# Daily Plan

*Anthony Sorace*
*a.9srv.net*

Strand 1 Technologies

*ABSTRACT*

This report describes the current set of plain text files and short scripts I use to track my day's work items and produce a log of past work. The system produces a historical set of daily files, pushes to both *finger* and (indirectly) the web, includes optional carry-forward of unfinished tasks, and uses mostly unstructured text for high flexibility.

The system described can be found at http://txtpunk.com/daily/.

## 1. Introduction

For several years I have found it helpful to create a daily list of tasks I'm hoping to accomplish for the day. This has evolved over time from a simple bullet list to something which adds prose descriptions of the work to be done. In addition to tracking the tasks themselves, the prose helps refresh my thinking about a problem when I've not worked on it for some time (often just a weekend). This has been particularly helpful as my work schedule has become more erratic.

This system consists of a *mkfile* and two support tools to work with a set of plain text files, formatted with one formatting rule and a few conventions, which produce to-do files for "today" and "tomorrow". Over time, the system builds up an archive of these, which can be helpful for many purposes, including simply getting an overview of what was done over a given timeframe (useful for, say, putting together yearly reports or evaluations).

In addition to the standard Plan 9 tools, the system described depends on *datefmt*[1]. It can also run on unix using plan9port.

## 2. The Daily Files

All the content in the system is a series of plain text files with very minimal formatting rules. The files conventionally consist of prose describing the work to be done on that day, followed by checklist items representing specific tasks. The checklist items can be interspersed with the prose or consolidated at the end of the file. The only real formatting rule is that checklist items must begin each line with a □ followed by either a space or tab. This character is used to identify incomplete items for the "carry forward" process (described below). By convention and for symmetry, I change that to ☑ when a task is completed. A representative sample for today:

[1]: http://9p.io/sources/contrib/anothy/src/cmd/datefmt.c

> I got the 'daily' writeup done yesterday, but found I don't seem to have a troff font which has the checkbox characters in it. Let's fix that and get it published.
>
> ☑ Fix font issue in "daily" lab report.
> ☐ Get the "daily" lab report published.

Note that in this example the checkboxes are the first character on the line, followed by a tab character; the other non-blank lines begin with one tab, although that is not required.

### 3. The Workday

I start my workday by running `mk` (which runs the *workday* target) and, most days, `cf` to look for unfinished items from yesterday. The *mkfile* will create the files, if needed, and then use *B* to open them in your editor of choice. The files can be edited and saved as normal. After initially updating today's file, I run `mk plan` (described below) and run it again whenever I wish to update progress (typically at the end of the day and when I finish a set of checkboxes associated with a prose paragraph).

When it looks like I'm not going to get to a task today, or working on part of a task reveals additional work, new tasks get added to the "tomorrow" file[2]. A task item gets added, with prose describing how it came up and any other context that might be helpful when I get back to the work.

### 4. Making a Plan

The original version of the daily task files, dating from 2004, *only* included the checklist items[3]. I used these intermittently for several years, but eventually found the checklists didn't offer the context I wanted when I had breaks on a task for more than a few days.

I have previously written a *finger* server[4]. Until recently, I'd mostly used that to advertise relatively static data, like contact information, organizational affiliation, or very high-level project descriptions. At the beginning of 2024, as part of a larger project to revisit how I was tracking (and focusing on) my technical projects, I started using the *plan* file in the more traditional way, as a prose description of what I was working on *now*. Writing out a very short prose description of what I was trying to accomplish for the day proved quite helpful. I soon added a checklist section to the bottom of the file. This helped me tell what remained to be done at a glance, and also helped track things between days. Frequently, a checklist item would persist for a few days, but the prose description accompanying it would change as I worked on the problem. In combination, these became more productive than I'd expected.

I soon wanted a way to archive my *plan* files so that, for items which took more than a day or two to complete, I could remind myself of my thinking and approach to the problem when I'd last worked on it. As I was thinking about how to archive these, I realized I'd started recreating the *daily* files I'd used years previous. Since those had years of history behind them and had been helpful at the time, I decided to return to those and integrate the plan file into it.

---

[2] Shortly after I started using this system, I discovered Bullet Journals, and have been happily using the paper versions since. That has suggested several additions to this system, most notably a "future" list for things which come up like this but aren't expected to get attention tomorrow.

[3] In the original version, each line was numbered, and the number was manually replaced with a "√" (using the square root symbol as a stand-in for a proper check) when the task was completed. The carry-forward script simply copied all lines not starting with √.

[4] See http://txtpunk.com/finger.

In the *mkfile* which controls this system, there is a target, *plan*, which will take today's file (and tomorrow's, if it is non-empty), prepend a date header, and write it to `$home/lib/plan`. My web server separately includes rules to check for updates to that file (and a few others) to update a */now* page on my web server, and also build an archive of these files in */plans*.

## 5. The Archive

Running `mk archive` will move any files in the current directory with names matching the pattern used for naming *daily* files, excluding those for today and tomorrow, into a directory named by the year and date, *YYYY/MM*; the directory will be created if needed. Note that this will include any future files beyond tomorrow. This system is not suited for long-term planning; if you'd like to use it that way, modify the `grep -v` invocation in *plan* target of *mkfile*. `mk plan` is suitable for automating with *cron*, but I do it manually, when it feels like enough work is at natural transition points. I've tried rules like "archive things more than a month old", but no such rules have felt correct enough to bother automating.

## 6. Future Work

The old *daily* files were useful for years, and adding in the prose text I started doing when playing with *finger* has addressed the main thing that caused me to stop using them. The current setup is helpful and productive, but a few lower-priority pain points remain.

The first of two main points of frustration is the fact that some of the content in these files ends up duplicating my work log, *$home/lib/work*, which is also used for time tracking and invoice generation. That file is much more focused on "work done" and descriptions tend to be shorter than the prose in these daily files, but it also often contains the *resolutions* to the work items described here. It would be nice for these to be less separate.

The second significant pain point is disconnected operation. There is no synchronization or conflict resolution if I edit some of these files on my laptop vs. my cpu server, although in practice that comes up rarely. The fact these are unstructured, short text files makes it easy to sync manually, but that's far from ideal — especially having to remember to do so. I'd like to automate that using replica; that should probably also include products of this, like the *plan* file.

Arguably, references like "yesterday" and "tomorrow" should respect weekends, but my work schedule is such that it hasn't been worth implementing. I would like to make `cf` look for the latest file before today, rather than just yesterday.

`cf` should be a target in the *mkfile* so one could (optionally) include it as a target in the *workday* target.

Finally, I don't like the manual formatting with `|f`; I'll probably automate that at some point.