PKM, PARA and Johnny Decimal

How does one locate information previously found, curated, and stored? How does one keep up as years pass and the numbers of files, filing methods, and applications go up?

There is a tension between doing what will get you *direct access* to a piece of information versus the complexity of *entering and maintaining* the access method.

At one end of the spectrum, one clicks around on a folder hierarchy with attendant false starts, back-ups, and head scratches. At the other end is the Platonic perfect coding system, in which you type in some magical combination that brings you directly and without fail to the right piece of information. As with most things, Aristotle trumps Plato, and the best solution is in the middle.

For me, that sweet spot allows for some folder-clicking to get to a topic, but also requires a method to go directly to information with minimal hunting and pecking. For those in the first 1/4 of their career, this might not seem very relevant. However it will be later!

For those who can remember everything and jump directly to it with a few keystrokes–move along, nothing to see here. But if you spend more than a few minutes a day fumbling around for a piece of data (particularly if you look for information in real-time, on a call or in a meeting) then the methodology described below might help.

A good system for knowledge management is one that grows and evolves as your needs change, and does not evolve into a straight-jacket!

PARA and Johnny Decimal

When designing a personal knowledge management (PKM) system there is a choice among simplicity, flexibility, and durability: pick any **one**. I have been designing my own PKM system for the last forty years–from well before the term was coined. All during that time I've struggled with the tension between my brain, which is deeply interested in a lot of different things–some for days, some for decades–and my file system. My brain goes all over, but my file system has remained stuck in a rut, with folder hierarchies now stretching across semi-connected systems, with a very ineffective assist from various tagging schemes.

Flexible text searching has made things much better, but I am many times either unsure of what word(s) to search for, or trying something general that works, but returns thousands of results. Meanwhile the possible places to lose my information have gone up linearly with the advent of the cloud, as Dropbox, iCloud, Google Drive, and others all struggle to be the VHS versus Betamax of cloud storage. In the battle to retrieve my stored knowledge, I'm losing. But the good news is that I am not alone, and PKM tools are coming to the rescue.

I have tried several of them–Roam, logseq, Obsidian, Noteplan, and Notion among others. All have strengths and weaknesses, and this article isn't a feature comparison; in fact, my experience with all of these systems has been that each is excellent in some way. This has made me realize that I fundamentally need to think about organization and systematization rather that features and functions.

This insight has provided me a gateway to flexibility and (I hope) durability, and has allowed me to escape dependence on any quirky software developer or open source collective. Before describing my system and the thoughts behind it, I should make it clear that my primary concerns are a) to quickly retrieve the tens of thousands (or hundreds of thousands) of facts and files I have collected over decades and b) to create a streamlined, repeatable workflow that allows me to task shift with minimum cognitive overhead.

These are my goals as a knowledge management worker with lots of projects and people to organize in a business setting and in (mostly) real-time. If your goals are more contemplative or creative–say if you are writing a dissertation or a novel–then this system might not be the best for you.

The building blocks of my system are borrowed (or stolen) primarily from two sources–although I have modified (or bastardized) each for my purposes. The first component is Tiago Forte's PARA system.[1] I also incorporated several of the ideas from Elizabeth Bulter's work.[2] The second is the Johnny Decimal System. These two are tied together through the use of Noteplan as my knowledge dashboard. I have used Roam extensively as well, but the ability to create Javascript plugins and the use of markdown files as native format has pushed me toward Noteplan.

PARA

The PARA system is a top-down method of organizing information that can be implemented in virtually any system that has a hierarchy and allows naming. Your "world" is separated into *projects*, *areas* (or as I prefer *activities*), *resources*, and *archives*. I use activities because "areas" seems vague and activities captures that idea that action occurs; I apologize to Tiago, but Activities just seems clear to me.

Projects are things you work on with dates and completion points, and they differ from **Activities** in that the latter are, to some extent, ongoing and semi-permanent.

Resources are things that are (loosely speaking) reusable and support either projects or activities.

Archives are for projects that end and activities that are no longer active.

It is worth noting that both of these systems are best defined and described by their authors or owners, and my interpretation is unique to my cognitive needs and has been adapted for my system, and at the bottom are links to those primary sources and authors.

[1] Forte, "The PARA Method."
[2] Butler, "Elizabeth Butler - Personal Knowledge Management."

Why do I find PARA so useful? First is the simplicity of dividing work up into these four unambiguous categories, rather than regarding everything as, say, a project. By focussing on the "date-ness" of *projects* you get some of the discipline of methods like GTD without all the cognitive overhead– projects start, have milestones in time, and (one hopes) end. A project review can concentrate on dates and deadlines.

Activities on the other hand might go on forever, spinning off the occasional *project* when there is a deadline or time goal. Conversely, *projects* might evolve into *activities*, as for example, training for a road race evolves into a weekly run. This close relationship can be maintained through the Johnny Decimal System as described later.

This suggests an immediate concern about clarity however. How do the two running topics get separated? One way would be in the name, so "Training for Race **Project"** would be easily distinguished from"Training for Fitness **Activity"**.

Resources are things that have value *across* multiple projects and activities. They are either permanent (like a social security number) or ephemeral, like a set of party plans. In the event that they are not relevant any longer, they can either be deleted or archived with the latter being a default since, well, you never know.

In order to simplify workflow, the choice to archive should be low effort and easily reversible since recalling from an archive has given employment to a great many monks and librarians over the ages.

The most important trait of the PARA system is simplicity: it's immediately understandable and when you context-switch and return to it, your learning curve is nil. Once the framework that a) projects have deadlines and b) activities go on forever, and c) resources support both is internalized, you can mostly forget about the tool and concentrate on work.

Johnny Decimal System

While the PARA system is very helpful in immediately and approximately deciding where a piece of information belongs, it lacks intrinsic granularity, and if implemented atop a series of nested folders it can fall victim to the two fundamental traps of unstructured hierarchy, which are inconsistency and "buried alive."

Inconsistency just means treating two similar pieces of information in two different ways, for example having lists of participants on a project called "Project X Dramatic Personae" on one project and "Project Y Ma Peeps" on a second. Spreadsheets and textual documents suffer from this all the time: on one project they are in their own directory ("Project X Spreadsheets") and in another they are specific to a phase of the project, say "Project 2 Initial Negotiations" in which one combines offers, meeting notes, and financial analyses in a single directory. While this is a problem that cannot be solved

perfectly (especially on collaborative project) it begs for some simple rules of engagement to guide *naming* and *placing*. This is where the Johnny Decimal System (JDS) shines.

A second issue is the "buried alive" problem, which is shorthand for creating hierarchies of folders that go multiple levels down and effectively conceal information, perhaps permanently. For a great example of this, see the Mac file system with particular emphasis on "Application support!" One of the key ideas of JDS is "nothing can go more than two folders deep."

The JDS website points out that remembering a prefix for a broad area, like "Finance" and a category, like "tax return" is easy with a number like 12.02, and that shorthand can get unambiguously embedded in cloud storage filenames and folders, emails, tags, memos, and spreadsheets. That's the genius of JDS–four numbers and a period to rule the world!

Basic Johnny Decimal System to Organize Folders Johnny Decimal calls for **buckets** to be created; in my version, these are aligned with the PARA system and they are Projects, Resources, Activities, and Archives.

Within each of these buckets, one creates **categories** based on what one engages with most frequently (these are likely to be categories you use currently); one gives each category a number.

In my system, Projects go from 10-19, Resources from 20-29, Activities from 30-39, and Archives from 40-49.

Remember that they are grouped in tens, so our first ten categories (say from 10-19) will all be related to each other, in this case all being groups of projects. In my version, 10-19 represent project areas such as *Work*, *Research*, *Writing*, *Financial*, and *Personal*.

The number after the decimal is just a counter. JohnnyD calls it the **ID:** it starts at .01 and increases with each bin you create.

Advantages of this system First, JDS knits together all knowledge capture methods. For example cloud storage (Dropbox and iCloud and Google Drive in my use-case) together with a knowledge dashboard like Noteplan, Roam, logseq or Obsidian.

Second, it gives life to the idea of a **permanent, transportable PKM**. After a lot of thought, I have decided that any PKM I employ will be based on markdown formatted text files; this allows me to maintain a hierarchical structure for knowledge which will remain effective as long as a) markdown structure exists and b) the system I use has the ability to group documents with at least two levels of hierarchy.

A related benefit involves scripting automation, since the ID system makes reading and relocating notes a simple matter of extracting the first five characters (after any markdown) from the file. This

works well, for example, with regex which can easily parse the ID system, like so: A good example of this "unification principle" involves email clients, which seem to be rooted in a hierarchical folder system from now until, well, forever. By using the JDS system for your email folders, they can connect up with your other storage methods.

The JDS system also works with a normal, human memory. The JDS web page says "you will be amazed at how many JD codes you will remember." With my own system and memory this hasn't exactly been the case; I've memorized quite a few often-used numbers, but certainly not "many". This might change over time, however, just having a well-established structure provides a huge advantage in knowledge retrieval. Combining the JDS number with a shallow folder hierarchy (no more than two levels deep) with a rapid text search (such as in Noteplan) solves my knowledge retrieval task.

Johnny + PARA Examples

Take financial projects. They are projects, so they begin with a 1, and (somewhat arbitrarily) things financial are my fifth project category, so they all begin with "15." The digit 5 is used here (and throughout the system)t to indicate something financial: Financial Resources are 25, Financial Activities are 35, and Financial Archives are 45. Very quickly you begin to internalize that anything financial will, at the top of it's tree, have a 5 as a second digit. Finding Financial Activities requires just remembering that Activities are all 3's and Financial is always 5.

Parenthetical note: when you are working on some activity (say something financial) it is likely that you will be moving back and forth between either a project and its resources, or an activity and its resources. This makes it easy to navigate, because you know that whatever project you have which starts with 1X as it's JDS number will have Resources in something labeled 2X.

An important distinction between "pure" Johnny Decimal and my version involves the depth of the folder structure. In the JDS "you're not allowed to create any folders inside a Johnny.Decimal folder." This is to avoid having to hunt deep into a folder hierarchy, which is a common problem with folder-based systems. This means that no note will be more than two levels deep within the system, with the first level being the simple PARA framework. I find this discipline too hard to follow, so I have (say) a financial project folder of 15, a 2021 Taxes folder of 15.01, and perhaps folders within that–however those sub-folders are all labled 15.01 also, so my hybrid doesn't break the rule completely since there isn't any nomenclature that goes more than the two levels.

Use of the System

How can you use this system in practice? It's key benefit is that it does not require replacing any of your existing tools, and being simple and text-based it can be retrofit on virtually any system.

Email

Email is a continual challenge to keep organized, and everyone has said “would you mind resending that email to me..can’t find where I put it.” Tremendous waste of time.

Do you use tags or folders? Both aren’t great, for reasons descibed in Part One. I have decided that tags are the lesser of two evils, although having to remember to tag every email seems like complete overload, and modern email clients like Superhuman and Spark attempt to automate the process somewhat..however absent true AI there help will be constrained to junk, newsletters, and notifications.

The Johnny Decimal solution for me with most emails is: nothing. It’s not worth the effort to try to label every back and forth. However, post hoc it sometimes becomes clear that a conversation is worth keeping track of, and when that happens I label the conversation.

If one knows from the start that a chain will have some import, it’s also possible to start the conversation with a JDS code in the subject line. As reply chains are built, the JDS code follows along in the subject line even if others aren’t cognizant of it; they might say “what’s the funny number for?” and you have your choice of being mysterious or explaining your system.

Cloud Systems and Launchers

Whether you use Finder, Spotlight, Launchbar or Alfred, you can open files and folders instantly almost instantly. Just type the code and you head to the directory, and depending on the tool you can get directly to the file you are looking for with a few clicks. If you use a PKM system like Noteplan (see the next section) you can use its search function to find, well, anything and the search will immediately give you the JDS code, which you can then enter into your preferred file finder.

Note Systems: Noteplan

There are many PKM systems and all of them that I’m aware of will support this system to a greater or lesser extend, especially systems like Obsidian and logseq that use hierarchies of markdown files. The one I use is Noteplan for a variety of reasons including its interface, its use of CloudKit for file synching, its developer and community, and in particular it’s Javascript API.

All the functions that you would expect are included, for example opening and editing notes, creating new notes and folders, and (important for the next section) the ability to extract all of the folder names directly from a system object. This article will not be going into detail about the NotePlan API–for that you should look at the documentation **here**. However, here is some code that illustrates how to extract

the Johnny Decimal prefix from a note and then relocate the note into the appropriate folder in the hierarchy:

Creating a Folder Hierarchy

How do you retrofit this system on top of your existing file hierarchy. As mentioned above, I use Dropbox, iCloud, and Google Drive (our company is standardized on the gSuite). One possible approach is just to incrementally add folders as you need places to store information on the file system. Using the NotePlan system as a reference, when there is a need to have a file stored permanently on a drive, you can just mirror the Johnny Decimal system (manually) onto the file system as needed.

Another possible approach for the programmer is to build the Johnny Decimal system in NotePlan (or any PKM system with an API) and then extract the hierarchy of folders into its own note. Then use that note to create a text file which can be fed into a program to create a mirror of the system's folder hiearchy. Here is code to create this note inside NotePlan, which could not be much simpler:

```
async function makeNameNote() {
  try {
    var allFolders = DataStore.folders;
    let newNoteContent = "# Johnny Folders";
    allFolders.forEach((fldr) => {
      newNoteContent = newNoteContent + "\n" + fldr;
    });
    let filename = DataStore.newNoteWithContent(
      newNoteContent,
      "Johnny Folders",
      "Johnny Folders List"
    );
    console.log("Created note with filename: " + filename);
  } catch (error) {
    console.log("Plugin code error: \n" + JSON.stringify(error));
    return false;
  }
  return newNoteContent.length;
}
```

Here is some example pseudo-code that uses the hierarchy (copyied from NotePlan into a text file named "hierarchy.txt") to create a mirrored structure on your disk. Disclaimer: this pseudo-code is

just for illustration purposes; if you code to it, be sure to backup and test before you let it run amok among your folders..but you knew that already.

```
global filePrefix = ROOT_DIRECTORY_HERE
Function getFileContents(fileNameString)
    allFileContents ← readFile(fileNameString)
    Loop For Each line in allFileContents
        fullFolderName ← filePrefix + line
        makeDirectory(folder)

Function makeDirectory(fullFolderName)
    If !fileExits(fullFolderName) Then
        createDirectory(fullFolderName)
        return True
    Else
        return

Function main
    getFileContents("hierarchy.txt")
```

For my own purposes, I coded this in JavaScript using node.js in under thirty lines of code. By changing the root directory, this allows me to mirror my file hierarchy across Dropbox, iCloud, and Google Drive; a bit messy, but that's the nature of my life since I'm on a Mac, have been a committed Dropbox user since it was first released, and work at a company that has standardized on the gSuite. Although have files spread over three different cloud systems is less than ideal, I primarily use Dropbox for storage and I use iCloud only for apps that prefer to store their data therem, and gSuite for collaborative documents which are best handled through my workplace ecosystem.

Conclusions

TIago Forte has made PARA a popular conceptual framework for knowledge management, and his book is highly recommended. Once your knowledge is segregated into the PARA form, the method of implementation begs one important question, which is: "what should the structure underneath look like?" and one answer is the Johnny Decimal naming system. The benefits of JDS include simplicity of implementation, ease of recall, and consistency across multiple storage systems.

I've implemented this system in NotePlan because of its reliance on Markdown files locally stored, it's interface, it's highly responsive developer, and its burgeoning community. During my use I've exported notes from Roam and have just under 2,000 notes in total. The system does not have any issues

with this number (as you might expect) and I'm hopeful that it will handle an order of magnitude more. I've integrated the system with Dropbox and use Hook to link files together (Hook is a remarkable utility and will be the subject of a future post to Medium).

I've developed a series of plugin routines through the NotePlan API which do things like rename and relocate a note (giving it a new JD prefix and using that to place it into the correct folder), create a note with the folder hierachy (as above), and renumber all of the notes in a folder with the correct parent JDS prefix. Utility code tests the "correctness" of a JDS prefix and verifies if a note is in the correct folder. The API is simple to understand and seems very reliable, although I have encountered some issues with latency, perhaps stemming from the API's use of CloudKit for synchronization. Workarounds have been easy to create, and as mentioned above there is a passionate and skilled developer community working on plugins and willing to help if you need it.

Overall I'm extremely happy and exited with my PKM implementation, and look forward to opening up NotePlan every morning, and it's become my most-used tool by far.

Butler, Elizabeth. "Elizabeth Butler - Personal Knowledge Management." https://elizabethbutlermd.com/, May 2022.

Forte, Tiago. "The PARA Method: A Universal System for Organizing Digital Information - Forte Labs." https://fortelabs.co/blog/para/, May 2022.