Taskwarrior Nautical V3.0 - Systems Manual

1) Philosophy

Life is a highly complex endeavour so the focus should be on managing systems, not just tasks. Taskwarrior has a powerful system and Nautical gives you two complementary engines for describing recurring work:

Chains for period-based recurrence. Give it a duration like `3d` or `28h` - Nautical calculates the next due time from your completion.

Anchors for calendar-position recurrence. Specify patterns like `w:mon,wed` (Mondays and Wednesdays) or `m:2sat` (2nd Saturday) - Nautical walks your local calendar to find exact matches.

Both respect your time, preserve context across iterations, and stop exactly when you tell them to.

---

2) Reading this manual

Start with the installation and then jump to point 9 at the recipes library and write some test tasks to get a feel for the system. When you want to find out more about the work under the hood and the capabilities then browse the other sections. Nautical is made to be reliable, intuitive and easy to use. **Best of luck and Godspeed**.

Installation

Nautical = **nautical_core.py** + **on-add-nautical.py** + **on-modify-nautical.py** + **on-exit-nautical.py**.

1. Put the hooks in place:

- `on-add-nautical.py` → `~/.task/hooks/`
- `on-modify-nautical.py` → `~/.task/hooks/`
- `on-exit-nautical.py` → `~/.task/hooks/`
- `nautical_core.py` → `~/.task/` *(single fixed location; override with* `NAUTICAL_CORE_PATH` *for dev/tests)*

Environment and Files

- `TASKDATA` points to the Taskwarrior data directory; Nautical reads/writes queues and locks there.
- `NAUTICAL_CORE_PATH` overrides the core load path (dev/tests); otherwise core is loaded from `~/.task`.
- Hook files create these in `TASKDATA`:
  - `.nautical_spawn_queue.jsonl` and `.nautical_spawn_queue.lock`
  - `.nautical_dead_letter.jsonl` and `.nautical_dead_letter.lock`
  - `.nautical_spawn_queue.lock_failed` (last failure marker only; overwritten each time)
  - `.nautical_spawn_queue.lock_failed.count` (counter for lock contention)

Durable Queue Mode

Set `NAUTICAL_DURABLE_QUEUE=1` to force `fsync` on queue/dead-letter writes and queue staging replaces. This improves crash/power-loss durability but adds IO latency; leave unset for default (faster) behavior.

Spawn Queue + Recovery Model

When a chain link is completed, Nautical queues a spawn intent and the **on-exit** hook imports the child task and then updates the parent's `nextLink`. This is an intentional "outbox" pattern to avoid re-entering Taskwarrior while it holds its datastore lock.

If a crash happens between queuing and import, the intent remains in `.nautical_spawn_queue.jsonl` and will be picked up on the next Taskwarrior command. Dead-letter entries (permanent failures) are recorded in `.nautical_dead_letter.jsonl`.

Break-glass Mode

If a hook is misbehaving or Taskwarrior changes its input format, you can bypass hooks temporarily:

`task rc.hooks=off <command>`

Dead-letter logs live in `TASKDATA` as `.nautical_dead_letter.jsonl` (with optional rotated overflow files).

1. Make them executable: `chmod +x ~/.task/hooks/on-*-nautical.py`.
2. Add the UDAs from headline 4 to your `~/.taskrc`.
3. Open config-nautical.toml and set your preferences.

Here's a clean README-ready "Configuration" section you can drop in.

Configuration

Nautical reads an optional TOML file named `config-nautical.toml` that lives **next to** `nautical_core.py`.

If the file is missing, sensible built-in defaults are used. Changes take effect on the next hook run.

Config precedence is: `config-nautical.toml` overrides built-in defaults.

Example: `config-nautical.toml`

```
# TimeZone and salt for rand

wrand_salt = "nautical|wrand|v2"

tz = "Australia/Sydney"


# Anchor cache & precompute (ACF+timeline hints) — file-based cache

enable_anchor_cache = true # faster on Termux

anchor_cache_dir = "" # default next to core



#
# ChainID is always enabled in v3+.
# Legacy chains without chainID should be backfilled with tools/nautical_backfill_chainid.py

#
# on-modify hook toggles
#

chain_color_per_chain = false
show_timeline_gaps = true
show_analytics = true
analytics_style = "clinical"
analytics_ontime_tol_secs = 14400
verify_import = true
debug_wait_sched = false
panel_mode = "rich"
fast_color = true
spawn_queue_max_bytes = 524288
```

```
max_chain_walk = 500
max_anchor_iterations = 128
max_link_number = 10000
sanitize_uda = false
sanitize_uda_max_len = 1024
max_json_bytes = 10485760
```

Keys

chainID (mandatory in v3+)

- New nautical task (has `anchor` or `cp`) gets `chainID = short(uuid)`.
- Spawned tasks inherit parent's ChainID.
- Modify won't overwrite an existing ChainID or stamp linked tasks.
- Hooks use `task export chainID:<short>` for O(1) chain fetches.
- If your existing chains are missing ChainID, run `tools/nautical_backfill_chainid.py`.

enable_anchor_cache (bool, default: true)

When true, Nautical precomputes and writes **ACF (Anchor Canonical Form) + anchor hints** (e.g., first due, timeline preview) to a cache file for near-instant previews. When false, no cache file is written and ACF (Anchor Canonical Form) is only kept in memory for the session (never stored as a UDA).

anchor_cache_dir (string, default: next to core)

Where to write the cache when `enable_anchor_cache=true`. Path can be relative (to the core's folder) or absolute. Make sure it's writable.

NAUTICAL_DNF_DISK_CACHE (env, default: enabled)

On-add JSONL cache for parsed anchors. Set `NAUTICAL_DNF_DISK_CACHE=0` to disable.

max_anchor_iterations (int, default: 128)

Upper bound on anchor iteration scans before using a fallback date.

max_link_number (int, default: 10000)

Upper bound for auto-incremented chain link numbers.

sanitize_uda (bool, default: false)

When true, sanitize string fields by removing control characters and clamping length.

sanitize_uda_max_len (int, default: 1024)

Maximum length for sanitized string values.

max_json_bytes (int, default: 10485760)

Maximum size accepted from hook stdin.

NAUTICAL_CLEAR_CACHES (env, default: disabled)

When set to `1`, clear in-process LRU caches after anchor parsing.

DST policy

When Nautical converts a local wall-clock time to UTC (e.g., anchor @t times), it applies a deterministic DST policy:

- Ambiguous times (fall back) choose the earlier occurrence.
- Nonexistent times (spring forward) shift forward to the next valid local time.

on-modify toggles (config)

Hook behavior can be tuned via `config-nautical.toml`:

- **chain_color_per_chain** (bool, default: false)
- **show_timeline_gaps** (bool, default: true)
- **show_analytics** (bool, default: true)
- **analytics_style** ("coach" or "clinical", default: clinical)
- **analytics_ontime_tol_secs** (int seconds, default: 14400)
- **verify_import** (bool, default: true)
- **debug_wait_sched** (bool, default: false)
- **panel_mode** ("rich", "fast", or "line", default: rich)
- **fast_color** (bool, default: true)
- **spawn_queue_max_bytes** (int bytes, default: 524288)
- **max_chain_walk** (int, default: 500)

---

3) Notation & conventions

This section is the "grammar" Nautical understands.

3.1 Engines & where they read time from

- **Chains (`cp`)** - Use a **period** (e.g., `3d`, `28h`, `P2W`). Next link is computed from the **completion time** (and may preserve wall-clock; see §3.2).
- **Anchors (`anchor`)** - Use **calendar positions** (weekly / monthly / yearly). Next link is computed by **walking local calendar dates** that match your pattern.

3.2 Dates, times, and time zones

- **Local vs UTC**: All calendar reasoning (anchors) is done on **local dates**; panels display local time. Internally, times persist in UTC.
- **Seed time**: The **first link's `due:` time** becomes the wall-clock reference.
- If your period is a **multiple of 24h** (e.g., `2d`, `1w`), Nautical keeps that wall-clock (due time).
- What if you want a chain period multiple of 24h but want to have an exact add from completion time? The simplest solution is to use "cp:24h+1s"; that extra second is going to cause the engine to respect the completion time.
- For other periods (e.g., `28h`, `33h`) Nautical does an **exact add from end**.
- **Per-term time**: You can attach time to specific anchor terms with `@t=HH:MM` (e.g., `w:mon@t=09:00,fri@t=15:00`). If omitted, Nautical uses the seed link's `due:` time.
- **Date style for yearly anchors**: `y:MM-DD` only (same MM-DD style as Taskwarrior dates).

3.3 Anchor tokens you can use (by family)

Weekly

- **Weekdays**: `mon`, `tue`, `wed`, `thu`, `fri`, `sat`, `sun`.
- **Lists**: `w:mon,fri`.
- **Ranges**: `w:mon..wed` (inclusive).
- **Stepped cadence**: `w/2:mon,tue` (every 2 weeks on Mon & Tue), `w/3:fri` (every 3rd week on Fri).

Monthly – by date

- **Specific days**: `m:1`, `m:15`, `m:31`.
- **Last day**: `m:-1`.
- **Lists**: `m:1,15,-1`.
- **Ranges/buckets**: `m:1..7` (days 1–7), `m:22..28`, etc.
- **Business-day ordinals**: `m:5bd` (5th business day), `m:15bd`.
- **Stepped cadence**: `m/2:-1` (every 2 months on last day), `m/3:1` (every 3 months on the 1st).

Monthly – by weekday position

- **Nth weekday**: `m:1mon` … `m:5sun` (1st–5th).
- **Last weekday**: `m:last-fri`, `m:last-mon`, etc.
- **Lists**: `m:2mon,4thu`.
- **Stepped cadence**: `m/2:2sat` (every 2 months, 2nd Saturday), `m/4:last-fri`.

Yearly

- **Specific dates**: `y:05-20` (or `y:05-20` / `y:20-05` per your setting).
- **Lists**: `y:01-15,04-15,07-15,10-15`.
- **Ranges (inclusive)**: `y:20-01..27-01` (Jan 20–27), `y:20-04..15-05` (Apr 20–May 15).
- **Random month pick**: `y:10-rand` (one random day in October, deterministic per chain).
- **Leap day**: `y:02-29` (appears only in leap years).
- **Stepped cadence**: y/3:06-07 (every 3 years on 7 of June)

3.4 Modifiers (`@…`) and what they do

Attach modifiers to **individual terms** (right after them). Multiple modifiers can be chained; order is not important unless noted.

- **Time of day**: `@t=HH:MM` - sets the time for that term only.
- Example: `w:mon@t=09:00,fri@t=15:00` (Monday at 9, Friday at 15)
- **Business-day rolls** (for date-based terms):
- `@nbd` - roll to **next business day** if the date falls on weekend.
- `@pbd` - roll to **previous business day** before the date if the date falls on weekend.
- `@nw` - **nearest weekday** to that date (Fri if Sat, Mon if Sun).
- `@bd` - **weekdays only** (filters out Sat/Sun for random/bucket picks).
- **Specific weekday rolls** (apply to a date to find the next/previous named weekday):
- `@next-mon`, `@prev-fri`, `@next-sat`, `@prev-mon`, etc.
- For example anchor:"y:12-31@prev-fri" will match the Friday before the end of the year.
- **Scope notes**:
- Rolls are most meaningful for **monthly day** terms (e.g., `m:1@nbd`).
- Weekly terms (`w:mon`) already select weekdays; applying `@nbd` is redundant there.
- Yearly fixed dates pair well with `@t=…`; business-day rolls are typically used through monthly forms.

3.5 Logic & grouping

- **AND**: `+` - both sides must match.

Example: `w:mon + m:1,15` → Mondays that are also the 1st or 15th.

- **OR**: `|` - either side may match.

Example: `m:1sat | m:3fri`.

- **Parentheses**: group sub-expressions.

Example: `(m:1..7 + m:rand@bd) | (m:8..14 + m:rand@bd)`.

- **Precedence**: `+` (AND) binds **tighter** than `|` (OR). Use parentheses to make intent explicit.
- **Whitespace**: optional; ignored inside patterns. When your shell could parse `+` or `|`, **quote** the pattern:

`anchor:'w:mon + m:1,15'`.

3.6 Modes (anchor backfill semantics)

Attach via UDA: `anchor_mode:skip` (default), `all`, or `flex`.

- **skip** - do **not** backfill missed anchors; jump to the next future match. (training or vitamins)
- **all** - **backfill every** missed anchor before moving forward (useful for bills).
- **flex** - skip backlog **once**, then behave like **all** going forward.
- Explanation: This is used when the mode of a task is 'all' and you don't want to backfill the overdues but want to continue doing the task going forward; in that instance you modify the anchor_mode to 'flex' and on the next completion of the task, Nautical is going to jump overdues and go to the next anchor match and change the mode to 'all'. This is a one-off convenience mode.

3.7 Caps (hard stops)

- **By count**: `chainMax:N` - stop at the **N-th** link; panels mark second-to-last and last.
- **By date**: `chainUntil:YYYY-MM-DD[Thh:mm]` - include anchors ≤ **this local moment**; last link is the final anchor on/before that time.
- **Both set**: Nautical stops at **whichever comes first**.
- **Overdue rule**: If a link is overdue, **nothing spawns** until you finish/delete it.

3.7.1 Manual stop of the chain

- Modification of the "chain" UDA to **off**. When this is executed and the task is completed, Nautical isn't going to spawn the next instance.
- **Deletion** of the task.

3.8 Random behavior (deterministic)

- w:rand - one random day per week.
- `m:rand` - one random day each month.
- `m:rand@bd` - one random **weekday** each month.
- `y:MM-rand` - one random day in that month each year.
- `y:rand` - one random day in the year
- check the anchor library for more examples.

3.9 Validation & edge cases

- **Strict parsing**: malformed tokens are rejected (e.g., `m:1,1as5`, non-leap `y:02-29`, out-of-range days).
- **Case-insensitive** tokens (`Mon`, `MON`, `mon` all work); prefer lowercase.
- **Business day definition**: **Mon..Fri**; no holiday calendar... *yet*.
- **Quoting**: Always quote patterns with spaces, `+`, or `|`.

- **One engine per task**: Use either `cp` **or** `anchor` on the same task.

4) UDAs setup

```
#░███░███░█░█░███░███░███░███░█░░
#░█░█░███░█░█░░█░░░█░░█░░░███░█░░
#░█░█░█░█░███░░█░░███░███░█░█░███

## Classic Chain Recurrence

uda.cp.type=duration
uda.cp.label=Chain Period
uda.chain.type=string
uda.chain.label=Chain Status
uda.chain.values=on,off
uda.chain.default=off



## Advanced Anchor Recurrence

uda.anchor.type=string
uda.anchor.label=Anchor
uda.anchor_mode.type=string
uda.anchor_mode.values=flex,all,skip
uda.anchor_mode.default=skip
uda.anchor_mode.label=Anchor Mode



## Limits

uda.chainMax.type=numeric
uda.chainMax.label=Chain Max
uda.chainUntil.type=date
uda.chainUntil.label=Chain Until



## Lineage

uda.prevLink.type=string
uda.prevLink.label=Previous Link
uda.nextLink.type=string
uda.nextLink.label=Next Link
uda.link.type=numeric
uda.link.label=Link Number
uda.chainID.type=string
uda.chainID.label=ChainID

#
```

5) Core behavior

- **One pending link.**
- **Overdue pauses.** If a link is overdue, Nautical waits. Complete or delete it to move on.
- **Copies your context.** Project, tags, UDAs, dependencies, and **annotations (timestamps kept)** travel to the next link.
- **Caps are hard stops.** `chainMax` (by count) / `chainUntil` (by date). Panels mark second-to-last and last link.
- **Drift-free.** Multiples of 24h keep due time; odd spans add exactly.

6) Chains (Classic Chain Recurrence)

6.1 Period syntax you can use

- Short forms: `12d`, `2w`, `28h`, `90m`.
- ISO-8601 works too: `P12D`, `P2W`, `PT28H`.

6.2 Wall-clock vs exact-add

- **Multiples of 24h** (e.g., `2d`, `1w`) → keep the same local time as the seed task's `due:`.
- Use your desired chain period +1s or -1s to use exact add. (Example, cp:3d-1s)
- **Other spans** (e.g., `28h`, `33h`) → next due = **end + cp** (exact add), so odd cadences don't drift.

6.3 Seeding the time

Give the first link a due time; Nautical carries it when appropriate.

```
# Trim the grass every 12 days at 09:00

task add "Trim the grass" due:tomorrow+9h cp:12d



# The next link is going to have a due time of 09:00 regardless if you completed the current one early or late.
```

6.4 Stopping conditions

- **By count**: `chainMax:5` → stop after 5 links.
- **By date**: `chainUntil:2025-12-31` → stop on/at the last link before that date.
- By manually modifying the chain UDA to **off**. (task 112 modify chain:off)
- By deletion of the task.

Panels show **links left** and the **final date**.

```
# 33h cadence, cap by count

task add "Calibration checks" cp:33h chainMax:5 due:today+12h


# Daily at noon, cap by date

task add "Deep work block" cp:1d chainUntil:2025-12-20T12:00 due:today+12h
```

---

7) Anchors (Real-World Patterns)

7.1 Mental model

Anchors point to **calendar positions** (days, weekdays, nth weekdays) and Nautical walks those exact dates. You can combine rules with AND/OR and adjust to business-day reality.

7.2 Weekly anchors

- **Single weekdays**: `mon`, `tue`, `wed`, `thu`, `fri`, `sat`, `sun`.
- **Lists** (comma-separated): `w:mon,fri`.
- **Ranges**: `w:mon..wed` (equivalent to `w:mon,tue,wed`).
- **Shortcuts**: `w:wk` (Mon–Fri), `w:we` (Sat–Sun).
- **Shortcut**: `w:wd` (Mon–Fri).
- **Per-term time**: attach `@t=HH:MM` to **each weekday** if you want **different times per day**, e.g., `w:mon@t=09:00,fri@t=15:00`.
- **Combine** with logic: AND `+` (e.g., `w:mon + m:1,15`), OR `|`.
- **Time tip**: If you omit `@t=`, Nautical uses the **seed task's due time**.

```
# Mon & Fri (skip missed)

task add "Gym sessions" anchor:w:mon,fri anchor_mode:skip due:today


# Range - Mon..Wed

task add "Study block" anchor:w:mon..wed anchor_mode:skip due:today


# Different times per weekday - 09:00 on Mon, 15:00 on Fri

task add "Split training" anchor:w:mon@t=09:00,fri@t=15:00 anchor_mode:skip due:today


# Only Mondays that are also 1st or 15th (AND)

task add "Friends meeting" anchor:'w:mon + m:1,15' due:today


# Either Friday or Sunday (OR)

task add "Weekend wind-down" anchor:'w:fri | w:sun' due:today
```

Tip: You don't have to mention anchor_mode in the task addition if you are happy with the default set by the UDA.

7.3 Monthly anchors - by date

- **Specific days**: `m:1`, `m:15`, `m:31`; **last day**: `m:-1`.
- **Lists**: `m:1,15,-1`.
- **Ranges/buckets**: `m:1..7` (days 1–7), `m:22..28`.
- **Shortcuts**: `m:ld` (last day), `m:lbd` (last business day).
- **Business-day ordinals**: `m:5bd` (5th business day), `m:15bd`.
- **Roll modifiers**: `@nbd` (next business day), `@pbd` (previous BD), `@nw` (nearest weekday), `@bd` (weekdays only), and specific weekday rolls `@next-mon`, `@prev-fri`.
- **Per-term time**: attach `@t=HH:MM` to **individual dates** for **different times per date**, e.g., `m:1@t=09:00,15@t=15:00`.
- **Logic**: AND `+`, OR `|`, and parentheses `( … )`.
- **Time tip**: Without `@t=`, the seed task's `due:` time is used.

```
# 1st and last day (backfill if missed)

task add "Billing sweep" anchor:m:1,-1 anchor_mode:all due:today


# First business day at 09:00

task add "Payroll" anchor:m:1@nbd@t=09:00 anchor_mode:all due:today


# Mid-month on nearest weekday

task add "Mid-month billing" anchor:m:15@nw anchor_mode:skip due:today


# 5th business day

task add "Supplier payments" anchor:m:5bd anchor_mode:all due:today


# Different times per date - 1st at 09:00, 15th at 15:00

task add "Billing windows" anchor:m:1@t=09:00,15@t=15:00 anchor_mode:all due:today


# Buckets (days 1-7) - pair with logic or random

task add "Focus window" anchor:m:1..7 anchor_mode:skip due:today
```

7.4 Monthly anchors - by weekday position

- **Nth weekday**: `m:1mon` … `m:5sun` (1st–5th).
- **Last weekday**: `m:last-fri`, `m:last-mon`, etc.
- **Lists**: `m:2mon,4thu`.
- **Time**: seed via `due:` or add `@t=HH:MM`.
- **Logic**: AND/OR with other monthly or weekly rules.

```
# 2nd Saturday
task add "Sourdough bake day" anchor:m:2sat anchor_mode:skip due:today

# Last Friday of the month
task add "Design session" anchor:m:last-fri anchor_mode:skip due:today

# 1st Wednesday and 3rd Friday
task add "Parent-teacher meeting" anchor:m:1wed,3fri due:today
```

7.5 Yearly anchors

- **Specific dates**: `y:05-20` (or `y:05-20` / `y:20-05` depending on your configured DD-MM vs MM-DD style).
- **Lists**: `y:01-15,04-15,07-15,10-15`.
- **Ranges** (inclusive): `y:01-20..01-27` (Jan 20–27), `y:04-20..05-15` (Apr 20–May 15).
- **Random month pick**: `y:10-rand` (one day in October, deterministic per chain).
- **Quarter aliases**: `y:q1..q4` (quarter window), `y:q1s` (start month), `y:q1m` (mid month), `y:q1e` (end month).
- **Quarter ranges**: `y:q1s..q2s` (start months of Q1–Q2), `y:q1m..q3m` (mid months), `y:q2e..q4e` (end months).
- **Per-term time**: attach `@t=HH:MM` to **individual year-dates** to vary **by month**, e.g., `y:06-01@t=09:00,12-01@t=15:00`.
- **Logic**: AND/OR with other yearly terms.
- **Note**: Business-day rolls mostly apply to monthly day rules; yearly dates pair well with `@t=` for month-specific times.

```
# Anniversary reminder
task add "Anniversary dinner" anchor:y:05-20 anchor_mode:all due:today

# Quarterly review (simple yearly list)
task add "Quarterly review" anchor:y:01-15,04-15,07-15,10-15 anchor_mode:all due:today

# Different times by month - June 1 at 09:00, Dec 1 at 15:00
task add "Seasonal review" anchor:y:06-01@t=09:00,12-01@t=15:00 anchor_mode:skip due:today

# A random day in October each year
task add "Spooky surprise" anchor:y:10-rand anchor_mode:skip due:today

# Leap day (appears only in leap years)
task add "Leap-day check" anchor:y:02-29 anchor_mode:skip due:today

# Quarter-based anchors (advanced)
task add "Quarter start kickoff" anchor:'y:q1s..q4s + m:1' anchor_mode:skip due:today
task add "Quarter end closeout" anchor:'y:q1e..q4e + m:lbd' anchor_mode:all due:today
task add "Mid-quarter check-in" anchor:'y:q1m..q4m + m:15' anchor_mode:skip due:today
```

---

8) Caps with anchors

Caps stop a chain **by count** (`chainMax`) or **by date** (`chainUntil`). They work with both **Chains** and **Anchors**.

How Nautical counts (Anchors)

- `chainMax` → stops after the **N-th link** (the panel marks *second-to-last* and *last*).
- `chainUntil:YYYY-MM-DD[Thh:mm]` → includes anchors **up to and including** that moment in your **local time**. The final link shown is the last anchor ≤ **until**.
- If you set **both**, Nautical stops at the **earliest** of the two limits.

Good to know

- If a link is **overdue**, nothing new spawns until you complete/delete it.
- **Modes** (`skip`/`all`/`flex`) still apply: caps don't override catch-up semantics.
- Panels show **Links left**, **Final (max / until)**, and a timeline that marks *(last link)*.

```
# Weekly anchors (Mon/Fri), stop after 6 links
# → exact final date shown in the completion panel
task add "Bootcamp" anchor:w:mon,fri anchor_mode:skip chainMax:6 due:today

# Business-day monthly, stop by date (end of Q4)
task add "AP run" anchor:m:1@nbd due:today chainUntil:2025-12-31T17:00

# Chain (33h cadence), stop after 5 links
```

```
task add "Stability checks" cp:33h chainMax:5 due:today+12h
```

9) Recipes library

Chains

```
# Grass trim - every 12 days at 09:00

task add "Trim the grass" due:tomorrow+9h cp:12d


# Vitamins - every 28 hours (exact add)

task add "Take the vitamin" due:today+15h cp:28h


# Two-day sprint - stop after 6 links

task add "Focus sprint" cp:2d chainMax:6 due:today+09:00


# Daily deep work until a deadline (keeps 12:00)

task add "Deep work block" cp:1d chainUntil:2025-12-20T12:00 due:today+12h
```

Anchors

Paste this once and replace `<pattern>` with any of the patterns below:

```
task add "Anchor demo" anchor_mode:skip due:today project:nautical.test anchor:'<pattern>'
```

Tip: attach `@t=HH:MM` **on individual terms** when you need **different times per term** (e.g., 09:00 on Monday, 15:00 on Friday).

Weekly anchors

```
w:mon → Mondays

w:mon,fri → Mondays and Fridays

w:mon..wed → Monday through Wednesday

w:wk → Mon–Fri (weekday shortcut)

w:wd → Mon–Fri (weekday shortcut)

w:we → Sat–Sun (weekend shortcut)

"w:wd@t=09:00 | w:we@t=11:00"  → Mon–Fri @ 09:00 and Sat–Sun @ 11:00

w:fri|w:sun → Fridays or Sundays (OR)

w:mon + m:1,15 → Mondays that are also the 1st or 15th (AND)

w:mon@t=09:00,fri@t=15:00 → Mon at 09:00, Fri at 15:00 (per-term times)

w/2:mon,tue → every 2 weeks on Monday & Tuesday

w/3:fri → every 3 weeks on Friday

w:mon..fri@t=09:00,17:30 → Mon–Fri with two times per day
```

Monthly anchors - by date

```
m:1 → 1st day of month

m:-1 → last day of month

m:ld → last day of month (shortcut)

m:1,15,-1 → 1st, 15th, and last day

m:1..7 → bucket: days 1–7 of month

m:5bd → 5th business day of month

m:lbd → last business day of month (shortcut)

m:1@nbd → 1st rolled to next business day (if weekend)

m:1@pbd → previous business day before the 1st

m:15@nw → nearest weekday to the 15th

m:1@t=09:00,15@t=15:00 → 1st at 09:00, 15th at 15:00 (per-term times)

m/2:-1 → every 2 months on the last day

m/3:1 → every 3 months on the 1st

m:1@prev-mon → 1st rolled back to previous Monday

m:1@next-sat → 1st rolled forward to next Saturday
```

```
m:1 + y:01..06,12 → 1st of month except Jul–Nov (via inclusion)
```

Monthly anchors - by weekday position

```
m:2sat → 2nd Saturday of each month

m:last-fri → last Friday of each month

m:1wed,3fri → 1st Wednesday and 3rd Friday


m/2:2sat → every 2 months, 2nd Saturday

m/4:last-fri → every 4 months, last Friday
```

Yearly anchors

```
y:05-20 → May 20th (DD-MM or MM-DD per your setting)

y:01-15,04-15,07-15,10-15 → Quarterly markers

y:04-20..05-15 → Apr 20 – May 15 (inclusive range)

y:10-rand → one random day in October (deterministic per chain)

y:02-29 → Feb 29 (appears only in leap years)

y:06-01@t=09:00,12-01@t=15:00 → June 1 at 09:00; Dec 1 at 15:00 (per-term times)
```

Random gallery

```
w:rand → one random day each ISO week

(w:rand | w:wed) → one random day per week or Wednesday (two days per week)

(w:sat,sun + m:rand) → one random Sat/Sun for each month.

y:q1 → Q1 window (Jan–Mar)

y:q1s → Q1 start month (Jan)

y:q1m → Q1 mid month (Feb)

y:q1e → Q1 end month (Mar)

(m:rand + y:04-20..05-15) → one random day each month and within Apr 20–May 15 each year

m:rand → one random day each month

m:rand@bd → one random **weekday** each month

(m:1..7 + m:rand@bd) → one random weekday chosen from the 1–7 bucket

(m:8..14 + m:rand@bd) → one random weekday from 8–14 (use with OR buckets)

y:10-rand@t=12:00 → one random October date at 12:00

y:rand → one random day in the Year
```

Rolls gallery (quick reference)

```
@nbd → next business day if weekend

@pbd → previous business day before a date

@nw → nearest weekday to a date

@bd → weekdays only (filter)

@next-mon → roll forward to the next Monday

@prev-mon → roll backward to the previous Monday

@next-sat → roll forward to the next Saturday
```

Anchor combinations (use `+` for AND, `|` for OR, parentheses to group)

```
'w:mon + m:1,15' → Mondays that are also the 1st or 15th

'm:1sat | m:3fri' → either 1st Saturday or 3rd Friday

'(m:1..7 + m:rand@bd) | (m:8..14 + m:rand@bd)' → one random BD in days 1–7 OR 8–14

'w:mon..wed | w:fri' → Mon–Wed or Friday

'w:mon@t=09:00 + m:1..21' → Mondays within days 1–21 at 09:00

'm:1@nbd + w:fri' → 1st business day that is also a Friday
```

Modes reminder: `anchor_mode:skip` (skip missed), `all` (backfill all), `flex` (skip backlog once, then be strict).

12) FAQ (short)

Q: How do you change the yearly date format?

A: You don't in v3+. Yearly anchors use MM-DD only.

Q: Can you disable caching of the anchors?

A: Use the config file (config-nautical.toml) for the related toggle.

Q: Can you mix `cp` and `anchor` on the same task?

A: Use one engine per task. You can keep separate tasks for each behavior.

Q: Is random really random?

A: No, it's **deterministic per chain** (seeded by the root). Every task gets a fair, stable draw.

Q: Why didn't the link keep the same time?

A: Only multiples of 24h preserve wall-clock. Odd spans add exactly from **end** (completion time).

Q: You forgot to mark as done a classic recurrence task on the day when you completed it?

A: Use the command 'task ID done end:[date-of-completion]. Nautical is going to calculate the new link from the 'end' variable.

Q: Why not keep the chain as completion time + chain period regardless if it is a multiple of 24h?

A: This way a schedule can be mentained with ease regardless of early/late completions in the day. Small discrapancies will otherwise cause drift. You can't manage a system (effectively) without routines.

Q: Can I backfill missed anchors only once, then go strict?

A: Yes - that's `flex` **mode**.

---

13) Performance tips

- **Keep it simple**: prefer concise patterns over massive OR lists.
- **Termux speed**: Nautical has been designed to run fast on mobile (especially with the file caching enabled).
- **Quote complex patterns** to avoid shell parsing: `anchor:'(m:1..7 + m:rand) | m:last-fri'`.

---

14) Operations & Diagnostics

These flags are useful for troubleshooting, performance tuning, and getting quick anchor feedback.

Self-check and diagnostics

```
python3 nautical_navigator.py --self-check

NAUTICAL_DIAG=1 python3 nautical_navigator.py --self-check
```

Anchor explain / validate

```
python3 nautical_navigator.py --explain "m:last-fri"

python3 nautical_navigator.py --validate "w:mon..fri@t=09:00,17:00"
```

Panel modes and colors

- `panel_mode="fast"` in config forces plain panel rendering (no Rich import).
- `panel_mode="line"` in config shows a single summary line inside a compact panel.
- `fast_color=false` in config disables ANSI colors in fast panels.
- `chain_color_per_chain=true` in config enables color per chain in the on-modify timeline.

Caches and queue controls

- `NAUTICAL_DNF_DISK_CACHE=0` disables the on-add JSONL cache (default: enabled).
- `spawn_queue_max_bytes` in config caps the on-exit spawn intent queue size.
- `max_chain_walk` in config caps how far chain summaries/analytics run.

Load testing

```
python3 tools/load_test_nautical.py --tasks 2000 --concurrency 4

python3 tools/load_test_nautical.py --ramp --ramp-start 200 --ramp-step 500 --ramp-max 10000 --concurrency 16

python3 tools/load_test_nautical.py --rate-ramp --rate-secs 30 --rate-start 5 --rate-step 5 --rate-max 100
```

---

15) Troubleshooting

Nautical is a complex system handling intricate calendar logic, time zone calculations, and state transitions across hundreds of edge cases. While it's been tested rigorously across various scenarios, the surface area for bugs remains significant.

If you encounter unexpected behavior:

- Check your pattern syntax against the reference sections above
- Verify your UDA configuration matches the expected setup
- Review the panel output for clues about what Nautical computed

When something breaks:

Open a GitHub issue with:

- Your pattern or chain configuration

- The unexpected behavior (what you expected vs. what happened)
- Relevant panel output or task details
- Your Taskwarrior version and environment (desktop/Termux/etc.)

The more context you provide, the faster we can isolate and fix the issue.

Repository

---

16) Support the project

I've been working on this project through the darkest time of my life, where I'm still at when I'm typing this.

The only thing kept me going was the promise I've made to my-self to never give up.

If Nautical has improved your workflow and you'd like to support continued development:

Buy me a book

Your support helps sustain the time invested in building, testing, and maintaining systems like this. Every contribution is greatly appreciated.