character's persistent state, unlike the `-tag` update on a standalone tag line (§4.1).

Use category prefixes inside the brackets when your game distinguishes types of modifiers:

```
d: Investigate +[power: Be kind to
others, Naive] -[weakness: reluctant to
talk] 2d6=8 -> Mixed
```

Brackets *after* the dice expression belong to the Dice Notation Add-on, which uses them for individual die results (`2d6[4, 4]=8`).

(Logs written between v1.3 and v1.6 place roll context after the dice, as in `d: Investigate 2d6 [Naive] = 8`. That form is still readable.)

When to use: Only when the active tags aren't obvious from the action line, or when your system requires explicitly tracking which tags contributed to a roll. For most games, the action line alone is sufficient.

4.2 Progress Tracking

Some things in your campaign don't happen all at once—they build up over time. The ritual takes twelve steps to complete. The guards' suspicion grows with each noise you make. Your escape plan inches forward. The air supply counts down.

Progress tracking gives you a visual way to see these accumulating forces. Three formats handle different types of progression:

Clocks (fill up toward completion):

```
[Clock:Ritual 5/12]
[Clock:Suspicion 3/6]
```

Use for: Threats building, spells preparing, danger accumulating. When the clock fills, something happens (usually bad for you).

Tracks (progress toward a goal):

```
[Track:Escape 3/8]
[Track:Investigation 6/10]
```

Use for: Your progress on projects, journey advancement, long-term goals. When the track fills, you succeed at something.

Timers (count down toward zero):

```
[Timer:Dawn 3]
[Timer:AirSupply 5]
```

Use for: Deadlines approaching, resources depleting, time pressure. When it hits zero, time's up.

The difference? Clocks and tracks both go up, but clocks are threats (bad when full) and tracks are progress (good when full). Timers go down and create urgency.

Advancing them: progress tags carry their counter right after the name. Write the new value, or the change:

```
[Clock:Suspicion 3/6]     sets it to 3 of
6
[Clock:Suspicion+1]       advances it by
one
[Track:Escape+2]          two steps of
progress
[Timer:Dawn-1]            one step closer
to zero
```

You don't need to track everything numerically. Only use these when the accumulation matters to your story and you want a concrete way to measure it.

4.3 Random Tables & Generators

Solo play thrives on surprise. Sometimes you roll on a table to see what you find, or use a generator to create an NPC on the fly. When you do, it helps to record what you rolled—both for transparency and so you can recreate the logic later.

`tbl:` records any random result that **gives content**: a thing, a name, an event, a trait (§3.2.2). A single roll on a simple table and a multi-step generator are both `tbl:`.

```
tbl: Loot d100=42 -> A broken sword
tbl: Merchant Mood d20=15 -> Nervous
tbl: SWN NPC d8=3,d10=7 -> Gruff / Pilot
```

The table name comes first, then the roll, then `->` and the result. The roll is optional, and so are quotes around the result:

```
tbl: Loot -> A broken sword
```

Integrating with oracle questions:

```
? What do I find in the chest?
tbl: Loot d100=42 -> A broken sword
=> An ancient blade, snapped in two, with
strange runes on the hilt.
```

(Logs written before v1.7 may use `gen:` for generators. Read it as `tbl:`.)

Why record the rolls? Three reasons:

1. **Transparency**: If you're sharing the log, others see your process
2. **Reproducibility**: You can trace how you got surprising results
3. **Learning**: Over time, you see which tables you use most

That said, if you're playing fast and loose, you can skip the roll details and just record the result: `tbl: Loot -> A broken sword`. The important part is the fiction, not the math.

4.3.1 Inline Table Definitions

The examples above assume your table lives somewhere else — a rulebook, a supplement, a separate file. You roll, you record the result, and anyone reading your log has to trust you (or own the same book) to verify it.

But what if you made the table yourself? What if you filtered options from a larger set to fit your campaign? What if you're playing a game where content generation *is* the game — systems like Bivius Companion, homebrew oracles, or any setup where the possibility space is part of the creative act?

In those cases, embedding the table directly in your log makes it **self-contained**. Readers see the full option space *and* the result. No external references, no "see page 47."

Format:

```
tbl: TableName (die)
  1: Result one
  2: Result two
  3: Result three
  4: Result four
  5: Result five
  6: Result six
```

The table name and die type go on the first line. Each entry is indented with its number and result. A `tbl:` line **without** `->` is a definition; a `tbl:` line **with** `->` is a result. Roll against the table normally:

```
tbl: TableName d6=3 -> Result three
```

Complete example:

```
tbl: Forest Encounter (d6)
  1-2: Nothing — eerie silence
  3: Animal tracks, fresh
  4: Abandoned campsite
  5: Traveler on the road
  6: Something is following you

? What do I encounter on the forest path?
tbl: Forest Encounter d6=5 -> Traveler on
the road
=> A cloaked figure waves me down.
[N:Traveler|unknown|friendly?]
```

When to define inline vs. reference externally:

- **Inline** — when you created the table, when the table is short (roughly 10 entries or fewer), when shareability matters, or when the table only exists in your head
- **External** — when you're rolling on a published table that readers can look up, or when the table is too long to include without cluttering your log

For tables you use often, define them once at the start of a session or campaign, then reference them by name throughout play:

```
tbl: Forest Encounter d6=5 -> Traveler on
the road
```

If the table was defined earlier in the log, readers can scroll back to find it. If it's a published table, the name and die type provide enough context to locate the source.

4.3.2 Filtered Option Sets

Some games don't use numbered tables — they use curated lists you pick or draw from. You might filter a larger set of options down to the ones relevant to your scene, then select randomly or intuitively.

Format:

```
tbl: TableName [Option A, Option B,
Option C, Option D]
```

Square brackets signal "these are the options in play." No numbers, no die — just the possibility space.

Rolling against a filtered set:

```
tbl: Mood [Tense, Melancholic, Hopeful,
Uncanny]
tbl: Mood -> Uncanny

tbl: Weather [Clear, Fog, Rain, Storm]
tbl: Weather d4=2 -> Fog
=> A thick fog rolls in from the coast.
Visibility drops to nothing.
```

Building a filtered set from a larger source:

```
(source: Bivius Companion themes,
filtered for this arc)
tbl: Theme [Betrayal, Redemption,
Sacrifice, Secrets]
```

```
tbl: Theme -> Sacrifice
=> The scene will center on what someone
is willing to give up.
```

Dynamic filtering mid-session:

```
tbl: Available Leads [The dockworker's
tip, The torn letter, The locked room]
tbl: Available Leads -> The torn letter
=> I follow up on the letter I found in
Session 2.
[Thread:Torn Letter|Open]
```

The key difference from numbered tables: filtered sets capture *what was available*, not just what was chosen. This is especially valuable when you're sharing logs — readers see the roads not taken alongside the one you picked.

Rolling on a list: add a die after the name, and the entries are numbered 1, 2, 3… in the order written. The die should match the length of the list:

```
tbl: NPC Role (d6) [Guard, Merchant,
Scholar, Beggar, Noble, Priest]
tbl: NPC Role d6=2 -> Merchant
```

4.3.3 Multi-Line Result Blocks

Some generators produce compound results — multiple axes of meaning that together create something greater than any single roll. An NPC might have a role, a personality trait, and a motivation. A

location might have a feature, a mood, and a secret. Recording each axis makes the creative logic transparent.

Format:

```
tbl: GeneratorName
  Axis1: roll -> result
  Axis2: roll -> result
  Axis3: roll -> result
```

Each axis is indented under the generator name. Roll details are optional — include them when transparency matters, skip them when speed matters.

NPC generator example:

```
tbl: NPC (custom)
  Role: d6=3 -> Merchant
  Trait: d6=5 -> Secretive
  Want: d6=1 -> Escape
=> [N:Unnamed Merchant|secretive|wants to
flee town]
```

Location generator example:

```
tbl: Ruin (custom d6 tables)
  Feature: d6=4 -> Collapsed tower
  Mood: d6=2 -> Oppressive silence
  Secret: d6=6 -> Hidden passage beneath
the rubble
```

```
=> [L:Old Watchtower|collapsed|eerie|
hidden passage]
```

With inline table definitions — you can combine these features. Define the axes, then roll:

```
tbl: NPC Role (d6) [Guard, Merchant,
Scholar, Beggar, Noble, Priest]
tbl: NPC Trait (d6) [Nervous, Secretive,
Boisterous, Cold, Kind, Obsessive]
tbl: NPC Want (d6) [Escape, Revenge,
Wealth, Knowledge, Power, Peace]

tbl: NPC
  Role: d6=2 -> Merchant
  Trait: d6=6 -> Obsessive
  Want: d6=4 -> Knowledge
=> [N:The Collector|merchant|obsessive|
seeks forbidden texts]
```

Minimal format — when you just need the output:

```
tbl: NPC -> Merchant / Secretive / Escape
=> [N:Unnamed Merchant|secretive|wants to
flee]
```

Use the expanded multi-line format when you want to show your work — especially useful in shared logs, for generators you created yourself, or when you want to trace how the fiction emerged from

the mechanics. Use the minimal single-line format when speed matters more than process.

4.4 Narrative Excerpts

You don't need to write narrative at all. The shorthand captures everything mechanically. But sometimes the fiction demands more—a piece of dialogue that's too perfect not to record, a description that sets the mood, a document your character finds.

That's what narrative excerpts are for: the moments where shorthand isn't enough.

Inline prose (short descriptions):

```
=> The room reeks of mildew and decay.
Papers are scattered everywhere.
```

Use for: Quick atmospheric details, sensory information, emotional beats. Keep it short—a sentence or two.

Dialogue (conversations worth recording):

```
N (Guard): "Who's there?"
PC: "Stay calm... just stay calm."
N (Guard): "Show yourself!"
PC: [whispers] "Not happening."
```

Use for: Memorable exchanges, character voice, important conversations. You don't need to record every word—just the exchanges that matter.

Long narrative blocks (found documents, important descriptions):

```

```

```
\---
The diary reads:
"Day 47: The tides no longer obey the
moon. The fish have stopped
coming. The lighthouse keeper says he
sees lights beneath the waves.
I fear for our sanity."
---\
```

Use for: In-world documents, lengthy descriptions, key revelations. The `\---` and `---\` markers separate it from your log, making it clear this is in-fiction content. The asymmetric delimiters prevent conflicts with Markdown horizontal rules.

How much narrative should you write? Only as much as serves you. If you're playing for yourself and shorthand tells you everything you need to remember, skip the prose. If you're sharing your log or you love the writing process, add more. There's no right amount—just what makes your log useful and enjoyable to you.

4.5 Meta Notes

Sometimes you need to step outside the fiction and leave yourself a note: a reminder about a house rule you're testing, a reflection on how a scene felt, a question to revisit later, or a clarification about your interpretation of a rule.

That's what meta notes are for—your out-of-character asides to yourself (or to readers, if you're sharing).

Format: a label, a colon and the text, inside parentheses: `(label: text)`. The parentheses signal "this is meta, not fiction."

```
(note: testing alternate stealth rule
where noise increases Alert clock)
(reflection: this scene felt tense! the
timer really worked)
(house: giving advantage on familiar
terrain)
(todo: revisit this thread next session)
(question: should I have rolled for that?
seemed obvious)
```

Bare parentheses without a label still read as a meta note, but a label tells the reader what kind of aside it is.

Suggested labels

These labels are **suggestions, not rules**, and the list isn't complete. Use them when they fit, and invent your own when they don't: any lowercase word followed by `:` works. The point of a shared set is that two players writing about the same thing reach for the same word.

Label	Use for	Example
`note:`	Anything; the default when no other label fits	`(note: playing with the alt stealth rule)`
`house:`	A house rule or rule variant in use	`(house: advantage on familiar terrain)`
`ruling:`	An on-the-spot decision about how a rule applies	`(ruling: the rope counts as climbing gear)`

Label	Use for	Example
`interp:`	How you read an ambiguous oracle or table result	`(interp: honest, but hiding something)`
`retcon:`	A correction to something already logged	`(retcon: forgot +1 from cover — Success)`
`todo:`	Something to follow up later	`(todo: revisit the torn letter next session)`
`question:`	Something you're unsure about, rules or fiction	`(question: should I have rolled for that?)`
`reflection:`	How the play felt; what worked and what didn't	`(reflection: the timer made this tense)`
`source:`	Where a table, rule or prompt came from	`(source: Mythic GME 2e)`
`break:`	A real-world pause in play	`(break: 20 min)`

`reminder:` and `house rule:`, found in older logs, read as `todo:` and `house:`.

Retcons and corrections

Everyone misreads a rule or forgets a modifier now and then. Don't edit the log silently: leave the original line as it was, and add a `retcon:` note directly after it.

```
d: Stealth d6=3 vs TN 4 -> Fail
(retcon: forgot +1 from cover — actually
4 vs TN 4, Success)
=> I slip past unnoticed.
```

Keeping the original line means the log shows both what happened and what was corrected. The same form works on paper.

When NOT to use meta notes: Don't let them overwhelm your log. If you're stopping every few lines to reflect, you're probably over-thinking it. The game is the thing—meta notes are occasional margin comments, like a director's commentary you only switch on now and then.

5. Optional Structure

So far we've talked about *what* you write (actions, rolls, tags). Now let's talk about *how you organize it.*

Structure helps in two ways: it makes your notes easier to navigate, and it signals boundaries (this session ended, that scene began). But structure adds overhead—more headers to write, more formatting to maintain.

This section shows you the organizing elements: campaign headers (metadata about your whole campaign), session headers (marking play sessions), and scene structure (the basic unit of play). Use what helps you stay oriented without slowing you down.

The key difference? **Digital and analog formats handle structure differently.** Digital markdown uses headings and YAML; analog notebooks use written headers and markers. We'll show both.

5.1 Campaign Header

Before you dive into play, it helps to record some basics: What are you playing? What system? When did you start? It's the cover page of your campaign log.

This is especially useful when:

- You're running multiple campaigns (helps you remember which is which)
- You're sharing logs with others (they need context)
- You return to a campaign after a break (reminds you of tone/themes)

If you're just trying out the notation with a quick one-shot, skip this entirely. But for campaigns you plan to revisit, a header is worth the 30 seconds.

Digital and analog formats differ here. Digital markdown uses YAML front matter (structured metadata at the top of the file). Analog notebooks use a written header block.

For digital markdown files, use YAML front matter at the very top, between two `---` lines:

```
---
title: Clearview Mystery
ruleset: Loner + Mythic Oracle
genre: Teen mystery / supernatural
player: Roberto
pcs: Alex [PC:Alex|HP 8|Stress 0|
Gear:Flashlight,Notebook]
```

```
start_date: 2025-09-03
last_update: 2025-10-28
tools: Oracles - Mythic, Random Event
tables
themes: Friendship, courage, secrets
tone: Eerie but playful
notes: Inspired by 80s teen mystery shows
---
```

For analog notebooks, write a campaign header block:

```
=== Campaign Log: Clearview Mystery ===
[Title]        Clearview Mystery
[Ruleset]      Loner + Mythic Oracle
[Genre]        Teen mystery /
supernatural
[Player]       Roberto
[PCs]          Alex [PC:Alex|HP 8|Stress
0|Gear:Flashlight,Notebook]
[Start Date]   2025-09-03
[Last Update]  2025-10-28
[Tools]        Oracles: Mythic, Random
Event tables
[Themes]       Friendship, courage,
secrets
[Tone]         Eerie but playful
[Notes]        Inspired by 80s teen
mystery shows
```

Optional fields (add as needed):

- `[Setting]` — Geographic or world details
- `[Inspiration]` — Media that inspired the campaign
- `[Safety Tools]` — X-card, lines/veils, etc.

5.2 Session Header

A session header marks the boundary between play sessions and provides context: when did you play, how long, what happened?

Why use session headers?

- **Navigation**: Jump to specific sessions quickly
- **Context**: Remember when you played and what was happening
- **Reflection**: Track your play patterns (how often? how long?)
- **Continuity**: Connect sessions with recaps and goals

When to skip them:

- One-shot games (no multiple sessions)
- Continuous play (you play daily with no clear breaks)
- Pure shorthand logs (you just want the fiction, not the meta-structure)

Like campaign headers, digital and analog formats handle sessions differently. Choose the style that fits your medium.

5.2.1 Digital format (markdown heading)

```
## Session 1
*Date: 2025-09-03 | Duration: 1h30 |
Scenes: S1-S2*

**Recap:** First session, introducing
Alex and the mystery.
```

```
**Goals:** Set up the central mystery,
establish the lighthouse as key location.
```

5.2.2 Analog format (written header)

```
=== Session 1 ===
[Date]        2025-09-03
[Duration]    1h30
[Scenes]      S1-S2
[Recap]       First session, introducing
Alex and the mystery.
[Goals]       Set up the central mystery,
establish the lighthouse.
```

Optional fields:

- `[Mood]` — Planned or actual tone for the session
- `[Notes]` — Rules variants, experiments, or special conditions
- `[Threads]` — Active threads this session

5.3 Scene Structure

Scenes are the basic unit of play within a session. At its simplest, a scene is just a numbered marker with context.

Digital format (markdown heading):

```
### S1 *School library after hours*
```

Analog format:

```
S1 *School library after hours*
```

The scene number helps you track progression and reference events later. The context (in italics/asterisks) frames where and when the scene takes place.

5.3.1 Sequential Scenes (Standard)

Most campaigns use simple sequential numbering:

```
S1 *Tavern, evening*
S2 *Town square, midnight*
S3 *Forest path, dawn*
S4 *Ancient ruins, midday*
```

When to use: Default for linear play. Scene 2 happens after Scene 1, Scene 3 after Scene 2, etc.

Numbering: Start at S1 each session, or continue across the whole campaign (S1-S100+).

Example in play:

```
S1 *Tavern common room, evening*
@ Ask the barkeep about rumors
d: Charisma d6=5 vs TN 4 -> Success
=> He leans in close and tells me about
strange lights at the old mill.
[Thread:Strange Lights|Open]

S2 *Outside the tavern, night*
```

```
@ Head toward the mill
? Do I encounter anything on the way?
-> Yes, but... (d6=4)
=> I see a shadowy figure, but they don't
seem hostile.
[N:Stranger|mysterious|watching]
```

5.3.2 Flashbacks

Flashbacks show past events that inform the current story. Use letter suffixes branching from the "present" scene.

Format: S#a, S#b, S#c

When to use:

- Revealing backstory mid-session
- Character memory triggers
- Showing how something happened
- Explaining mysterious elements

Example structure:

```
S5 *Investigating the mill*
=> I find my father's old journal.

S5a *Flashback: Father's workshop, 10
years ago*
=> Father: "Promise me you'll never go to
the mill alone."

S6 *Back at the mill, present day*
```

Complete example:

```
S8 *Lighthouse keeper's quarters*
@ Search the desk for clues
d: Investigation d6=6 vs TN 4 -> Success
=> I find a faded photograph. It's... my
mother? She's standing at this
lighthouse!
[Thread:Mother's Connection|Open]

S8a *Flashback: Home, 15 years ago*
(note: memory triggered by the
photograph)
? Did mother ever mention a lighthouse?
-> Yes, but... (d6=5)
=> She mentioned it once, briefly, then
changed the subject quickly.

PC (Young me): "Mom, where is this?"
N (Mother): [nervous] "Just an old place.
Nothing important."

S8b *Flashback: Mother's study, 14 years
ago*
? Was I snooping in her papers?
-> Yes, and... (d6=6)
=> I found a deed. The lighthouse
belonged to our family!
[Clock:Lighthouse Secret 1/4]

S9 *Lighthouse keeper's quarters,
present*
```

```
=> Armed with this memory, I search more
carefully for family records.
```

Numbering tips:

- Branch from the scene that triggers the flashback
- Return to sequential numbering afterward
- Keep flashbacks short (1-3 scenes usually)
- Note in context when returning: `*Present day*` or `*Back at the...*`

5.3.3 Parallel Storylines

When several storylines run at the same time, or you alternate between them, prefix each scene with its storyline number.

Format: `T#-S#`, where `T#` is the storyline number and `S#` is the scene number within that storyline.

Parallel storylines are not the same as story threads. A `[Thread:…]` tag (§4.1.4) tracks a story *question*; a `T#` prefix tracks a line of *play*.

When to use:

- Multiple characters/viewpoints
- Simultaneous events in different locations
- Alternating between plot lines
- Separate but related story arcs

Example structure:

```
T1-S1 *Main character at the lighthouse*
T2-S1 *Meanwhile, ally in the city*
T1-S2 *Back to lighthouse*
```

```
T2-S2 *Back to city*
T1-S3 *Lighthouse, continuing*
```

Who `@` means. Inside a storyline, a bare `@` means that storyline's point-of-view character. Declare the POV once, at the end of the storyline's first scene line: `T2-S1 *City archives* @(Jordan)`. Without a declaration, `@` keeps its usual meaning, the primary PC (§3.1.1).

Complete example:

```
=== Session 3 ===
[Storylines] Main story (T1), City
investigation (T2)

T1-S1 *Lighthouse tower, dusk* @(Alex)
@ Climb to the top
d: Athletics d6=4 vs TN 4 -> Success
=> I reach the top. The light mechanism
is still functional!

? Is anyone else here?
-> No, but... (d6=3)
=> Fresh footprints in the dust lead
down.

T2-S1 *City archives, same time*
@(Jordan)
@ Search for lighthouse records
d: Research d6=6 vs TN 4 -> Success
=> I find construction documents from
```

```
1923. There's a hidden basement!
[E:Basement Plans Found]

T1-S2 *Lighthouse basement stairs*
@ Follow the footprints down
d: Stealth d6=3 vs TN 5 -> Fail
=> A step creaks loudly.

? Does someone react?
-> Yes, and... (d6=6)
=> A voice from below: "Who's
there?" [N:Cultist|hostile|armed]

T2-S2 *City archives, moments later*
@ Call Alex to warn about the basement
? Does the call go through?
-> No, and... (d6=2)
=> No signal. The lighthouse is in a dead
zone!
[Clock:Alex in Danger 2/6]

T1-S3 *Lighthouse basement*
@ Try to talk my way out
d: Deception d6=2 vs TN 5 -> Fail
=> The cultist isn't buying it. He
advances with a knife!
```

When storylines merge:

Continue with plain scene numbers, numbering on from the highest scene number used so far, and say so in a note. From the merge on, use `@(Name)` for each PC, as in any multi-PC scene.

```
T1-S6 *Alex escapes the lighthouse*
T2-S6 *Jordan drives toward the
lighthouse*

S7 *Lighthouse entrance* (note:
storylines T1 and T2 merge)
[PC:Alex|wounded] [PC:Jordan|worried]
@(Jordan) Helps Alex into the car
```

5.3.4 Montages and Time Cuts

For activities that span time or multiple quick vignettes, use decimal notation.

Format: `S#.#` (e.g., `S5.1`, `S5.2`, `S5.3`)

When to use:

- Traveling across long distances
- Training/research over weeks
- Multiple quick encounters
- Gathering resources
- Time-lapse sequences

Example structure:

```
S7 *Beginning the journey*
S7.1 *Day 1: Forest*
S7.2 *Day 3: Mountains*
S7.3 *Day 5: Desert*
S8 *Arriving at destination*
```

Complete example:

```
S12 *Preparing for the ritual*
=> I need to gather three components
across the region.
[Track:Ritual Components 0/3]

S12.1 *Herb shop, morning*
@ Buy sacred herbs
d: Persuasion d6=5 vs TN 4 -> Success
=> The herbalist gives me a discount.
[Track:Ritual Components 1/3]
[PC:Alex|Gold-5]

S12.2 *Blacksmith, afternoon*
@ Obtain silver dagger
? Is it in stock?
-> No, but... (d6=4)
=> He can make one by tomorrow.
[Timer:Ritual Deadline 2]

S12.3 *Graveyard, midnight*
@ Collect cemetery soil
? Am I interrupted?
-> Yes, and... (d6=6)
=> The groundskeeper catches me AND calls
the guard!
[Clock:Suspicion 3/6]

@ Run and hide
d: Stealth d6=6 vs TN 5 -> Success
=> I escape with the soil.
[Track:Ritual Components 2/3]
```

```
S13 *Blacksmith shop, next morning*
=> I collect the silver dagger.
[Track:Ritual Components 3/3]
```

Travel montage example:

```
S8 *Setting out from Port Ashan*
=> Three-week journey to the Northern
Wastes begins.

S8.1 *Week 1: Coastal road*
? Who do I meet on the road?
tbl: Road Encounter d100=23 -> Merchant
caravan
=> I join a caravan for safety.
[N:Merchants|friendly]

S8.2 *Week 2: Mountain pass*
? Weather problems?
-> Yes, and... (d6=6)
=> Blizzard hits. The pass is blocked!
[Clock:Supplies Dwindle 2/4]

@ Find shelter
d: Survival d6=5 vs TN 5 -> Success
=> I locate a cave. [L:Mountain Cave|
shelter|dark]

S8.3 *Week 3: Descending into wastes*
```

```
@ Navigate the frozen terrain
d: Survival d6=4 vs TN 6 -> Fail
=> I'm lost for two days.
[Clock:Supplies Dwindle 4/4]
[PC:Alex|out of rations]

S9 *Arriving at the Northern Wastes*
=> Exhausted and hungry, but I've made
it.
```

5.3.5 Choosing Your Approach

Use sequential (S1, S2, S3) when:

- Playing straightforward, linear story
- Don't need complex time manipulation
- Want simplicity
- Most common choice

Use flashbacks (S5a, S5b) when:

- Revealing backstory mid-game
- Character development moments
- Explaining mysteries
- Short diversions from main timeline

Use parallel storylines (T1-S1, T2-S1) when:

- Playing multiple characters
- Tracking simultaneous events
- Alternating between locations
- Complex, interwoven plots

Use montages (S7.1, S7.2) when:

- Covering long time periods
- Series of quick scenes

- Travel sequences
- Resource gathering
- Training/research periods

5.3.6 Scene Context Elements

Beyond numbering, enrich scenes with context in the frame:

Location:

```
S1 *Lighthouse tower*
S1 [L:Lighthouse] *Tower room*
```

Time markers:

```
S1 *Lighthouse, midnight*
S1 *Lighthouse, Day 3, dusk*
S1 *Two weeks later: Lighthouse*
```

Emotional tone:

```
S1 *Lighthouse (tense)*
S1 *Lighthouse - moment of calm*
```

Multiple elements:

```
S1 *Lighthouse tower, midnight, Day 3*
S5a *Flashback: Father's workshop, 10
years ago*
T2-S3 *Meanwhile in the city, same
```

```
evening*
S7.2 *Day 2 of journey: Mountain pass*
```

Minimal (just the number), with the context given in the first action or consequence:

```
S1
```

Choose the level of detail that helps you track your story. More detail helps future reference; less detail keeps notes cleaner.

6. Complete Examples

Theory is one thing, but seeing the notation in action is where it clicks. This section shows complete play examples in different styles—from ultra-compact shorthand to rich narrative logs—so you can find the approach that works for you.

Each example demonstrates the same notation system, just with different levels of detail. Pick the style that matches your preference, or mix and match as your session demands.

6.1 Minimal Shorthand Log

Compact form (§3), no formatting — perfect for fast play:

```
S1 @Sneak d:4<5 ->F => noise [Clock:Alert
1/6] ?Seen? ->Nb (3) => distracted
S2 @Search d:6≥4 ->S => find key
```

```
[Track:Clues 1/4] ?Trapped? ->Ya (6) =>
spikes, and they're poisoned!
S3 @Dodge d:3<5 ->F => [PC:Alex|HP-2] =>
bleeding, need to retreat
```

6.2 Hybrid Format: Digital and Analog

Shorthand mixed with narrative. In a Markdown file, notation goes in code blocks and prose sits between them:

````
### S7 *Dark alley behind tavern,
Midnight*

```
@ Sneak past the guards
d: Stealth d6=2 vs TN 4 -> Fail
=> My foot kicks a barrel. [Clock:Alert
2/6]

? Do they see me?
-> No, but... (d6=3)
=> Distracted, but one guard lingers.
[N:Guard|watchful]

N (Guard): "Who's there?"
PC: "Stay calm... just stay calm."
```

The guard's torch light sweeps across the
````

```
alley walls. I press myself
into the shadows, barely breathing.
```

In a notebook, the same scene is written straight through, with no fences: the notation lines are identical, and the prose simply follows them.

6.3 Complete Campaign Log (Digital)

````
---
title: Clearview Mystery
ruleset: Loner + Mythic Oracle
genre: Teen mystery / supernatural
player: Roberto
pcs: Alex [PC:Alex|HP 8|Stress 0]
start_date: 2025-09-03
last_update: 2025-10-28
---

# Clearview Mystery

## Session 1
*Date: 2025-09-03 | Duration: 1h30*

### S1 *School library after hours*

```
@ Sneak inside to check the archives
d: Stealth d6=5 vs TN 4 -> Success
=> I slip inside unnoticed. [L:Library|
````
````

````
dark|quiet]

? Is there a strange clue waiting?
-> Yes (d6=6)
=> I find a torn diary page about the
lighthouse. [Track:Lighthouse Clues 1/6]
```

The page is yellowed with age. The
handwriting is shaky: "The light
calls to us. We must not answer."

```
[Thread:Lighthouse Mystery|Open]
```

### S2 *Outside the library, empty hall*

```
? Do I hear footsteps?
-> Yes, but... (d6=4)
=> A janitor approaches, but he doesn't
notice me yet. [N:Janitor|tired|
suspicious]

N (Janitor): "Thought I heard
something..."
PC (Alex, whisper): "Gotta get out of
here."
```
````
````

I freeze. His keys jangle as he walks past the doorway.

```
@ Slip out while he's distracted
d: Stealth d6=6 vs TN 4 -> Success
=> I escape into the night safely.
```

Session 2
Date: 2025-09-10 | Duration: 2h

Recap: Found diary page hinting at lighthouse. Nearly spotted in library.

S3 *Path to the old lighthouse, Day 2*

```
@ Approach quietly at dusk
d: Stealth d6=2 vs TN 4 -> Fail
=> I step on broken glass, crunching loudly. [Clock:Suspicion 1/6]

? Does anyone respond from inside?
-> No, but... (d6=3)
=> The light flickers briefly in the tower window. [L:Lighthouse|ruined|haunted]
```

````
### S4 *Inside lighthouse foyer*

```
@ Search the floor for signs of activity
d: Investigation d6=6 vs TN 4 -> Success
=> I find fresh footprints in the dust.
[Thread:Who is using the lighthouse?|
Open]

? What's lying near the stairs?
tbl: Lighthouse Debris d100=42 -> A
broken lantern
=> A cracked lantern, still warm.
[Track:Lighthouse Clues 2/6]

PC (Alex, thinking): "This place isn't as
abandoned as everyone thinks..."
```

Someone's been here. Recently.
````

6.4 Complete Campaign Log (Analog)

The same campaign in a notebook. Only the headers change; every notation line is written exactly as in 6.3.

```
=== Campaign Log: Clearview Mystery ===
[Title]        Clearview Mystery
[Ruleset]      Loner + Mythic Oracle
```

```
[Genre]        Teen mystery /
supernatural
[Player]       Roberto
[PCs]          Alex [PC:Alex|HP 8|Stress
0]
[Start Date]   2025-09-03
[Last Update]  2025-10-28

=== Session 1 ===
[Date]        2025-09-03
[Duration]    1h30

S1 *School library after hours*

@ Sneak inside to check the archives
d: Stealth d6=5 vs TN 4 -> Success
=> I slip inside unnoticed. [L:Library|
dark|quiet]

? Is there a strange clue waiting?
-> Yes (d6=6)
=> I find a torn diary page about the
lighthouse. [Track:Lighthouse Clues 1/6]

The page is yellowed. Shaky writing: "The
light calls to us."

[Thread:Lighthouse Mystery|Open]
```

7. Best Practices

You've learned the notation—now let's talk about using it well. This section shows proven patterns that make your logs clearer and more useful, plus common mistakes to avoid.

These are guidelines drawn from the solo community's collective experience. They're not rigid rules, but they'll help you create logs that are easy to read, reference, and share.

7.1 Good Practices ✓

These patterns make your logs cleaner, more searchable, and easier to reference later. You don't need to follow all of them, but they represent what works well for most solo players.

Do: Keep actions and rolls connected

```
@ Pick the lock
d: d20=15 vs DC 14 -> Success
=> The door swings open silently.
```

Do: Use tags for persistent elements

```
[N:Jonah|friendly|wounded]
[L:Lighthouse|ruined]
```

Do: Record consequences clearly

```

```

```
=> I find the key. [Track:Clues 2/4]
=> But the guard heard me. [Clock:Alert
1/6]
```

Do: Use reference tags in later scenes

Establish the element at its first mention, then refer back to it:

```
[N:Jonah|friendly]
...
[#N:Jonah] approaches cautiously
```

Do: Mix shorthand and narrative as needed

```
@ Sneak past guard
d: 5≥4 -> Success
=> I slip by unnoticed, heart pounding.
```

7.2 Bad Practices ✗

These are common pitfalls that make logs harder to read or parse. If you catch yourself doing these, don't worry—just adjust for next time. We've all been there!

Don't: Bury mechanics in prose

```
❌ I tried to pick the lock and rolled a
15 which beat the DC so I opened it
```

```
✔ @ Pick the lock
  d: 15≥14 -> Success
  => The door opens quietly.
```

Don't: Forget to record consequences

```
❌ @ Attack the guard
  d: 8<10 -> Fail

✔ @ Attack the guard
  d: 8<10 -> Fail
  => My blade glances off his armor. He
counterattacks!
```

Don't: Lose track of tags across scenes

The first version leaves the reader asking how and when the guard changed:

```
❌ [N:Guard|alert] ... then later ...
[N:Guard|sleeping]

✔ [N:Guard|alert] ... then later ...
  @ Knock him out
  d: 6≥5 -> S
  => [N:Guard|alert>unconscious]
```

Don't: Mix action and oracle symbols

Sneaking is something you do, so it's an action. Whether they notice is something the world decides, so it's a question:

```
❌ ? Sneak past guards

✔ @ Sneak past guards
  ? Do they notice?
```

Don't: Forget scene context

```
❌ S7
  @ Sneak past guards

✔ S7 *Dark alley, midnight*
  @ Sneak past guards
```

8. Templates

Starting from a blank page can be daunting. These templates give you a structured starting point—copy them, fill in the blanks, and start playing.

Each template comes in both **digital markdown** and **analog notebook** formats. Choose whichever matches your play style, or use them as inspiration to create your own.

Don't treat these as rigid forms. They're scaffolding. Once you're comfortable with the notation, you'll probably develop your own templates that fit your specific needs better.

8.1 Campaign Template (Digital YAML)

For digital markdown files, use YAML front matter to store campaign metadata. This goes at the very top of your file, before any other content.

Copy this template, fill in your details, and you're ready to start your first session.

```
---
title:
ruleset:
genre:
player:
pcs:
start_date:
last_update:
tools:
themes:
tone:
notes:
---

# [Campaign Title]

## Session 1
*Date: | Duration: *

### S1 *Starting scene*

```
Your play log here...
```
```

8.2 Campaign Template (Analog)

For paper notebooks, write this header block at the start of your campaign log. Keep it simple—you can always add more details later if needed.

```
=== Campaign Log: [Title] ===
[Title]
[Ruleset]
[Genre]
[Player]
[PCs]
[Start Date]
[Last Update]
[Tools]
[Themes]
[Tone]
[Notes]

=== Session 1 ===
[Date]
[Duration]

S1 *Starting scene*

Your play log here...
```

8.3 Session Template

Use this at the start of each play session to mark boundaries and provide context. The digital version uses markdown headings; the analog version uses written headers.

Fill in what's useful and skip what's not. The only essential field is the date—everything else is optional.

Digital:

```
## Session X
*Date: | Duration: | Scenes: *

**Recap:**

**Goals:**

### S1 *Scene description*
```

Analog:

```
=== Session X ===
[Date]
[Duration]
[Recap]
[Goals]

S1 *Scene description*
```

8.4 Quick Scene Template

This is your workhorse template—the basic structure you'll use scene after scene. It's intentionally minimal: just enough structure to keep you oriented without slowing you down.

Use this as your default starting point for every scene, whether you're playing digitally or analog.

````
### S# *Location, time*

```
@ Your action
d: your roll -> outcome
=> What happens

? Your question
-> Oracle answer
=> What it means
```
````

9. Adapting to Your System

This notation works with *any* solo RPG system. *Ironsworn*, *Mythic GME*, *Thousand Year Old Vampire*, your own homebrew—doesn't matter. The core symbols stay the same; only the resolution details change.

This section shows you how to adapt the `d:` roll notation and `->` oracle formats to match your specific game system. We'll cover common systems (PbtA, FitD, Ironsworn, OSR) and oracles (Mythic, CRGE, MUNE), but the principles work for anything.

The key insight: The notation separates *mechanics* from *fiction*. Your system determines how mechanics work; the notation just records them consistently.

9.1 System-Specific Roll Notation

The `d:` notation works with any system—you just need to adapt it to your specific dice mechanics. Here's how the notation looks across popular solo RPG systems.

These examples show the pattern: record what you rolled, compare it to what you needed, note the outcome. The details change by system, but the structure stays the same.

9.1.1 Powered by the Apocalypse (PbtA)

```
d: 2d6=9 -> Strong Hit (10+)
d: 2d6=7 -> Weak Hit (7-9)
d: 2d6=4 -> Miss (6-)
```

9.1.2 Forged in the Dark (FitD)

The pool is listed die by die, and the highest die decides:

```
d: Prowl 4d6=6,6,4,2 -> Critical (two 6s)
d: Prowl 3d6=6,5,2 -> Success (6)
d: Prowl 3d6=4,4,2 -> Partial Success
(4-5)
d: Prowl 2d6=3,2 -> Failure (1-3)
```

9.1.3 Ironsworn

The action die plus your stat, against two challenge dice:

```

```

```
d: Face Danger d6+2=6 vs 4,8 -> Weak Hit
d: Strike d6+3=9 vs 2,7 -> Strong Hit
```

9.1.4 Fate/Fudge

Four Fate dice plus your skill, against the opposition:

```
d: Overcome 4dF=+1 (++0-) +3=4 vs 2 ->
Success
d: Overcome 4dF=-3 (-0--) +2=-1 vs 1 ->
Fail
```

9.1.5 OSR/Traditional D&D

```
d: Attack d20+2=17 vs AC 16 -> Hit
d: Save d20-1=7 vs 10 -> Fail
```

9.2 Oracle Adaptations

Different oracle systems have different output formats. Some give yes/no answers, others generate complex results. Here's how to record results from popular oracle systems.

The rule from §3.2.2 applies: an oracle that gives a verdict is recorded with `d:` or `->`, and an oracle that gives content is recorded with `tbl:`. Then capture whatever information your oracle provides.

9.2.1 Mythic GME

Fate questions give a verdict. Mythic's answers are Yes, No, Exceptional Yes and Exceptional No; record the Chaos Factor if you want it on file:

```
? Does the guard notice me? (Unlikely)
d: d100=83 vs 25 (CF 5) -> No
=> He doesn't see me.

? Is the door locked? (50/50)
-> Exceptional Yes (d100=4)
=> Locked, barred, and the bar is
chained.
```

A random event gives content:

```
tbl: Mythic Event
  Focus: d100=78 -> NPC Action
  Meaning: d100=34 -> Betray
```

9.2.2 CRGE

```
? What is the merchant's mood?
tbl: CRGE -> Surge: Actor + Focus / Angry
+ Betrayal
=> The merchant is furious about being
cheated.
```

9.2.3 MUNE

For a *likely* question, roll two dice and keep the higher:

```
? Is anyone home? (Likely)
d: 2d6=2,4 keep highest -> Yes
=> Lights are on, someone's definitely
inside.
```

9.2.4 UNE (Universal NPC Emulator)

```
tbl: UNE Motivation -> Power + Reputation
=> [N:Baron|ambitious|seeks recognition]
```

9.3 Handling Edge Cases

Every system has quirks. Here's how to handle common situations that don't fit the basic notation patterns.

9.3.1 Multiple Rolls in One Action

When you need to make multiple rolls for one action:

Advantage/Disadvantage:

```
@ Attack with advantage
d: Attack 2d20=15,8 keep highest=15 vs AC
12 -> Hit
=> I strike true, blade finding a gap in
the armor.
```

Several rolls for one action: one `d:` line per roll, and the consequence after the last one.

```
@ Perform complex ritual
d: INT d6=4 vs TN 4 -> Success
d: WILL d6=5 vs TN 4 -> Success
=> The spell takes hold, energy crackling
between my fingers.
```

Contested rolls:

```
@ Arm wrestle the sailor
d: STR d20=12 vs sailor d20=15 -> Fail
=> His grip tightens. My arm slams to the
table.
```

9.3.2 Unhelpful Oracle Results

Sometimes an answer is hard to read. Record how you read it with an `interp:` note:

```
? Is the merchant trustworthy?
-> Yes, but... (d6=4)
(interp: trustworthy, but hiding
something)
=> He seems honest, but keeps glancing at
the door nervously.
```

Sometimes an answer sparks nothing. Ask a sharper question, and leave the first one in the log so the reader can follow your thinking:

```
? What's behind the door?
tbl: Room Contents d6=3 -> Nothing of
note
(note: doesn't fit the scene, asking
something sharper)
? Is there danger behind the door?
-> Yes, and... (d6=6)
=> Danger, and it's immediate!
```

The oracle serves your story, not the other way around.

10. Add-ons

The five core symbols — `@`, `?`, `d:`, `->`, `=>` — cover the vast majority of solo play. But some games go deeper in specific directions: tactical combat with initiative and damage tracking, dungeon crawling with room states and light management, resource systems where every torch matters. These needs are real, but they're not universal.

That's what add-ons are for.

10.1 What Add-ons Are

An add-on is a **standalone extension document** that deepens Lonelog notation for a specific type of play. Each add-on:

- Works with the five core symbols — it extends them, never replaces them
- Introduces conventions (tags, formats, structural blocks) suited to its domain
- Functions independently — you read just the add-on you need, not the whole ecosystem
- Integrates cleanly with other add-ons if you use more than one

Add-ons live in separate files rather than in this document. That's a deliberate choice: a dungeon crawler who never fights in initiative order shouldn't have to scroll past four pages of combat rules. Lonelog's core should stay lean. Add-ons let the system grow without bloating the manual you carry to the table.

Think of the core as a language, and add-ons as domain vocabularies. A linguist and a sailor both speak English, but the sailor has words for things the linguist doesn't need. The words don't conflict — they extend.

10.2 Why Separate Files

Three reasons:

Download only what you need. On itch.io, in a Markdown vault, or printed and tucked into a notebook — you grab the add-ons that match your current campaign. Running Ironsworn? Grab the resource tracking add-on. Running a dungeon crawl? Add the dungeon add-on. Nothing you don't use.

Update independently. If the Combat Add-on refines its initiative notation, that update doesn't touch the core spec. Core and add-ons can evolve at their own pace, stay in sync where they need to, and diverge where they legitimately differ.

Share and remix freely. The community can write, publish, and share add-ons without modifying the core document. A player who develops a brilliant notation for hex crawling can release it as a Lonelog add-on. The shared core ensures it's immediately readable to anyone who knows Lonelog.

10.3 How to Use Add-ons

Start with the core. If you're new to Lonelog, spend at least a session or two with just the five core symbols before layering anything on top. The core handles more than you might expect.

Add one at a time. If you're adding a dungeon crawl notation and a resource tracking system in the same campaign, introduce them one session apart. That gives you time to settle each one before combining them.

Mix and match freely. Add-ons are designed to coexist. The Combat Add-on and the Dungeon Crawling Add-on, for example, are written to work in the same session log without symbol conflicts.

When in doubt, skip it. If an add-on feels like overhead rather than help, don't use it. The core notation is always sufficient. Add-ons serve your play; your play doesn't serve add-ons.

10.4 Available Add-ons

The following add-ons are part of the official Lonelog ecosystem:

Add-on	Best For
Combat Add-on	Tactical fights, initiative, HP tracking
Dungeon Crawling Add-on	Room exploration, light, traps, mapping notes
Resource Tracking Add-on	Inventory, usage dice, wealth, supply
Dice Notation Add-on	A full grammar for dice expressions: keep/drop, exploding dice, pools
Card Notation Add-on	Playing cards, tarot and decks as oracles or mechanics
Solo Wargaming Add-on	Units, battles and campaigns

Alongside the add-ons, three **companion notations** share Lonelog's symbols for neighbouring jobs:

Notation	For
Partylog	Group play with a GM, or several players at one table;

Notation	For
	adds ! for GM and world events
Lorelog (draft)	Worldbuilding logs: history, factions and places built before or between sessions
Maplog (draft)	Drawing maps during play; the state of things stays in the Lonelog log

Community-created add-ons follow the same conventions. See the **Community Add-on Guidelines** for how to write one, and the Lonelog itch.io page for the community library.

10.5 A Note for Add-on Authors

If you're writing a Lonelog add-on — for your own use, to share with friends, or to publish — the **Community Add-on Guidelines** and **Add-on Template** are your starting point. They cover the design constraints that keep add-ons compatible with core, the required metadata format, and how to structure examples so they read naturally alongside core Lonelog logs.

The guiding principle: **extend, don't replace.** A Lonelog add-on that invents its own action symbol isn't a Lonelog add-on — it's a fork. The power of the ecosystem comes from shared conventions at the core, with creativity in the extensions.

Appendices

A. Notation Legend

This is the cheat sheet to keep handy while you play. Everything here is explained in detail earlier; this is the condensed version for fast lookup.

A.1 Core Symbols

Symbol	Meaning	Example
`@`	Player action — primary PC, or the storyline's POV character	`@ Pick the lock`
`@(Name)`	Action by a named actor (other PC, companion, NPC)	`@(Jonah) Covers the door`
`@(A > B)`	Lead/Assist — A helps B, B makes the roll	`@(Elara > Jonah) Boosts the climb`
`@(A+B)`	Joint action — both contribute to one roll	`@(Elara+Jonah) Force the door`
`?`	Oracle question, with optional likelihood	`? Is anyone inside? (Likely)`
`d:`	Roll or rule that gives a **verdict**	`d: Stealth d6=5 vs TN 4 -> Success`

Symbol	Meaning	Example
`tbl:`	Random result that gives **content**	`tbl: Loot d100=42 -> A broken sword`
`->`	Outcome: a roll's verdict or an oracle's answer	`-> Yes, but... (d6=4)`
`=>`	Consequence	`=> The door opens quietly`

Standard form puts each element on its own line. Compact form chains them on one line, each symbol preceded by a space.

A.2 Rolls and Answers

The `d:` line: `d: [Label] <dice>[+mod]=<total> [vs <target>] -> <outcome>`

Notation	Meaning
`≥` or `>=`	Meets or beats the target
`<`	Falls short of the target
`vs`	Neutral "against"
`-> S` / `-> F`	Success / Fail
`+[…]` / `-[…]` before the dice	Roll context: what helps, what hinders

Oracle answers: full `d: d100=60 vs 35 -> No`, compact `-> No (d6=2)`, or bare `-> No`.

Code	Meaning	Code	Meaning
`Y`	Yes	`N`	No
`Ya`	Yes, and…	`Na`	No, and…

Code	Meaning	Code	Meaning
`Yb`	Yes, but…	`Nb`	No, but…
`Yx`	Exceptional yes	`Nx`	Exceptional no

A.3 Tags

- `[PC:Name|values]` — Player character
- `[N:Name|values]` — NPC
- `[L:Name|values]` — Location
- `[Thread:Name|state]` — Story thread (`Open`, `Closed`, `Abandoned`, …)
- `[E:Name]` — A notable event that happened
- `[#N:Name]` — Reference to an element defined earlier (any type)
- `[N:Name|cat:v,v|cat:v]` — Values grouped in categories

Updates (tags accumulate):

Write	Effect
`value`	Adds a value
`-value`	Removes a value
`old>new` (or `old→new`)	Replaces a value
`category:value`	Replaces what the category held
`category:+v` / `category:-v`	Adds or removes one entry in a category
`HP 6`	Sets a number
`HP-2` / `HP+3`	Adjusts a number
`×12` / `×-1`	Sets / adjusts a count

Multi-line form: open with `[Type:Name`, put each `| value` on its own indented line, close with `]` on its own line.

A.4 Progress Tracking

Tag	Meaning
`[Clock:Name X/Y]`	Fills up toward a threat
`[Track:Name X/Y]`	Fills up toward a goal
`[Timer:Name X]`	Counts down to zero
`[Clock:Name+1]` / `[Timer:Name-1]`	Advances a counter

A.5 Random Tables

Notation	Meaning
`tbl: Name (d6)` + indented `1: …` lines	Defines a numbered table
`tbl: Name [A, B, C]`	Defines a list of options
`tbl: Name (d6) [A, B, …]`	A list numbered 1… in order
`tbl: Name d6=3 -> Result`	A result (a `tbl:` line with `->`)
`tbl: Name` + indented `Axis: roll -> result` lines	A multi-part result

A.6 Structure

Notation	Meaning
`S#`	Scene
`S#a`, `S#b`	Flashback branching from scene `#`
`S#.1`, `S#.2`	Montage beats within scene `#`

Notation	Meaning
`T#-S#`	Scene in parallel storyline `T#`
`T#-S# *context* @(Name)`	Storyline's first scene, declaring its POV character

A.7 Narrative

Notation	Meaning
`=> Prose`	Short inline description
`N (Name): "Speech"`	NPC dialogue
`PC (Name): "Speech"`	PC dialogue
`\---` … `---\`	Long block: documents, extended description

A.8 Meta

`(label: text)` — an out-of-character note. Suggested labels: `note:`, `house:`, `ruling:`, `interp:`, `retcon:`, `todo:`, `question:`, `reflection:`, `source:`, `break:`. Any other label works too.

A.9 Complete Example Line

```
S3 @Pick lock d:15≥14 ->S => door opens
quietly [N:Guard|alert]
```

B. FAQ

Got questions? You're not alone. These are the most common questions from people learning the notation, along with straight answers.

If your question isn't here, remember: the notation is flexible. If you're wondering whether you can do something differently, the answer is probably "yes, if it works for you."

Q: Do I need to use every element?
A: No! Start with just `@`, `?`, `d:`, `->`, and `=>`. Add other elements only if they help you.

Q: Can I use this with traditional RPGs (with a GM)?
A: The core notation works great for any RPG notes. The oracle elements (`?`, `->`) are specifically for solo play, but the action/resolution notation works everywhere.

Q: What if my system doesn't use dice?
A: Use `d:` for any resolution mechanic: `d: Draw from deck -> Queen of Spades`, `d: Spend token -> Success`

Q: Should I use digital or analog format?
A: Whichever you prefer! They use the same notation. Digital has better search/organization; analog is immediate and tactile.

Q: How detailed should my notes be?
A: As detailed as you want! The system works for pure shorthand (Example 6.1) or rich narrative (Example 6.3).

Q: Can I share my logs with others?
A: Yes! That's one reason for standardized notation. Others familiar with the system can read your logs easily.

Q: What about house rules or custom symbols?
A: Document them in meta notes: `(house: using + for advantage, - for disadvantage)`. The system is designed to be extended; for conventions you'll reuse, consider writing an add-on (§10).

Q: Do scene numbers have to be sequential?
A: No. Use `S1`, `S2`, `S3` for simplicity, but branch (`S3a`, `S3b`) or use storyline prefixes (`T1-S1`) if helpful.

Q: Should I update tags every time something changes?
A: Show significant changes explicitly: `[N:Guard|alert>unconscious]`. Minor changes can be implied through narrative.

Q: If I post a recorded session log publicly, does it need to be under the ShareAlike license?
A: No. The CC BY-SA 4.0 license covers the Lonelog specification document, not content created using the notation. Your session logs are your own independent creative work — publish and license them however you choose. The ShareAlike clause would only apply if you were adapting or redistributing the spec itself, for example by forking Lonelog into a new notation system.

Q: I made my own random table. How do I include it in my log?
A: Define it inline with `tbl: Name (die)` followed by indented entries, or use `tbl: Name [Option A, Option B, ...]` for unnumbered option sets. See §4.3.1 and §4.3.2. This makes your log self-contained — readers see the full table and the result without needing external references.

Q: When do I use `d:` and when `tbl:`?
A: Ask what came back. A verdict (success, failure, yes, no) is `d:`. Content (a thing, a name, an event) is `tbl:`. The number of dice doesn't matter.

Q: I made a mistake in an earlier entry. Should I fix it?
A: Leave the original line and add a `(retcon: …)` note after it (§4.5). The log then shows both what happened and what you corrected.

Q: My old logs use forms this version no longer shows. Are they still valid?

A: Yes. Lonelog never invalidates an old log. Deprecated forms stay readable; Appendix E lists them with their current equivalents.

C. Symbol Design Philosophy

Lonelog's symbols were chosen for specific reasons:

- **`@` (Action)**: Represents "at this point" or the actor taking action. Changed from `>` in v2.0 to avoid conflict with Markdown blockquotes.
- **`?` (Question)**: Universal symbol for inquiry. Unchanged from v2.0.
- **`d:` (Dice/Resolution)**: Clear abbreviation for dice rolls. Unchanged from v2.0. Since v1.7, `d:` is defined by what it gives back, a verdict, rather than by the use of dice.
- **`tbl:` (Table)**: Random content from a table or generator. Since v1.7 it also covers what used to be `gen:`, because the number of rolls behind a result doesn't change what the reader needs to know.
- **`->` (Resolution)**: Retained from v2.0. Now unified for ALL resolutions (dice and oracle). The arrow visually shows "this leads to outcome."
- **`=>` (Consequence)**: Retained from v2.0. Double arrow shows cascading effects. Clarified usage: consequences only (v2.0 overloaded this for dice outcomes too).

Markdown Compatibility: All symbols work cleanly in code blocks and don't conflict with markdown formatting or mathematical operators. Always wrap notation in code blocks when using digital markdown to prevent conflicts with Markdown extensions.

D. Specification Summary

This appendix restates the rules of Lonelog in compact, normative form, for people writing tools, parsers or add-ons. The rest of the

document explains them; this appendix is the reference. The key words MUST, SHOULD and MAY mean what they mean in technical standards: a requirement, a recommendation and an option.

D.1 Lines

1. A log is a sequence of lines. In digital Markdown files, notation lines SHOULD sit inside fenced code blocks.
2. A **core element** is a line, or part of a line, introduced by one of: `@`, `@(…)`, `?`, `d:`, `tbl:`, `->`, `=>`.
3. In **standard form**, each core element starts a line. In **compact form**, several elements MAY share a line; each symbol MUST then be preceded by whitespace, and the whitespace after it MAY be omitted.
4. `->` marks an outcome and nothing else: it appears after `d:` or `tbl:`, under a `tbl:` line, or at the start of an answer. It MUST NOT be used as an arrow in prose, tags or action text; inside a tag, a change is written `old>new`.
5. Indentation carries no meaning, except under a `tbl:` line, where indented lines are that table's entries or result axes.
6. Any other line is prose, dialogue, a tag line, a scene or session header, or a meta note.

D.2 Actions and questions

1. `@` declares an action by the primary PC, or by the current storyline's POV character when one is declared (D.6).
2. `@(Name)` attributes the action to a named actor. `@(A > B)` means A assists and B rolls. `@(A+B)` means a joint roll.
3. `?` asks a question of the oracle. A likelihood MAY follow in parentheses at the end of the question.

D.3 Resolutions

1. `d:` records a roll or rule that gives a **verdict**: success or failure, yes or no, or a degree of either. Its outcome MUST follow `->` on

the same line, unless the total is itself the result (for example, a generated stat).

2. A `d:` line SHOULD follow the order `[Label] <dice>[+mod]=<total> [vs <target>] -> <outcome>`. Every part except `d:` and the outcome MAY be omitted. Dice expressions follow the Dice Notation Add-on where more than `NdS±N` is needed; individual die results MAY follow the expression in brackets, before `=`.
3. Roll context MAY appear before the dice expression as `+[…]` (what helps) and `-[…]` (what hinders). Brackets after the dice expression are reserved for individual die results.
4. Comparison operators MUST state a true comparison: `≥`/`>=` for meeting or beating a target, `<` for falling short. `vs` is neutral.
5. `tbl:` records a random result that gives **content**. A `tbl:` line with `->` is a result; a `tbl:` line without `->` is a table definition. The roll on a `tbl:` result MAY be omitted.
6. `->` on its own line records an oracle answer. A parenthesis directly after the answer, without a label, is a roll reference.
7. The answer codes `Y`, `N`, `Ya`, `Yb`, `Na`, `Nb`, `Yx`, `Nx` MAY replace the written answer.

D.4 Consequences

1. `=>` records a consequence. Several `=>` lines MAY follow one resolution.

D.5 Tags

1. A tag has the form `[Type:Name|value|value…]`. `Name` MUST be present for every type.
2. A value MAY be grouped as `category:v,v`.
3. A tag MAY span several lines: `[Type:Name` on the first line, one `| value` per following line, and `]` on its own line.
4. Tags **accumulate**: a later tag for the same element adds its values to those already known. `-value` removes a value. `old>new` (or

`old→new`) replaces one. A value written as `category:value` replaces whatever that category held; `category:+v` and `category:-v` add or remove one entry in it. A number written as `Stat  N` sets it; `Stat+N` and `Stat-N` adjust it. A count is written `×N`, is set by a new `×N` and adjusted by `×+N` or `×-N`. Counts MUST NOT be part of the name.

5. `[#Type:Name]` refers to an element defined earlier and changes nothing.
6. Core types are `PC`, `N`, `L`, `Thread`, `E`, `Clock`, `Track` and `Timer`. `Clock` and `Track` take `X/Y` after the name; `Timer` takes `X`; `E` takes no counter. A progress counter is set by restating it and adjusted with `+N` or `-N` after the name (`[Clock:Alert+1]`). A thread holds one state, changed with `old>new`. Add-ons MAY define further types.
7. Values are identified by what they are, not by their position: a labelled number (`HP  6`), a count (`×12`), a category (`status:cleared`) or a plain value. Add-on tags SHOULD NOT depend on the order of their values.
8. A tag's name identifies one element. For tags that track something owned, such as a purse or a party's stores, the name is the owner: `[Wealth:Kael|Gold 45]`.

D.6 Structure

1. A scene header is a scene identifier, optionally followed by context in *…*. Identifiers: `S#` (scene), `S#a` (flashback from scene `#`), `S#.#` (montage beat), `T#-S#` (scene in parallel storyline `T#`).
2. In digital files, the campaign title is a level-1 heading, sessions are level-2 headings and scenes are level-3 headings. Campaign metadata SHOULD be YAML front matter.
3. A storyline's first scene header MAY end with `@(Name)` to declare its POV character.

D.7 Meta notes

1. A meta note is `(label: text)`. A bare `(text)` is also a meta note but SHOULD carry a label.
2. Labels are free. The suggested set is in §4.5.
3. A correction MUST NOT replace the original entry; it SHOULD be recorded as a `(retcon: …)` note after it.

D.8 Compatibility

1. A log valid under any earlier 1.x version remains valid. Deprecated forms (Appendix E.3) MUST be read with their current meaning.
2. Add-ons MUST NOT redefine the core symbols in D.1.

E. History and Migration

E.1 Why "Lonelog"?

This system started life as **Solo TTRPG Notation**, a name that was descriptive but unwieldy. Nearly 5,000 downloads later, it was clear the concept resonated with the community. But real-world use brought valuable lessons about what worked, what caused friction, and where the notation could evolve.

The rename to **Lonelog** reflects three insights:

- **A name that sticks.** "Solo TTRPG Notation" got abbreviated a dozen different ways. *Lonelog* is compact and evocative: *Lone* (solo play) + *log* (session record).
- **A name you can find.** Search "solo ttrpg notation" and you'll drown in generic results. Search "lonelog" and you get *this system*. **Markdown** succeeded as both a format and a brand; it isn't called "Text Formatting Notation."
- **A name built to last.** As the system matures, a clear identity makes it easier for the community to share resources, tools, and session logs under one banner.

The core philosophy hasn't changed: separate mechanics from fiction, stay compact at the table, scale from one-shots to long campaigns, and work in both markdown and paper notebooks.

E.2 Migrating from Solo TTRPG Notation v2.0

v2.0 Symbol	Lonelog Symbol	Why the Change
`>`	`@`	Avoids conflict with Markdown blockquotes
`->` (oracle only)	`->` (all resolutions)	Unified for both dice and oracle results
`=>` (overloaded)	`=>` (consequences only)	No longer doubles as dice outcome

In v2.0, `=>` was used for both dice outcomes and consequences. Lonelog uses `->` for all resolutions and keeps `=>` for consequences only.

Your old logs are still readable, and you don't need to convert them. If you want consistency across a campaign, use find and replace in your text editor:

1. Replace `>` at the start of a line with `@`.
2. Where `=>` follows a roll and gives its result (Success, Fail, a hit), replace it with `->`. Leave `=>` where it describes what happens in the story.

E.3 Deprecated Forms

These forms appear in logs written for earlier versions. They remain valid and keep their meaning, but current examples use the form on the right.

Deprecated	Current form	Since
`d: 2≤4 -> Fail` (≤ meaning failure)	`d: 2<4 -> Fail`	v1.7
`d: 5≥4 S` (flag without `->`)	`d: 5≥4 -> S`	v1.7
`d: Label 2d6 [tags] = 8`	`d: Label +[tags] 2d6=8`	v1.7
`[E:Name X/Y]`	`[Clock:Name X/Y]`	v1.7
`gen: Name -> result`	`tbl: Name -> result`	v1.7
`=> result [tbl]`	`tbl: Name -> result`	v1.7
`T1+T2-S#`	`S#` with `(note: storylines merge)`	v1.7
`(reminder: …)`, `(house rule: …)`	`(todo: …)`, `(house: …)`	v1.7

v1.7 also tightened how tags are written and updated. These older tag forms stay readable too:

- `[N:Bandits×6]` — a count inside the name. Now: `[N:Bandits|×6]`, so the element keeps its name as the count changes.
- A one-at-a-time state rewritten as a plain value, like `[N:Guard|alert]` followed by `[N:Guard|unconscious]`. Read the later value as replacing the earlier one. Now: `[N:Guard|status:unconscious]`, or `[N:Guard|alert>unconscious]`.
- A thread restated with a new state, like `[Thread:Escape|Closed]`. Now: `[Thread:Escape|Open>Closed]`.

- Arithmetic on the name, like `[Inv:Torch-1]` or `[Wealth:Gold+15]`. Now the change is a value: `[Inv:Torch|×-1]`, `[Wealth:Kael|Gold+15]`.
- A tag with no name, like `[Wealth:Gold 45]` or `[Party:Gold 150]`. Now the name is the owner: `[Wealth:Kael|Gold 45]`, `[Party:Crew|Gold 150]`.
- Fields that depend on their position, like `[R:3|cleared|library]`. Read each value for what it is. Now: `[R:3|status:cleared|library]`.
- `->` used as an arrow in prose or inside a tag. Now `->` marks outcomes only; inside a tag, a change is `old>new`.

Credits & License

© 2025-2026 Roberto Bisceglie

This notation is inspired by the Valley Standard.

Thanks to:

- matita for the `+/-` method to track changes in tags
- flyincaveman for the suggestion on the use of the `@` symbol for character actions (in the tradition of the early ASCII rpgs)
- r/solorpgplay and r/Solo_Roleplaying for the positive reception of this notation and the useful feedback.
- Enrico Fasoli for playtesting and feedback

Version History:

- v 1.7.0: Consolidation release. Three equivalent forms for oracle answers, with likelihood on the question (§3.2.2). A standard `d:` line and truthful comparison operators: `<` replaces `≤` for failure, outcome abbreviations after `->`, and answer codes for compact logs (§3.2). Roll context moves before the dice as `+[…]/-[…]` (§4.1.9). `d:` gives a verdict, `tbl:` gives content, and `gen:` is folded into `tbl:` (§3.2.2, §4.3). `[E:]` marks events; counters use

`[Clock:]` (§4.1.3). Tags accumulate, with `-tag` and `old>new` for changes, `category:value` for states that replace each other, `×N` for counts, and `+N`/`-N` to advance progress tags (§4.1, §4.2). Parallel storylines with POV declaration (§5.3.3). Suggested meta-note labels and retcons (§4.5). Line position and compact form (§3). New Appendix D (Specification Summary) and Appendix E (History and Migration). Rebuilt Appendix A; corrected system examples in §9; restructured and shortened throughout. No v1.6 log becomes invalid.

- v 1.6.0: Added Lead/Assist `@(A > B)` and Joint Action `@(A+B)` syntax to §3.1.1, upstreamed from Partylog. Updated Appendix A.1.
- v 1.5.0: Updated §3.2.2 — `d:` is now the preferred notation for oracle dice rolls, consistent with mechanics resolution. `->` alone demoted to shorthand. Clarified when to use `d:`, `tbl:`, and `gen:` for oracle resolution.
- v 1.4.1: Added §2.3 Indentation — clarified that indentation is optional, carries no structural meaning, and is encouraged as a readability aid.
- v 1.4.0: Added §3.1.1 Multiple Actors — `@(Name)` convention for multi-PC and companion play, promoted from the Combat Add-on.
- v 1.3.0: Added tag category syntax (§4.1.7), multi-line tag form (§4.1.8), and roll context blocks inside `d:` (§3.2.1).
- v 1.2.0: Added Section 10: Add-ons.
- v 1.1.0: Clarified the use of the license. Added specifications for inline definitions, filtered option sets and multi-line result blocks in section 4.3.
- v 1.0.0: Evolved from Solo TTRPG Notation v2.0 by Roberto Bisceglie

This work is licensed under the **Creative Commons Attribution-ShareAlike 4.0 International License**.

You are free to:

- Share — copy and redistribute the material
- Adapt — remix, transform, and build upon the material

Under these terms:

- Attribution — Give appropriate credit
- ShareAlike — Distribute adaptations under the same license

Happy adventuring, solo players!