

Getting yourself organized with Org-mode

A supplement for the video course

Rainer König

August 2020



Contents

Preface	i
Conventions in this book	iii
1 Introduction	1
2 The basics of Org-mode	5
2.1 Installation of orgmode	5
2.2 Headlines & outline mode	7
2.3 ToDo keywords	9
2.4 Schedule, deadlines & agenda view	12
2.5 Repeating tasks	14
2.6 Checklists	16
3 Advanced topics	19
3.1 Tags	19
3.2 Advanced agenda views	20
3.3 Customized agenda views	23
3.4 Drawers, logging and quick notes	26
3.5 Archiving	27
4 Making things more smooth	31
4.1 Automatic logging of status changes	31
4.2 Splitting your system into several files	33
4.3 The first capture template	35
4.4 More capture templates	39
5 Workflows and time tracking	43
5.1 Ordered tasks	43
5.2 Timers	44
5.3 Clocking	45
5.4 Column view	47
5.5 Effort estimates	49

6	Linking, attachments and more	51
6.1	Linking (internal)	51
6.2	Linking (external)	52
6.3	Attachments	55
6.4	Priorities	58
6.5	Tables	60
7	Exporting and publishing	63
7.1	Exporting	63
7.2	Advanced exporting	65
7.3	Publishing	68
8	More advanced topics	71
8.1	Dynamic blocks	71
8.2	Tracking habits	74
8.3	Bulk agenda actions	77
8.4	Google calendar import	79
8.5	Working with source code blocks	81
8.6	Goal setting and tracking	84
8.7	Presenting my system	87
9	Congratulations, you're done	95
	Bibliography	96
	Index	98
A	Key combinations from all lessons	101
B	Acknowledgments	111

Preface

It was in 2016 when I was very active in the “Getting Things Done” group on LinkedIn. I was using Org-mode for long and when someone asked what GTD systems we were using in this group I tried to explain, that my GTD system is a text editor.

From the answers I realized, that most people cannot imagine how you can organize your life in a text editor. So, I had the idea to show them and recorded a quick video on June 1, 2016 that was almost 32 minutes long. Then I watched my video and thought “WTF, nobody will understand how this works”. Not only that I captured my secondary display with a resolution of 1280x1024 pixels, which is really not optimized for YouTube, it was just too much stuff. Of course, it could impress the audience, but nobody could ever learn something from that video.

So I decided to start over again and do it much better. My focus now was “one concept at a time” and short videos of 10 to 15 minutes in length. So the people watching the videos could learn Org-mode step by step and understand the concepts easily. Org-mode is a very powerful tool as you will see, but its learning curve is very steep. So my goal was to make lessons that are easy to “digest” and that also helped me to optimize my Org-mode work flow. For enabling the audience to see what is going on I searched for a tool that showed all my key pressures, so that people can really follow the use of the Emacs text editor. You probably know that Emacs stands for “Escape-Meta-Alt-Control-Shift”¹.

Now, almost 4 years later there are around 35 videos on YouTube and my channel has grown to more than 3000 subscribers. I got a lot of positive comments, and also a lot of feedback that my German accent makes me sound like Arnold Schwarzenegger.

I tried to supplement the videos with blog posts on my private blog, but it turned out that some people didn’t follow the links to the blog posts. Times are changing and my company decided to shut down our local factory, so I will lose my job this year. People then suggested to me, I should do an Org-mode course in Udemy, they would happily pay for that. So in the beginning of 2020 I decided to give it a try and remake all the videos, adding this course book as a supplement. I was watching several video courses on Udemy lately and was struggling because the instructors had a fast pace and I was not able to follow

¹Another explanation from the good old times when RAM was still very expensive and the best CPU you could get was a 486DX is “Eight megabytes and constantly swapping” which referred to the memory usage of virtual process memory.

all the things they were doing. Remembering the good old times when there was no web based training but people had to go to real classrooms I always enjoyed to get a printed course book where I can quickly lookup things later. So this book should serve this need that is probably not unique to me alone.

So I tried to make the course even better this time. The countless comments and questions on YouTube were very helpful for this project, they showed me what wrong assumptions I had or what is confusing for the people watching the videos. When this course will be recorded I'll upload it to Udemy and see if it will make me rich, but probably not. Nevertheless, the YouTube videos will remain where they are so people can still have to old and free version of the tutorial available.

Augsburg, August 2020
Rainer König

Conventions in this book

I'm trying to make this book in a way, so that you can easily follow the video lessons and also have the ability to lookup things whenever you need it.

Terminal input

When you have to type something in a command line it might look like this:

```
$ emacs mylife.org &
```

The dollar sign is indicating that you have a user prompt.

Key combinations

Sometimes inside Emacs you might need to type special key combinations in Emacs. This will look like the following:

Alt+**x** `list-packages` **↵**

That means you press the **Alt** key on your keyboard, hold it down and then press the “x” key. After releasing the keys you type “list-packages” and then you press the **↵** key.

Note that it is the lower key “x”. If I want you to press the upper case “X” then the correct notation would be:

⇧+**x**

Which then results in an upper case “X”.

Selecting from a menu

If you need to select something from a menu then it looks like this:

Org » **Documentation** » **Show Version**

So you click on the “Org” menu, inside there you click on “Documentation” and then on “Show Version”.

My platform used to make the videos

Everything you see here in this book and on the videos is produced on my Linux workstation. I'm currently running openSUSE Leap 15.1 as an operating system and I use the following packages for Emacs:

```
$ rpm -qa | grep emacs
emacs-25.3-lp151.3.60.x86_64
emacs-x11-25.3-lp151.3.60.x86_64
emacs-el-25.3-lp151.3.60.noarch
emacs-info-25.3-lp151.3.60.noarch
```

Videos were recorded with OBS (OpenBroadcastStudio). The book was edited in Emacs and produced with L^AT_EX.

Chapter 1

Introduction

Whenever you do something you usually get motivation because you know “why” you do it. So let me answer some common questions first.

Why shall I get organized?

One of the very limited resources that is distributed equally to everyone of us is time. Everyone of us gets 86400 seconds per day, no matter if we work as a underpaid intern or if we are a billionaire. Everyone gets the same amount of time and it is up to us to use it wisely.

Yes, you can argue that rich people can “make” time by delegating work to others, so they can somehow create time. But let me look at it from another perspective.

- We all get 24 hours, or 86400 seconds.
- To remain healthy we should sleep usually 8 hours per day. That means now we have 16 hours remaining.
- You can’t delegate some task to others, for example I cannot delegate having breakfast, lunch and dinner to somebody else. No matter what it is, I have to do it myself. The same is true for basic rituals like brushing your teeth or trying to remain healthy by doing sports.

So in the long run you have some time of your day left and you need to decide, what to do with it. On the other hand you have “things” to do. They can be categorized as well into 3 categories:

Have to do

Those are things that you absolutely need to do and cannot postpone it. Example: If you have a baby child that is crying because its hungry you need to prepare food and give it to the baby. No chance to postpone this for 3 hours.

Should do

Those are obligations that you usually cannot escape, some of them might

have a deadline, but somehow you still have enough time that you can “act” on them instead of “reacting” to emergencies (like the hungry baby child).

Want to do

This is what you love to do. It can be things like “spend more time with your family” or “practice piano” or even “create a course for Org-mode”. Those are the things that you are passionate about.

One of the most common problems is that many people have no idea what is on their plate. They tend to say “yes” to every job assignment, forget about appointments and are always under pressure, because they need to spend a lot of brain capacity on remembering what the need to do.

This is the time when you need to think about organizing your life better. David Allen wrote a book “Getting things Done”[AF15] where he describes a so called *trusted system* that should free you from the burden of remembering all the things you need to do.

Also note that the goal is not to fill up every second of your time with tasks. The goal is to enable you to “act” instead of “reacting” or “fire fighting”. By being on top of your ToDo list you can achieve a lot without being stressed. And in the long run you are able to achieve your goals and still have time for the “Want to do”¹ part of your life.

Its really important that you understand, that being productive does not mean that you have to run in a sort of treadmill around the clock. I’ve read the book “Brainchains” from Dr. Theo Compernelle[MDP14] and it clearly says that breaks during your day are very important. The purpose of those breaks is that your brain gets time to store the information you just got and also to do a sort of “garbage collection”. One of the well known time management techniques is called “Pomodoro” and it uses a kitchen timer. You work focused for a time slot of 25 minutes and then you have a 5 minutes break. This maximizes your productivity because your brain always has time to relax and do “housekeeping stuff”.

It was around 2010 when my wife got a cancer diagnosis. She needed to go to hospital for treatment three times a week. And suddenly I was in charge of taking care of the housework and our kids that were 8 and 10 years old in that time. Without having a trusted system already installed and working I would have probably sooner or later gotten a nervous breakdown. Using my system helped me to focus on the important (“have to do” and “should do”) tasks, so every thing that was important besides the health of my wife was written down and there was no risk of forgetting it.

At the end you have a system that takes away the fear of doing new things, because whatever idea you might have, you process it with the same simple steps:

1. Write it down. (Capture)

¹In fact it is very important that you also add the “Want to do” items to your trusted system. If you create a trusted system with the “Have to do” and “should do” items, then you have a monster that always reminds you of the unpleasant work that needs to be done. And sooner or later you will avoid using your system because of that.

2. Decide if its actionable or just information.
3. If its actionable and too big break it down to smaller tasks.
4. Decide the very next action to move towards “Done”.
5. Do your actions.
6. Frequently review your system.

This process makes doing things much easier. The trick is that you just worry about your “next” step towards the goal of completion. I remember when I was building a small wood house in my garden some years ago. I just had a basic idea on how much space I need, but I didn’t have a complete plan. So I started with laying some stone tiles as a foundation. When this was done the next step was to build a wooden frame. Then I was covering the frame with wood, so my garden house got walls. And the final step was creating a roof. But I didn’t worry about how that roof should look like until this was my “next” action. I didn’t even worry about the money to spend on that. If I had calculated everything in advance, if I would have made very detailed plans I probably would have given up the idea because it would have looked like “too much work” or “too expensive”. But living the process and always just working on what can really be done next enabled me to create that wood house in a few weeks.

David Allen does not recommend some kind of tool, you can do all those things with pen and paper, or some applications. So now we come to the next question.

Why shall I use Org-mode?

Some of the frequent questions I get asked is the reason why I prefer Org-mode over the big amount of fancy good looking smartphone and web tools that are out in the software market.

Well, I did try a lot of them and I can tell you some anecdotes about what happened. I still remember the good old time when I had a BlackBerry smartphone. I purchased a subscription of a great rated ToDo list app and yes, it somehow worked fine. But then we went down to Italy for summer holidays and since data roaming was quite expensive I turned off that and enjoyed one week of vacation. When I came back home, I found out that all my data of that app was gone. I had to ask the firm to restore it, thank god they could do it, but it was really a thrilling experience to find out that your extended memory is gone.

Then I tried several PC based and web based apps. Meanwhile I switched to an Android smartphone, so the application I used on the Blackberry was not available. I had to migrate and it was painful. I had to kiss a lot of frogs until I found an application that seemed to fit to my needs. It was IQTELL, a web based application that was also available on Android. I played around with it for a while, tried to customize it and was really a sort of power user of this app. And then the company that was providing the service announced that their business model didn’t work out and they will shutdown. Groups of users tried a sort of crowd funding which saved the application for a while, but eventually

they went out of business. So there you are, all your data in a proprietary system and all you can export and save a few CSV files.

Learning from those events I can see the following advantages of Org-mode:

- You have full control of your data files. They reside on your computer and if you lose them, then you will hopefully have a backup from which you can restore them immediately.
- The files are in plain text format. That means, you can read them easily, even if Emacs or Org-mode will totally stop working. Whatever editor you still have then, you can read the files.
- Emacs and Org-mode are Open Source projects. That means they won't go out of service because a business model does not work out as expected. Someone will always be there to maintain this software.
- Org-mode is highly customizable. So you are not bound to a work flow that some developer thinks is the best for you, you can design your own work flow that fits best to your needs.

Giving all those advantages it should be clear, why I prefer Org-mode over every other shiny new app.

Chapter 2

The basics of Org-mode

2.1 Installation of orgmode

Now let us start with the real stuff. First we want to get a recent version of Org-mode on our system.

Learning goals for this lesson

Usually when a Linux distribution comes with Emacs packages, they also ship a version of Org-mode. In my case openSUSE Leap 15.1 comes with Org-mode version 8.2.10. This is a bit outdated, so the goal is to get a more recent version.

Prerequisites

We assume that Emacs got installed on your system already.

How to install Org-mode

Start Emacs and then open a new file with the extension “.org”. That means type

`Ctrl+x Ctrl+f test.org ↵`

So now you should see a menu `Org` in the menu bar of your Emacs.

Let us see how we can find out which version is installed. You have 2 possibilities:

1. `Org >> Documentation >> Show version`
2. `Alt+x org-version`

In my case then Emacs shows “Org-mode version 8.2.10 (release_8.2.10 @ /usr/share/emacs/25.3/lisp/org/)” in the status bar at the bottom of the Emacs window.

To get a more recent version we use the Emacs package management. Usually Emacs comes predefined with the ELPA¹ repository, so all we have to do is:

```
[Alt]+[x] list-packages
```

See below if you get an error message about signature verification.

If everything works fine you get a list of packages and you will see a line:

```
org 9.3.2 available Outline-based notes management and organizer
```

“org” is underlined, so you can click on it and this will make your window split. There is an “Install” button. Click on that and you get a message box that asks if you want to install the package. Click on “Yes”.

Now the magic happens in our status bar on the bottom of the Emacs window.

When everything is done check the Org-mode version again, as described at the beginning of this section. Now I get “Org mode version 9.3.2 (9.3.2-1-g1d426e-elpa @ /home/joe/.emacs.d/elpa/org-9.3.2/)” which is the version that we just installed.

What could go wrong?

When I tried the “list-packages” on my system I got an error message²:

```
Failed to verify signature archive-contents.sig:
No public key for 066DAFCB81E42C40 created at 2020-02-01T11:05:02+0100 \
  using RSA
Command output:
gpg: Signatur vom Sa 01 Feb 2020 11:05:02 CET
gpg: mittels RSA-Schlüssel C433554766D3DDC64221BFAA066DAFCB81E42C40
gpg: Signatur kann nicht geprüft werden: Kein öffentlicher Schlüssel
```

Sorry for the German in this message, my Emacs installation is localized to German. The messages says that Emacs cannot verify the signature of the archive-contents, because there is no public key.

The problem is that my Emacs installation delivers an old public key for the ELPA archive that is expired meanwhile:

```
$ cd .emacs.d/elpa/gnupg/

$ gpg --list-keys --keyring ./pubring.kbx

gpg: /home/joe/.gnupg/trustdb.gpg: trust-db erzeugt
./pubring.kbx
-----
pub   dsa2048 2014-09-24 [SC] [verfallen: 2019-09-23]
      CA442C00F91774F17F59D9B0474F05837FBDEF9B
uid       [ verfallen ] GNU ELPA Signing Agent <elpasign@elpa.gnu.org>
```

So the old key expired on September 23, 2019. To work with ELPA archives again you need to install an update for the key. This is described here:

¹Emacs List Package Archive

²The shell input and output is partially reformatted so that it fits into the page.

<http://elpa.gnu.org/packages/gnu-elpa-keyring-update.html>

So let us do the magic as it is described on that web site.

```
$ gpg --homedir ~/.emacs.d/elpa/gnupg --receive-keys 066DAFCB81E42C40

gpg: Schlüssel 066DAFCB81E42C40: Öffentlicher Schlüssel "GNU ELPA\
Signing Agent (2019) <elpasign@elpa.gnu.org>" importiert
gpg: Anzahl insgesamt bearbeiteter Schlüssel: 1
gpg:                                importiert: 1

$ gpg --list-keys --keyring ./pubring.kbx

./pubring.kbx
-----
pub   dsa2048 2014-09-24 [SC] [verfallen: 2019-09-23]
      CA442C00F91774F17F59D9B0474F05837FBDEF9B
uid   [ verfallen ] GNU ELPA Signing Agent <elpasign@elpa.gnu.org>

pub   rsa3072 2019-04-23 [SC] [verfällt: 2024-04-21]
      C433554766D3DDC64221BFAA066DAFCB81E42C40
uid   [ unbekannt ] GNU ELPA Signing Agent (2019) \
      <elpasign@elpa.gnu.org>
```

So now we have a current key that will expire on April 21, 2024 in our keyring.
After that `list-packages` is working as expected.

2.2 Headlines & outline mode

Learning goals for this lesson

This lesson covers the general structure of Org-mode files. We learn to use Org-mode as an outline tool for our thoughts. And we learn, how to rearrange parts of our Org-mode file.

Headlines & blocks

Every headline starts with one or more “*”. Level 1 headlines have one asterisk, level 2 headlines two and so on.

Every text below a headline which is not another headline itself is belonging to that headline. Example:

```
* Level 1 headline
** Level 2 headline A
    This is some text belonging to the level 2
    headline A.
** Level 2 headline B
    This is another text belonging to the level 2
    headline B.
```

Visibility cycling

With visibility cycling you can extend or collapse the blocks under one headline or globally.

Local visibility cycling

Place your cursor on a headline and press `⌘⇧`. If the headline was collapsed then it will expand and show everything under it, if it was expanded already then it will collapse again.

Global visibility cycling

Place the cursor anywhere in your Org-mode file and then press `⌘⇧+⌘⇧`. The visibility cycles through 3 stages:

1. You see only the Level 1 headlines
2. You see Level 1 headlines and also expanded Headlines below the them
3. You see all expanded, Level 1 headlines as well as all headlines and text below them

Moving headlines around

Now that we understood the concept of headlines and underlying blocks let us move them around.

Increase and decrease headline levels

Let's assume you want to transform a level 1 headline into a level 2 headline. The easy solution would be just adding another asterisk. The bad side effect is that the indentation of the text below that headline will not be affected by that, so it might look bad.

The better to change the level of headlines is by the following key combinations:

- `Alt+→` increases the level by one and shifts the block below one character to the right
- `Alt+←` decreases the level by one and shifts the block below one character to the left

Moving up/down within a level

Inside the same headline level you can easily move the headlines and corresponding blocks up or down. Position the cursor on the headline and then press

- `Alt+↑` to move the headline and block upwards
- `Alt+↓` to move the headline and block downwards

Note that you can just move up and down within the same headline level. So for example when you're moving a level 2 headline up, you can't move it above a level 1 headline.

Refilng blocks

Sometimes you want to move a block under another headline. This action is called “refiling” and the easiest version for now is by pressing `Ctrl+c Ctrl+w`. Then you can enter the “target” of your refile operation which is in our simple case one of the headlines in your file.

Customization

There is one customization³ possibility regarding headlines. If you customize the variable *org-hide-leading-stars* to non nil then this means that only the right most asterisk of your headline is shown. Example:

```
* Level 1
* Level 2
* Level 3
```

Customizing this variable will change this on a global scope, meaning it it will affect all your Org-mode files. If you just want this customization being applied only to a specific file then put the following text at the beginning of your Org-mode file:

```
#+STARTUP: hidestars
```

If you add this line to an existing file and want to activate it immediately you need to place the cursor on that line and press `Ctrl+c Ctrl+c`.

2.3 ToDo keywords

Learning goals for this lesson

You will learn about the usage of ToDo keywords in headlines. This helps you to make your headlines “actionable” or see what items you already completed.

The default keywords

There are two default keywords:

TODO

means that this headline is an actionable item.

DONE

marks an already completed item.

³Customization means you can change the behavior of Org-mode by changing some settings in a config file or adding a customization option to your Org-mode file.

You can easily cycle through the ToDo keywords when your cursor is on a headline and you press `↑+→` or `↑+←`.

Extending the default keywords

For a more GTD like approach I have defined a few more keywords by adding this line at the start of my Org-mode file:

```
#+SEQ_TODO: NEXT(n) TODO(t) WAITING(w) SOMEDAY(s) PROJ(p) | DONE(d) CANCELLED(c)
```

The first important thing to notice is the “|” symbol. All keywords on the left side of this symbol are considered as actionable items. The keywords on the right side of the pipe symbol mark finished items.

Instead of `#+SEQ_TODO:` you can also just write `#+TODO:` to define the ToDo keywords.

You also notice that every keyword now has one character in parentheses behind it. This defines the hot key for the keyword.

Hot keys for the keywords can be used when the cursor is inside an item and you press `Ctrl+c` `Ctrl+t`. Then you will get a menu of all available ToDo keywords and pick one by pressing the assigned hot key.

Now let us have a look at the keywords I use:

NEXT

defines a *next action*. This is an action that you can do at any moment because you have everything in place to do this action.

TODO

defines an action that needs to be done, but you cannot start at the moment because you’re missing some requirement that is needed to start this.

WAITING

defines an action which only task is that you are waiting for something. This could be an answer from someone that you asked a question or even the delivery of some spare parts.

SOMEDAY

defines an action that you have not yet decided if you really want to do it or not. Its just a sort of “idea” that you want to keep in mind, but at the moment it is not really actionable.

PROJ

stands for *project* and is used to group some actions together to be a project.

DONE

marks successfully completed tasks.

CANCELLED

marks tasks that you started, but then stopped for some reasons. They are not complete, but you decided that you don’t work on them anymore.

Structuring your file

I use the level 1 headlines to group tasks according to an area of interest or focus. But this is completely up to you if you want to follow this scheme or not. Now let us just go through our easy example file:

```
* GTD releated things
** NEXT Weekly Review
```

The first block is an reminder that I should to a weekly review on my system. In the later progress of this course we will learn how to schedule this item as a recurring task. At the moment its just an easy example for a *next action*.

```
* Legal obligations
** NEXT Prepare tax declaration
    - collect all documents and send them to the tax lawyer
** TODO Call tax lawyer after documents were sent
```

This example block is a small work flow. The *next action* is to collect all documents and send them to the tax lawyer. Calling the tax lawyer is marked as TODO because without having him received the stuff it does not make sense to call him. Once the first item is marked as DONE we can change the second item to NEXT.

```
* Maintenance
** NEXT Wash the car
    - washing from outside and cleaing inside
** PROJ Repair bicycle
*** NEXT Order spare part (chain)
    - order it from an online shop
*** TODO Mount chain
    - you need to adjust the gear shifting
```

Here a small *project* for repairing the bike is defined. The first task is to order the spare part that you need. After placing the order in the online shop you can change the ToDo keyword from NEXT to WAITING. So you see that you have ordered that thing already, but you are still waiting for delivery. Once the spare part is delivered the WAITING task can change to DONE and the next step in our repair project can change from TODO to NEXT.

```
* House stuff
** SOMEDAY Build a bike port in the garden
    - Budget is $200 for the whole thing
```

This is an action that is marked as SOMEDAY because I think its nice to have but at the moment its just an idea. Sooner or later I will see that item in the weekly review and then I maybe decide to really start doing this. That would probably mean that the SOMEDAY transforms into a PROJ headline and I add some tasks like planning and getting all the material und that PROJ headline.

2.4 Schedule, deadlines & agenda view

Learning goals for this lesson

You will learn about schedules and deadlines for your actions. We also have a first look at the agenda view.

Schedule and deadline

In Org-mode the you can plan your actions in two ways:

Schedule

A schedule timestamp means, that you have planned to start working on a task on that specific day. The date can be set with `Ctrl+c` `Ctrl+s`. You will see a calendar display to pick the date. When a task is scheduled you will see a line with “SCHEDULED:” and a timestamp.

Deadline

a deadline timestamp means, that this task needs to be done until that specific date. Setting a deadline is done with `Ctrl+c` `Ctrl+d`. You will see a calendar display to pick the date. When you have defined a deadline you will see a line with “DEADLINE:” and a timestamp.

You can also set both a scheduled date to start and a deadline for your task. Example:

```
* NEXT File taxes at IRS
  SCHEDULED: <2020-04-01 We> DEADLINE: <2020-07-15 We>
```

This means you have decided on to work on your taxes on April 1 of 2020. Due to the special conditions due to the COVID19 pandemic the IRS extended the deadline to July 15. So you have both dates in your system.

Picking a date in the calendar view

You can either click with the mouse on a specific date to pick it, or if you prefer to use the keyboard you can use the `↑` key plus one of `↑`, `↓`, `←` or `→` to navigate inside the calendar view.

Agenda view

Putting your schedule timestamps wouldn’t make much sense if you would have to go through your Org-mode file and look for them. This is the point when you start using the agenda view.

An agenda view is a special view on your Org-file. It can for example show you all actions that have the Todo keyword “NEXT” or it can act like a calendar and show you what is scheduled for today or this week.

When you use the calendar view then you will see tasks scheduled for the specific day that is displayed in green color. If the scheduled day is in the

past (meaning you should work on that task already) then the line is in red. Deadlines will show up as well, but you will see upcoming deadlines a few days in advance.

So if your task is for example to get a birthday gift for your wife or husband, then its a good idea to mark the birthday as a deadline. This makes sure that Org-mode reminds you a few days before that its time to get that gift.

The agenda view is by default only accessible from the menu `Org >> Agenda Command` but you can add a key binding to your Emacs startup file (usually .emacs).

```
(global-set-key "\C-ca" 'org-agenda)
```

If you add the line above to your startup file then you can access the agenda view with the key combination `Ctrl+c a`.

Selecting your agenda files

One important step is that you need to specify which of your Org-mode files⁴ you want to see in your agenda file.

To customize this list you find a menu under `Org >> File list for Agenda`. Here you can either `Edit File List` to edit the list or `Add/Move Current File to Front of List` to add your current Org-mode file to that list.

This will set the variable *org-agenda-files*, which is a list of filenames that should be used when you generate your agenda view.

When you press the key combination `Ctrl+c a` then a menu opens. At the moment we just press the `a` key to get a sort of calendar like agenda view.

Navigating inside the agenda

Once you see your agenda for a day or a week you might want to go forward to the next day or week. In this case you use the `f` key. When you want to go back to the previous day or week you can use the `b` key.

When you press `↑+f` inside the agenda buffer, then you activate the follow mode. This means that you screen will split and you see the corresponding item of the line where your cursor is located in the agenda view in the split window.

Customization

One thing you can customize is the variable *org-deadline-warning-days* to define the number of days when Org-mode should notify you in advance about upcoming deadlines. The default value for this variable is 14 days.

If you want to set the warning period individually for a deadline in your file, then you can write this into the time stamp. Example:

```
* Example task
DEADLINE <2020-05-31 Su -5d>
```

⁴Yes, plural, you will learn later that you can use more than just one file in Org-mode.

This means that for the example task the deadline warnings will start 5 days in advance.

2.5 Repeating tasks

Learning goals for this lesson

You will learn how to make schedules that automatically reschedule the task when it is marked as done.

What is a repeating task

A repeating task is an action that you frequently have to do. Just as an example I had to learn the hard way that its a good idea to frequently clean your dishwasher with a machine cleaner. This empty run with the machine cleaner inside cleans all the pipes and so saves you from an expensive maintenance of your dishwasher.

So now I have a repeating task that reminds me all 12 weeks that I should do this.

That means, if my “clean dishwasher” task was supposed to run today and I mark it as finished then it will not be flagged as done but it will get the first `ToDo` keyword in our definition of `ToDo` keywords. And it will be scheduled for a date 12 weeks from now.

How do we do this?

If a task should repeat then this needs to be written into the timestamp of the scheduled or deadline dates. Example:

```
* Repeating tasks
** REPEAT Clean dishwasher
    DEADLINE: <2020-05-30 Sa +12w>
    - Use machine cleaner at highest temperatur
```

First you notice that I made a headline without `ToDo` keyword to collect all my repeating tasks under this headline.

Next you will notice a new `ToDo` keyword. For the convenience I added the keyword “`REPEAT`” at the beginning of my

```
#+SEQ_TODO: REPEAT(r) NEXT(n) TODO(t)...
```

definition of `ToDo` keywords for the file. The reason is simple that when you mark a repeating task as done, it won’t get the `DONE` keyword, but it will get the first keyword of the sequence. So `REPEAT` is there to jump in and also remind me that this task is repeating and won’t be finished if I mark it as done.

In case for the dishwasher cleaning I did use the deadline on purpose because then I will get the reminder a few days in advance so I have time to go and get the machine cleaner if I don’t have one in the house.

Specifying intervals

When you want to repeat a task after a given time you need to specify this time. In our example above we used “+12w”. The “+” is a specifier that we have a repeating task. The “12” is the number of weeks we want to go into the future and the “w” is the unit for 12, in this case it is weeks.

Units

You can use the following units to specify your repeat interval.

d days

w weeks

m months

y years

Repeat specifiers

In the section above I said that “+” is a specifier that we have an repeating task. But it is not the only specifier.

- + A single + means that the timestamp will be rescheduled by the interval when the task is marked as done. So for example if my initial timestamp was `<2020-04-01 We +1w>` then it will be shifted to `<2020-04-08 We +1w>`.
- ++ The single “+” works fine when you use your system daily and you always do what is needed. But imagine the situation that the timestamp was `<2020-04-08 We +1w>` and you mark the task done on April 16 because you didn’t do the task when it was due. Then with just one “+” the task will be rescheduled for, yes exactly April 15. This is probably not what you want, so you use “++” to make sure that the rescheduled timestamp is in the future. So if the original timestamp is `<2020-04-08 We ++1w>` and you mark it done on April 16, then it will be rescheduled to `<2020-04-22 We ++1w>`. So a “++” will make sure that the new due date for the task is in the future.
- .+ Now imagine you have a timestamp `<2020-04-01 We ++3w>` and you were lazy again and didn’t do the task until April 20. With the current timestamp this would mean that we make sure that the next occurrence of the task is in the future. So it will be rescheduled to April 21. Not the best idea, you would probably like something that takes the current completion date and just adds 3 weeks. This is the moment for “.+”. The dot indicates “take the current date” and then add the interval. So if your initial timestamp is `<2020-04-01 We .+3w>` and you mark it done on April 20, then it will reschedule to `<2020-05-11 Mo .+3w>`.

Customization

If you have the latest Org 9.3.6 version installed then you will get log entries every time you mark a repeating task as done. If you don't want this then you have to set the variable *org-log-repeat* to "nil". Another way to do this is to write a line with

```
#+STARTUP: nologrepeat
```

at the beginning of your Org-mode file.

2.6 Checklists

Learning goals for this lesson

You will learn the usage of checklists.

What is a checklist

Checklists are lists of short items that you can put into any block and that you just check off. It's like a pilot's checklist that has to be completed before takeoff. You check every item on your list off.

Just imagine you have a sports club (like the one where I'm the president for 30 years now) and someone applies for a membership. Then a simple checklist could look like this:

```
* NEXT New membership of John Doe [/]
- [ ] Enter membership data into database
- [ ] Generate welcome letter from database
- [ ] Generate bill for membership fees
- [ ] Send welcome letter and bill
```

You see that every checklist item starts with - [] followed by the text of the checklist item. The whole checklist is part of a NEXT action item. On the headline you also see [/]. This will give you a progress indicator. Just position your cursor behind that [/] and press **Ctrl**+**c** **Ctrl**+**c**. That will turn it into [0/4], indicating that your list has 4 items and 0 of them are currently checked off.

The progress indicator will be in red color until all items are checked off, then the color will switch to green.

Checking off items

After you entered the membership data into the database you place your cursor on that line and press **Ctrl**+**c** **Ctrl**+**c** again. The checklist will then look like this:

```
* NEXT New membership of John Doe [1/4]
```


- [X] Enter membership data into database
- [] Generate welcome letter from database
- [] Generate bill for membership fees
- [] Send welcome letter and bill

Now you see that the first item is checked off. And the progress indicator changed from [0/4] to [1/4].

If you would prefer to show a percentage instead of numbers then you could start with [%] as the initial progress indicator. After checking off the first list item it would change to [25%].

Also notice that even when all 4 checklist items are checked off your headline will still show “NEXT” as a ToDo keyword. Completing the checklist does not automatically set the ToDo keyword to “DONE”.

Writing checklists

You can easily write checklists. Start with the first line like

- [] first checklist item

When you're at the end of this line and you press **Alt** + **↑** + **↓** then the beginning of the next item (- []) will be inserted automatically with the correct indentation.

Chapter 3

Advanced topics

3.1 Tags

Learning goals for this lesson

You will learn the usage of tags.

What are tags?

Tags are a sort of *label* for headlines. They help you to label headlines according to contexts or whatever you want. You can also assign multiple tags to a headline.

How do tags look like?

Tags are basically text between colons. So `:TEST:` would be a tag with the name “TEST”. Its a good practice to write tags in uppercase letters. Tags are placed on the right border of the headline.

Tags can be used for a lot of purposes, here are some examples:

- I tag tasks that require a phone calls with a `:PHONE:` tag.
- If one of my tasks involves a family member, then I use the name of that family member as a tag.
- If I encounter a problem, then the headline will get the `:PROBLEM:` tag.
- Some people use tags to describe contexts, so if you need a computer to do a task then the task gets the tag `:COMPUTER:` assigned.

As you will learn in one of the next chapters you can create agenda views that select items with a given tag. So I can easily list all phone calls that I need to do or all tasks that involve my wife somehow.

Defining tags in your file

You can define a set of tags at the beginning of your file. Just add a line like this:

```
#+TAGS: PHONE(o) COMPUTER(c) SHOPPING(s) URGENT(u)
```

The character between “(“ and “)” defines the hot key for this tag.

How are tags assigned?

To assign a tag to a headline place your cursor in the headline or the block below it and then press `Ctrl+c` `Ctrl+q`.

This will open a small window that shows your predefined tags. So for example if I want to assign the “:PHONE:” tag to a headline I simply press `o` to assign that tag. That hot key has a toggle function, so if you press it a second time the tag gets unassigned.

Sometimes you want to assign a “free text” tag that is not predefined in your file. For that you just need to press `*→` and write the tag finished with `↵`.

Tag inheritance

If you had a close look at the menu that opens when you assign a tag you saw that there is “Inherited:” at the top. Tags can be inherited down the headline hierarchy. Example:

```
* Maintenance jobs           :MAINTENANCE:
** TODO clean dishwasher    :KITCHEN:
```

You have defined a section for maintenance jobs in your file and you assigned the tag `:MAINTENANCE:` to it. As one of the tasks you defined “clean dishwasher” and assigned the tag `:KITCHEN:` to it. But the “clean dishwasher” task also has the tag `:MAINTENANCE:` inherited from the level 1 headline.

3.2 Advanced agenda views

Learning goals for this lesson

You will learn more advanced ways to use the agenda view.

What agenda views are there?

Whenever you use the hot key¹ `Ctrl+c` `a` a menu with the agenda options will be shown in a split window.

¹Don’t forget to define that key binding in your `.emacs` file as it is described on page 13.

Agenda for current week or day

Just press **a** to access a day or week agenda. We already used that key when we introduced the agenda view in section 2.4.

The text in the menu says “Agenda for current day or week”. You can influence what is displayed in two ways.

First you can customize the variable *org-agenda-span* and set it to one of the following values:

day displays the agenda for the current day

week displays the agenda for the current week

month displays the agenda for the current month

year displays the agenda for the current year

The second way is by prefixing the agenda command. Prefixing is done by the key combination **Ctrl**+**u** followed by a number. Prefixing is working no matter what value you have defined for *org-agenda-span*. So if I want to get the agenda for today and the next 2 days, then I need the following key combinations:

Ctrl+**u** **3** to set the prefix for 3 days
Ctrl+**a** **a** call the agenda for days.

The result is an agenda view that says “3 days-agenda” in the headline.

List of all TODO entries

Pressing **t** in the agenda menu will show a list of all items that are actionable. Remember our definition of *ToDo* keywords on page 10 and that every keyword on the left side of the “|” symbol is considered actionable.

So this type of agenda view offers you a quick list of all items that still need to be done.

The headline for this view says “Global list of TODO items of type: ALL”.

Below that line you see “Available with 'N r’ and then a list that consists of “ALL” and all your defined *ToDo* keywords. Every keyword has a number assigned. So if you just want to see the list for any of those keywords you just need to type the number and then press the **r** key. If you want to go back to the initial “ALL” list you press **0**+**r**.

Entries with a special TODO keyword

Pressing **↑**+**t** will give you the opportunity to enter *ToDo* keywords that should be displayed in the agenda view. So imagine you would like to see all “NEXT” and “TODO” entries, then you would type **NEXT|TODO** and you get an agenda view that lists all entries with either “NEXT” or “TODO”.

Match a TAGS/PROP/TODO query

Pressing will give you the opportunity to enter tags and logical expressions. Remember the :PHONE: tag in section 3.1? Now we can create a query to display all items with the keyword “NEXT” and where the :PHONE: tag is assigned:

```
PHONE+TODO="NEXT"
```

Notice that the tag is written without the colons. The “+” is a logical AND operator.

The match with the lowercase will list all headlines, that have tags assigned. So if I just would query for “MAINTENANCE” in our snippet when we discussed tag inheritance on page 20 I would get both headline.

If you just want to see lines that have a ToDo keyword, then you can use the uppercase +. This filters for headlines with ToDo keywords.

Search for keywords

Pressing enables you enter full text search terms. This will search within blocks (headline + content) for the search term. Similar to matching you can also limit the search to items with ToDo keywords by using the uppercase +.

Search terms can be simple text like “computer” or boolean or regular expressions. Text used to search is case insensitive so “computer” as a search term would match both “computer” and “Computer”.

A binary search term would look like this:

```
+repair +car
```

searches for all items that contain both “repair” and “car”.

Regular expressions can be used if you are not sure how the word you are looking for is written in the file. If you search for “color” but you are not sure if its written as “colour” then you could use the a regular expression like this:

```
{colou?r}
```

The first thing you may notice is that regular expressions are written in curly braces. The “?” after the “u” in the regular expression defines, that you are looking for zero or one occurrences of the letter before the “?”, so the regular expressions matches both possible spellings.

So if you would search for “red color” then a good search expression would be

```
+red +{colou?r}
```

Boolean expression is indicated by the “+” characters, so both “red” and any spelling of “color” or “colour” must be in you text to match this search string.

3.3 Customized agenda views

Learning goals for this lesson

You will learn how to define your own custom agenda views that you use frequently.

What are customized agenda views good for?

Usually there are some queries that you frequently use on your system. Of course you could always type them in, but because you use them frequently it would be much better to “automate” them, just define them once and assign a menu key for it. Next time you need the same query you just use that hot key and its done.

Defining customized agenda views

There are 2 ways to define customized agenda views:

Custom Agenda View Editor

When you press `Ctrl+a a` and then `↑+c` you will enter the custom agenda view editor where you can easily define your agenda views.

Directly in your startup file

Custom agenda views are usually defined in the variable *org-agenda-custom-commands* as Emacs Lisp code. Once you understand the structure its usually easier and faster to edit the custom agenda views there.

The Custom Agenda View Editor

When you press `Ctrl+a a` to get the agenda menu then you can enter the editor for custom agenda views with `↑+c`. When no custom agenda view was defined so far then you get a first template that looks like what you see in figure 3.1.

The first important thing is that you define “Access Key(s)” which you will later use to access this custom agenda view. In a simple system you can use one key that is not already used for the built-in agenda views. But you can also create your own menu structure with sub-menus. So your top level custom agenda item could be “Agendas” with the hot key “A” and then an agenda for today. That means this agenda for today would then have the keys “AT” (“A” to enter the “Agendas” menu and “T” for “Today”) assigned.

Next is a description that will show up in the agenda dispatcher menu. This should be descriptive enough so that you know what you will see when you select that custom agenda view. So “my custom agenda” would be a bad choice, but “Daily agenda and phone call list” would be a good one.

In the screen shot you will also see that we have 2 components, one “Agenda” and one “TODO list (all keywords). There are buttons with `INS` and `DEL` to insert or delete elements.

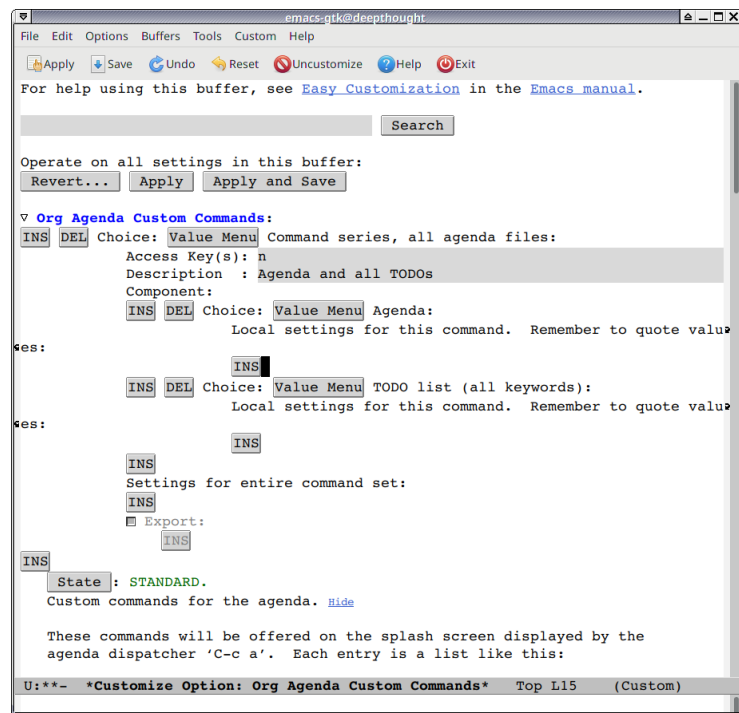


Figure 3.1: The custom agenda view editor

In the screen shot you also find an **INS** button to define local settings for this command. So for example you could define a value for the variable *org-agenda-span* here to set the number of days that are shown in the agenda.

When you are done you press **Apply and Save** on top of the screen to save your custom agenda definitions to your Emacs startup file. Then you can use your custom agenda view.

Editing the startup file

Once you have used the custom agenda editor you can look into your Emacs startup file and you will find the Emacs Lisp code that defines your custom agenda view. Example:

```
'(org-agenda-custom-commands
  (quote
    (("A" "Agenda and NEXT phone calls"
      ((agenda ""
        ((org-agenda-overriding-header "Agenda")
         (org-agenda-span
          (quote day))))
        (tags-todo "PHONE+TODO=\\"NEXT\\"")
         ((org-agenda-overriding-header "NEXT phone calls"))))
      nil))))
```

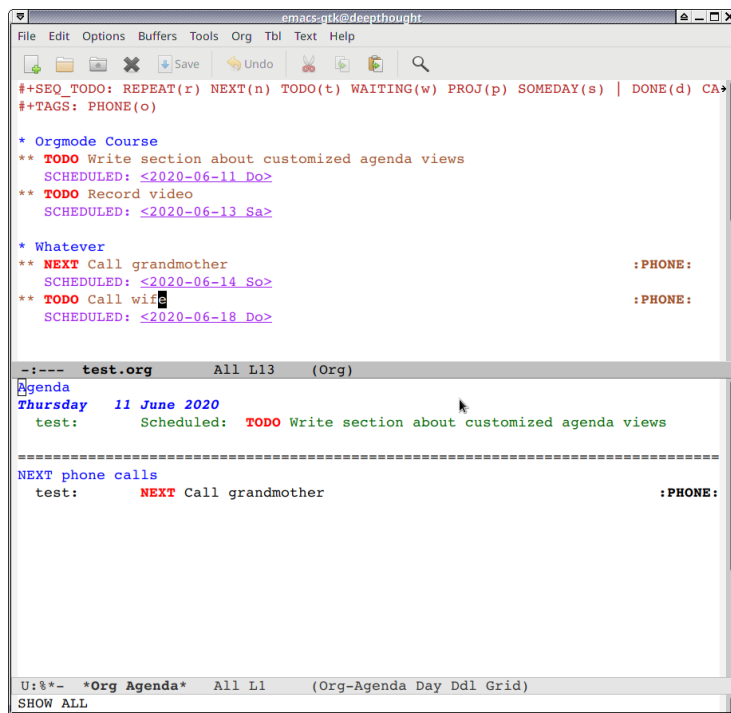



Figure 3.2: Example for custom agenda

Don't get confused by the big amount of “(“ and “)”, that is just part of the Lisp language. In the Third line we find the hot key “A” and the description “Agenda and NEXT phone calls”. Below that we have an agenda view that has 2 parameters set. First we set the variable *org-agenda-overriding-header* to “Agenda”. This will be the header for this agenda part. Next we set the variable *org-agenda-span* to “day”, so that we just get the entries for the current day.

The next part of our custom agenda is what we would get with the $\uparrow + m$ key in the agenda dispatcher menu. We define a match string `PHONE+TODO="NEXT"` to match all items that have the tag `:PHONE:` and the ToDo keyword “NEXT”. The header for this section is set to “NEXT phone calls”.

In figure 3.2 you see a screen shot that shows a simple Org-mode file in the upper part and the resulting agenda view in the lower part.

The screen shot was taken on Thursday 11 June 2020 as you can see in the agenda view. The upper line shows the header “Agenda” and then you see the items scheduled for this day. So you see “Write section about customized agenda views” because it was scheduled for that day, but not “Record video” or one of the phone calls that were scheduled for other days in the future (seen from June 11).

Then you see a line of “=” which starts the next part of our agenda. The next header is “Next phone calls”. Here only “Call grandmother” is listed, because it has “NEXT” assigned as a ToDo keyword. “Call wife” is not listed because the ToDo keyword assigned to this task is “TODO”.

3.4 Drawers, logging and quick notes

Learning goals for this lesson

You will learn the concept of drawers and how you can create a logbook with notes for your tasks.

What are drawers

You probably know those tidy houses where everything looks neat. Some nice book shelves and some cupboards with drawers. And if you open the drawer you see the hidden mess.

The concept of drawers in Org-mode is a bit like our tidy house. Drawers help you to “hide” stuff that would just distract you and that you usually don’t want to see. This means, in a collapsed view you just see the drawer name. Of course you can open the drawer and see what is in there, similar to your cupboard in the house.

Anatomy of a drawer

Drawers start with a line that has the name of the drawer between colons “(:)”. Drawer names are usually written in uppercase. Everything until the next line that has the `:END:` tag for that drawer is considered to be the contents of that drawer.


You can name drawers how you like, but there are some reserved names for drawers:

PROPERTIES

This drawer holds special configuration information for the current item or subtree in your Org-mode file. The PROPERTIES drawer has to start immediately under the headline². Only the schedule and deadline information can go between the headline and the start of the PROPERTIES drawer.

LOGBOOK

This drawer is used to log things. To enable logging to the logbook drawer you need to customize the variable *org-log-into-drawer* and set it to “t”. Then all logging goes into the drawer called LOGBOOK.

Drawers can be opened (unfolded) and closed (collapsed) by pressing the  key.

What is logging good for?

Logging in Org-mode means that you create a sort of micro blog for every task that you are doing. This will be very helpful for your future self, because you can see what actions you did on a specific task.

²This change was introduced in Org-mode version 8.3. You might find older examples on the web where the PROPERTIES drawer is located at another place

When you use Org-mode to track problems that you need to solve then a logbook that records my analysis and the solution that I found will be very helpful when a similar problem occurs in the future. And believe me, usually problems come back sooner or later, and with a logbook you have a good chance that you already solved that issue in the past and just have to revisit your logbook to see.

When I was leaving my previous job I was sending the “goodbye letter” to my internal contacts. A product manager replied and wanted to know if I still remember what happened with a specific problem that I worked on around a year ago. I would not have remembered what happened, but hey, I got my logbook drawer where I recorded every information. So I could answer, that we were unable to reproduce the customer problem and since we lost that sales project we did not investigate the problem any further. I could even exactly tell the dates when things happened.

Logging is also a very good tool for the common CYA³ strategy. When you have detailed notes on what is going on with your tasks nobody can blame you for mistakes that you are not responsible for.

How do I take a quick note

When your cursor is inside the task or even on a line with the task in an agenda view, you just need to press `Ctrl+c` `Ctrl+z` to open a window where you can type your note. Once you’re done press `Ctrl+c` `Ctrl+c` to store the note in the logbook.

If you open the LOGBOOK drawer of that task you should see your note at the top. Every quick note that you capture with `Ctrl+c` `Ctrl+z` will start with a line like this:

```
- Note taken on [2020-06-12 Fr 18:42] \\\
```

So you have a timestamp that records exactly date and time when you wrote this note. Timestamps for notes are in square brackets (“[” and “]”) which means that those entries will not show up in the agenda view. Timestamps with “<” and “>” would show up in the agenda view.

Below that line and perfectly aligned under the “N” of “Note” you find the text of the note that you entered.

As I explained above, the LOGBOOK drawer is organized like a micro blog, that means the newest entries will be always on top.

3.5 Archiving

Learning goals for this lesson

You will learn how to archive entries in your Org-file.

³Cover Your Ass

Why archiving?

When you use Org-mode to organize your life and you use it every day then your Org-file will grow. But there will be entries that are already done, so we could archive them so that your Org-file remains small which might speed up the work.

What archiving methods are possible?

Org-mode offers two archiving methods:

Internal archiving

Internal archiving just sets an `:ARCHIVE:` tag which disables expanding that entry and prevents that entry from showing in agendas. Internal archiving is done with the keys `Ctrl+c` `Ctrl+x` `a`.

Moving subtrees

Moving subtrees means that you move the subtree to another file, the archive file. This is the better approach because so your Org-file will remain small since the archived entries end up in another file.

So in this lesson we will focus on moving the subtrees to another file.

Moving subtrees to an archive file

The first question is of course, which other file? You can define an archive file by putting a line like this

```
#+ARCHIVE: %s_archive::
```

at the start of your Org-file. “%s” stands for the filename of the Org-file, so when your Org-file is named “mylife.org” the archive file will be “mylife.org_archive”.

You could also customize the variable *org-archive-location* to set such a string globally.

You can also set the archive target for a subtree. I have one top level headline for the books I want to read. So my setup looks like this:

```
* Books to read
:PROPERTIES:
:ARCHIVE: track-read-books.org::* Read books
:END:
```

Below there I have a list of “SOMEDAY” for every book I want to read and once I start reading it it goes to “NEXT” and then “DONE”. After reading it I archive the entry and because of this setup all my read books end up in a file “track-read-books.org” under a level 1 headline that says “Read books”.

The archiving of a subtree can be done with either `Ctrl+c` `Ctrl+x` `Ctrl+s` or shorter `Ctrl+c` `$`.

There is also a prefixing option. When you place your cursor on a headline and press `Ctrl+u` `Ctrl+c` `Ctrl+x` `Ctrl+s` then Org-mode will check all

children of that headline (all lower headlines) if they are considered as “DONE” and will prompt you if you want to move that item to the archive file.

Important: When you move your subtrees to the archive file, then Org-mode will just modify the Emacs buffer for that archive file, but will **not** write them to your hard disk automatically. So after you did some archive operations it is always a good idea to do the following command:

Alt + **x** `org-save-all-org-buffers`

This will write all modified Org buffers to the hard disk. This function can also be called from the agenda view by just pressing **Ctrl** + **x** **Ctrl** + **s**.

Revisiting archive files

Now imagine that you want to “search” your archives. Since everything is just plain text the most simple way could be by running a grep on your orgfiles.

Another way is to open the archive file in Emacs. Archive files are also Org-files, so you will see something that looks familiar. Now I will show you a smart way to work with archive files.

Just imagine I remembered that I read some book about habits, but I don’t remember the details. Well, it should be in my “track-read-books.org” file, shouldn’t it.

So I open this file. Then I switch to the agenda dispatcher with **Ctrl** + **c** **a**.

Now I press the **<** key which is described as “Buffer, subtree/region restriction”. The line at the bottom of my Emacs window will now say “Press key for agenda command (restricted to buffer):”.

This means that no matter what the variable *org-agenda-files* is set to, this will be ignored for this instance of the agenda command. We just use the current buffer (my reading archive) to build the agenda.

So now I press **s** and enter the search term “habit”.

The result will look like this:

```
track-books-read:DONE Book: Charles Duhigg - The Power of Habit
track-books-read:DONE Book: James Clear - Atomic Habits
```

So I see, that I read 2 books about habits. If I activate the follow mode with **↑** + **f** I can move my cursor through the agenda and see in the other window the details, So I learn that I finished Charles Duhigg’s book on the December 7, 2018 and I read “Atomic Habits” on January 8, 2019.

Chapter 4

Making things more smooth

4.1 Automatic logging of status changes

Learning goals for this lesson

You will learn how to we can automate the taking of notes when the status of task changes.

Why do I want this?

So far we know how to add quick notes to any task in our system. But now we want the system to prompt us directly when the status expressed in the ToDo keyword of this task changes. Examples:

- When a task is done, I just want to get prompted for a comment on the outcome of this task.
- When a task changes from lets say “SOMEDAY” to “TODO”, meaning that I finally decided that I really want to do this task, then I want to write a note about this decision.

How do I activate the automatic note taking

Since it is all about the change of the ToDo keyword assigned to a task we put the automatism into our definition of ToDo keywords.

So far we know how to assign hot keys like

```
#+SEQ_TODO: NEXT(n) TODO(t) ....
```

Now we extend the term between “(“ and “)”. Imagine we want to record a note whenever we change the ToDo keyword of a task to “TODO” and we just want to record a timestamp when we change the keyword from “TODO” to something else. Then the definition above would look like this:

```
#+SEQ_TODO: NEXT(n) TODO(t@/!) ....
```

The “@” indicates that we want to log a timestamp and a note when we change the keyword to “TODO”. The “!” on the right side of “/” defines that we want to log a timestamp when we change the keyword from “TODO” to something else.

Of course you could also write “@/@" to log timestamps and notes in both cases.

Also understand that if you have a definition like this:

```
#+SEQ_TODO: NEXT(n@/!) TODO(t@/!) ....
```

And now you change the keyword of a task from “TODO” to “NEXT”, then Org-mode just records one timestamp and the note. Your Logbook entry will then look like this:

```
:LOGBOOK:
- State "NEXT"          from "TODO"    [2020-06-16 Di 21:02] \\
  My note for the status change.
```

So you see that you entered the “NEXT” state coming from “TODO” on the recorded timestamp.

Similar to taking quick notes your note is submitted by pressing `Ctrl` + `c` `Ctrl` + `c`. Sometimes you were prompted for a note, but you don’t want to write a note, then you can kill the entry of a note by pressing `Ctrl` + `c` `Ctrl` + `k`. In this case neither a timestamp nor a note will be logged.

More Customizing

Disable automatic logging for a task

Since the ToDo keywords are defined on a file or global level they will work on any task item in your file. But sometimes you have tasks, especially repeating tasks where you don’t want to log anything, you just take them as a friendly reminder.

If you want to disable this automatic logging for any task, then you should have a line in the “PROPERTIES” drawer:

```
:PROPERTIES:
:LOGGING: nil
:END:
```

In this case automatic logging is disabled, but you can still write down quick notes for this task.

Org-mode variables that influence automatic logging

There are some variables that you can customize that influence automatic logging.

org-log-done

This variable defines globally, if tasks that are finished (meaning getting a keyword assigned that indicates “done”) will create an logbook entry.

org-log-reschedule

This keyword defines if you want to create a log entry whenever you reschedule a task. I recommend logging at least a timestamp for every time you reschedule a task, because then you can easily see the tasks that you always procrastinate by just looking at your logbook to see how many lines say “Rescheduled on...”.

Both variables can have 3 values:

nil means no logging at all.

time logs a timestamp.

note logs a timestamp and a note.

You can also set the values on a per file level by adding lines like this:

```
#+STARTUP: lognotedone logreschedule
```

This sets *org-log-done* to “note” and *org-log-reschedule* to “time” for the given file.

4.2 Splitting your system into several files

Learning goals for this lesson

You will learn how to work with multiple Org-mode files. We will also have a closer look at refiling items.

Why do I want this?

So far we just use one file like “mylife.org” for our system. It holds everything, but maybe we would like to separate the stuff that we do in our job from the things we do in our private life.

Figure 4.1 shows what we want to achieve. The file “mylife.org” has both entries for work (red) and private (green) items. We want to create 2 new files, “work.org” that should get all work items and “private.org” that gets all the private things.

How do we split things up?

Of course we could copy “mylife.org” to “work.org” and also to “private.org” and then edit the 2 files and delete all what doesn’t need to go there. That’s the quick and dirty way, but we want to go a bit deeper into refiling. We had a first contact with refiling in section 2.2, but now we want to dig deeper into it.

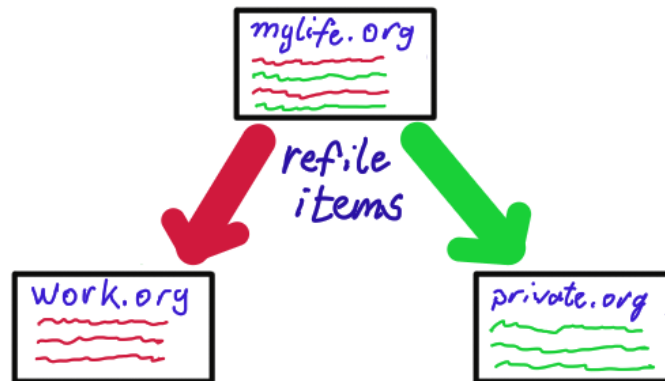


Figure 4.1: Splitting work from private stuff

Customizing refiling

We need to customize 3 variables for an elegant way to refile items:

org-refile-targets

This defines where refiling can place items. If this variable is set to “nil” then refiling only happens in the current buffer. For our purpose we set it to `org-agenda-files :maxlevel . 2` which means that items can be placed in all agenda files below headlines of level 1 and 2.

org-refile-use-outline-path

This needs to be set to “file” which means we see the target path including the file name of the Org-file where we want to put the item in.

org-refile-allow-creating-parent-nodes

This defines if we are allowed to create new parent headlines in the target file. We set this to the value “confirm” which asks for confirmation if the target we entered would create a new headline.


As we have seen above we also need to define *org-agenda-files* now because *org-refile-targets* is using this list to determine in which files items can be placed. So at the moment we set it to “work.org” and “private.org”.

The refiling process

As we learned in section 2.2 we can move entries with `Ctrl + c Ctrl + w`. If we do that they will disappear from the old place and moved to the new target.

If we want to keep our old file “mylife.org” with all the entries then its better to copy the entries during refiling. Copying entries is done with `Ctrl + c Alt + w`.

So when we want to refile items we place the cursor on the headline and then use one of the key combinations above. On the bottom line then the text “Refile subtree *item* to (default *last-target*:” which states the name of the headline

and also shows the last used target. We can write the filename e.g. “work.org” followed by a “/” and then the headline under which this needs to be filed. When writing those things you can always press  for an auto completion. If you want to create a new headline then just write it after the “/” and Org-mode will ask for confirmation if this new headline should be created.

A hack regarding agenda files

In the example above we just setup the list variable *org-agenda-files* to “work.org” and “private.org”. That means when we use the agenda view we will get entries from both files in one agenda view.

But maybe we want to really split things up. If we are in a work situation, we might only want to see work items in the agenda view, if we are in a private context we don’t want to see our work stuff. If you have defined custom agenda views you now could start multiplying them and just modify the list of agenda files used to create that custom view. That is possible, but your custom agenda views command will get very long.



So I made a simple Emacs Lisp hack in my Emacs startup file.



```
(defun org-focus-private() "Set focus on private things."
  (interactive)
  (setq org-agenda-files '("~/org/private.org")))



(defun org-focus-work() "Set focus on work things."
  (interactive)
  (setq org-agenda-files '("~/org/work.org")))



(defun org-focus-all() "Set focus on all things."
  (interactive)
  (setq org-agenda-files '("~/org/work.org"
    "~/org/private.org")))
```

Here I have defined 3 interactive functions that set the list variable *org-agenda-files*.

So now, if I want to switch my focus, I just type  +  and type the name of the function.

 +  **org-focus-work** would set the focus to work only.

 +  **org-focus-private** would set the focus to private stuff.

 +  **org-focus-all** would use all in my agendas.

The big advantage is that you just have to define your custom agenda view once and you can use it in different contexts.

4.3 The first capture template

Learning goals for this lesson

You will learn how to work with capture templates.

Why do I want this?

So far we were just entering our tasks by typing editing our Org-files. With capture templates you can define sort of forms to capture your ideas in a very smart way.

Prerequisites

To have an easy access to the capture function we assign a hot key for it. Add the following line to your Emacs startup file:

```
(global-set-key (kbd "<f6>") 'org-capture)
```

Once this is done and Emacs has read this definition you can always access the capture function by just pressing **F6**.

Defining the first template

Using the template editor

Once you press **F6** a window will open. The default is that you can capture a task by pressing **t**. We want to create our first template so we press **↑** + **c** here.

This opens an customizing editor for capture templates. To keep things simple at the moment we just start a menu tree for capture templates.

Figure 4.2 shows the editor view. On top we have something called “Multikey description”. The line “Keys” assigns “p” with the description “Private templates”. This means, next time we press **F6** to enter the capture function the menu will just show “p Private templates”.

The next definition is the “Template entry”. Keys assigned is “pt”, and the description is “TODO entry”. This means that once you press **F6** **p** and then **t** the template will be used.

The line “Capture Type” says that we want to capture an “Org entry”. After that we have to define a target location, the place where the captured entry should be placed. We chose “File & Headline”. The filename is “private.org” in our “org” directory. Inside that file we have a headline called “Capture”.

Now comes the Template type. We chose “File” here and use the file “tpl-todo.txt” from our “org” directory.

Anatomy of a template text

In our template we decided to take the real text for the template from a template file that we named “tpl-todo.txt”. This file looks like this:

```
* TODO %^{Description} :NEW:
  Desired outcome: %?
  :LOGBOOK:
  - Added: %U
  :END:
```

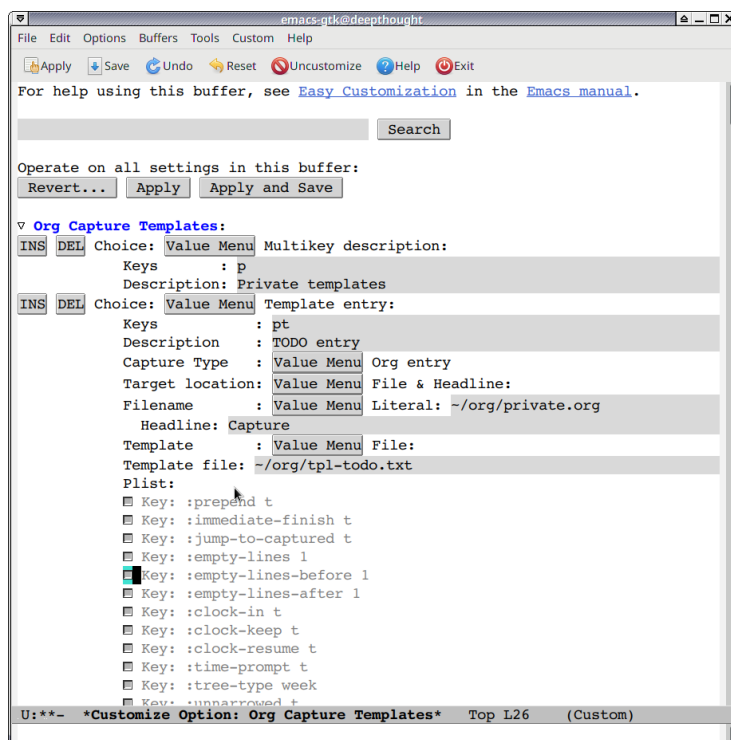


Figure 4.2: The first capture template in the editor view

The first line creates a header with the `ToDo` keyword “`TODO`”. The percentage sign is a special character that is like a macro definition. `%{Description}` means that the user will be prompted to write a description that then will be placed instead of that term. At the end of the line we put a tag named `:NEW:` so that we can easily see what is new.

The next line is “Desired outcome: “. The `%?` indicates that after the text from the template file is copied the cursor will be placed at this point.

Below that we create an initial logbook drawer that gets a line “Added: %U”. The `%U` will then create a timestamp. So your first logbook entry is the timestamp when you captured this `TODO`.

As explained above the cursor will be placed after “Desired outcome:” on the second line. That means you can think about what outcome you wish to achieve and write it down here. You can even write some more text and facts about that task.

Once you have finished the task description you press `Ctrl` + `c` `Ctrl` + `c` to save the entry at the target location, in our case inside the file “`private.org`” under the headline “* Capture”.

If you want to kill the capturing process you can also press `Ctrl` + `c` `Ctrl` + `k`, in this case no entry will be created.

Also note: Our template text has just one star, but since our target location is under “* Capture” which is a level 1 headline the capture process will create your new entry as a level 2 headline with two stars at the beginning.

How does this look in Emacs Lisp

Our first capture template definition is as usual stored in Emacs Lisp format in the startup file. So what we have created in the Editor above now looks like this:

```
'(org-capture-templates
  (quote
    ("p" "Private templates")
    ("pt" "TODO entry" entry
      (file+headline "~/org/private.org" "Capture")
      (file "~/org/tpl-todo.txt")))))
```

Once you have gathered a bit of experience you can easily understand what is going on. And you could also easily create a similar menu tree for “Work templates” assigned to the key “w” by just copy and paste at the right place. Hint: Count the parentheses to see the structure.

Chose your capture structure

At this point you should think a bit about your system. At the moment we have 2 files, “`private.org`” and “`work.org`”. If you want to capture for both files you practically have 2 possibilities.

- You create a menu structure that addresses the area of focus on the top level, e.g. “p” for private things and “w” for work. Then you create all

capture templates under both subtrees of *org-capture-templates*. The good part is that captured items at least end up in the file where you finally want them.

- You create a flat hierarchy and your target location is an extra file, e.g. “capture.org”. So everything you capture will end up in “capture.org” and you need to refile it to the final destination.

There is no “right” or “wrong” in both ways. I used the first way for several years now, but I think the second way could also work for me.

4.4 More capture templates

Learning goals for this lesson

You will learn more ways to use capture templates. I will just show you some examples of templates that I use and that proved to be helpful.

A journal template

If you want to use Org-mode for journaling, then this template might be perfect for you.

Template code

The Emacs Lisp code of this template is quite simple:

```
("pj" "Journal entry" entry (file+olp+datetree
  "~/org/journal.org") "* %U - %{Activity}")
```

Journal template in action

What is new here is that now we file to a “datetree”. So the Org-file “journal.org” has a special layout, the so called datetree. When using this template on an empty file and writing an entry the result looks like this:

```
* 2020
** 2020-06 June
*** 2020-06-20 Saturday
**** [2020-06-20 Sa 19:51] - Journal entry
This is my journal entry.
```

The level 1 headline is “2020” which is at the moment of writing this the current year.

Level 2 headlines are months, so we have “2020-06 June”.

Level 3 headlines specify the current date, it is “2020-06-20 Saturday”.

The level 4 headline is what you have captured. If you see the template code you find an “%U” which creates that timestamp with the date and the time.

A book reading list

I organize my “Books to read” list with Org-mode. So whenever I get a book recommendation, I capture this.

Template code

```
("pb" "Add book to read" entry (file+headline "~/org/private.org"
  "Books to read") (file "~/org/tpl-book.txt")
  :empty-lines-after 2)
```

Its almost the same as our first ToDo template. The target is our “private.org” Org-file with the level 1 headline “Books to read”. We use a template file and when storing the captured text we tell Org-mode to put 2 empty lines below it.

The file “tpl-book.txt” looks like this:

```
* SOMEDAY Book: %^{Author} - %^{Titel}
:PROPERTIES:
:Author: %\1
:Title: %\2
:END:
- Recommended by: %?
:LOGBOOK:
- Added: %U
:END:
```

The to-read template in action

When using the template we create a SOMEDAY entry in our to-read list. The user will be first prompted for the author of the book and then for the title.

What is new now is that we also create a PROPERTIES drawer with 2 self defined properties named “Author” and “Title”. The “%\1” means that we use the first value that was prompted already (the author) and the same applies to the second value. A LOGBOOK drawer is also created with the timestamp of the capture. The cursor is then placed after “Recommended by: “ so we can write down who recommended this book to us.

Here is an example for the result of such a capture:

```
* Books to read
** SOMEDAY Book: Chris Guillebeau - The $100 Startup
:PROPERTIES:
:Author: Chris Guillebeau
:Title: The $100 Startup
:END:
- Recommended by: Katrin
:LOGBOOK:
- Added: [2020-06-03 Mi 16:59]
:END:
```


So when I start reading this book I change the `ToDo` keyword to `NEXT` and due to automatic logging of status changes I get a note with a timestamp. So I can see when I started reading. And at the end the book is `DONE` which will also get a timestamp. In between I can do quick notes to write down some things I'm taking away from that book, just in case I have to write a review.

Capture a checklist

In section 2.6 we gave an example for a simple checklist if someone joins a club. So why not using a template file to automate this process a bit. The template code is nothing new, and the template file can look like this:

```
* NEXT New membership of %^Name [1/4]
- [ ] Enter membership data into database
- [ ] Generate welcome letter from database
- [ ] Generate bill for membership fees
- [ ] Send welcome letter and bill
:LOGBOOK:
- Added: %U
:END:
```

So all we need to do now is calling that template and entering the name of the new member and our Org-file will get the whole checklist including a LOGBOOK with the usual “Added” timestamp.

Final thoughts on capture templates

I hope that you now have some idea what powerful tool capture templates are. Just think on our last checklist template, you could also use a template file that instead of using a checklist you creates a project (ToDo Keyword `PROJ` and then creates 4 subsequent `TODO` entries.

Capture templates enable you to write down any idea quickly, so I use them daily. Whenever some idea pops up, I can capture it and the idea won't vanish.

Chapter 5

Workflows and time tracking

5.1 Ordered tasks

Learning goals for this lesson

You will learn how to enforce that tasks will be completed in a predefined order.

What does predefined order mean?

So far we managed tasks in a very simple and open way, you were able to mark any task at any given moment as “done”. But sometimes a sequence of tasks must be enforced. Imagine you have a project like this:

```
* PROJ Build a house
** TODO Build the basement
** TODO Build the ground floor
** TODO Build the roof
```

So far you were able to mark “Build the roof” as done, but in reality you cannot build the roof when there is no basement and no ground floor.

So we need to have a way to enforce an order, meaning that you have to complete “Build the basement” first before you can complete “Build the ground floor” and so on.

How is this done?

The trick is done with a special property named `ORDERED` which has to be placed for the parent task, in our example the “Build a house” project would get this property.

If this property has a value of “t” (true) then the order of the tasks is enforced.

You can easily toggle this property with the key combination `Ctrl + c Ctrl + x ↑ + o`.

Customizing variables

There are some variables that you can customize to fine tune the behavior.

org-enforce-todo-dependencies

controls, if you are able to mark a parent task as “done” when there are child tasks that are still open. If you set this to “t” (true) then you cannot mark the PROJ task in our example as done as long as not all child tasks are completed. This setting alone does not enforce a special sequence of task completion, for that you still need the “ORDERED” property.

org-track-ordered-property-with-tag

is used to create a special tag for the tasks that have the “ORDERED” property set to “t”. Usually the PROPERTIES drawer is collapsed, so you don’t see if there is an “ORDERED” property inside. If you set this variable to “t” (true) then tasks with an active “ORDERED” property will get a tag named “ORDERED”. You can also set the variable to a string that then will be used to name the tag. If the variable is set to “nil” then no tag will be assigned.

org-agenda-dim-blocked-tasks

If set to “t” (true) it will grey out tasks in the agenda view that cannot be marked as DONE at the moment, either because another task needs to be completed first because of an “ORDERED” property or the task is a parent task and at least one of the child tasks is not yet completed.

org-enforce-todo-checkbox-dependencies

If set to “t” (true) you cannot mark a task as “done” when this task has checklist and one of the checkbox items is still open.

5.2 Timers

Learning goals for this lesson

You will learn how to timers that count up or down.

Countdown timer

One function that I really frequently use is the countdown timer. When I do my weekly review I have 2 checkpoints that say “do a quick brainstorming for 5 minutes and write down new ideas”. But how can I tell when 5 minutes are over.

Org-mode offers help. You can always start a countdown timer. The key combination to start a countdown timer is `Ctrl+c` `Ctrl+x` `;`. If you don’t prefix this command then you will be prompted for the duration of the countdown, in the line at the bottom you will see “How much time left? (minutes or h:mm:ss)”. But you can also use prefixing to specify the countdown time.

So if I need a timer for 5 minutes I press `Ctrl+u` `5` `Ctrl+c` `Ctrl+x` `;` and the timer for 5 minutes will start. You can see the remaining time in

the status bar and if the timer is finished it will disappear from the status bar and you get a notification on your screen.

Start a timer that counts up

Sometimes you need a timer that just measures the time since you have started it. In this case you use the key combination **Ctrl**+**c** **Ctrl**+**x** **0** (number zero, not a capital “O”). If you prefix this with **Ctrl**+**u** then you will be prompted for an offset for this timer.

Pause timers, restart or stop them

If you want to pause a timer you just press **Ctrl**+**c** **Ctrl**+**x** **,** and the timer will stop. If you use this key combination again the timer will resume.

If you want to stop a timer you prefix this with **Ctrl**+**u**. So **Ctrl**+**u** **Ctrl**+**c** **Ctrl**+**x** **,** will stop your timer.

Inserting timestamps

One use of the relative timer (the one that counts up) is that you write down notes on something that happens. Imagine you watch a soccer match and you want to take notes on what happens. So when the 1st half starts you start your relative timer.

You have 2 types of timestamps that you can use:

Simple timestamp

A simple timestamp just inserts the current timer value in the form H:MM:SS into your text at the cursor position. You can insert this timestamp with **Ctrl**+**c** **Ctrl**+**x** **.** into your text.

Description timestamp

A description timestamp inserts a new line starting with a few spaces and then - H:MM:SS :: and puts your cursor after the two colons. You can insert this type of timestamp with **Ctrl**+**c** **Ctrl**+**x** **-** into your text.

So imagine in our soccer match team A scored a goal. Then you insert a description timestamp and write this down. The result could look like this:

```
- 0:09:22 :: Goal for the home team, scored by John Doe
```

If you feel confused by all the key combinations then you can also use the menu under **Org** » **Dates and Scheduling** where you can find the timer commands at the bottom of the sub menu.

5.3 Clocking

Learning goals for this lesson

You will learn how use clocking to track how long you worked on a task.

Why do I want to use clocking?

There are a lot of reasons for tracking the time you work on tasks:

- When I was employed and working on several projects “in parallel” I had to fill out a time sheet every month to report how many hours I worked on which project.
- When you are doing client work and your work time is billable you might need some real records on the time you spend for that client.
- Maybe you are just curious to find out how much time you spend on the things you are doing.

What does clocking mean?

The basic metaphor is that you “clock in” like the workers in former times when they had their punch card to put a timestamp on that card when they started working and when you are finished you “clock out” which means another timestamp so you can computer the time you spend in between.

Orgmode creates entries every time you “clock in” and then “clock out”. This is stored as a entry with the keyword “CLOCK:” and looks like this:

```
CLOCK: [2020-06-25 Do 20:20]--[2020-06-25 Do 20:22] => 0:02
```

So you see I clocked in on 20:20 and clocked out on 20:22 on June 25, 2020. That make 2 minutes for our example.

Customize where clock information goes

Those entries will by default show up in you LOGBOOK drawer. If you want them in an extra drawer you can customize the variable *org-clock-into-drawer* and set this to a name like “CLOCKING”. Then all clock information will be written into a drawer called CLOCKING

Key combinations

To “clock in” place your cursor on a task and press **Ctrl**+**c** **Ctrl**+**x** **Ctrl**+**i** (“i” like clock in).

To “clock out” you just need to press **Ctrl**+**c** **Ctrl**+**x** **Ctrl**+**o** (“o” like clock out).

If you accidentally clocked in at the wrong task you can cancel the clock with the key combination **Ctrl**+**c** **Ctrl**+**x** **Ctrl**+**q**. This will delete the current clock from the CLOCKING drawer.

If you want to put your cursor on the task that is currently clocked, then press **Ctrl**+**c** **Ctrl**+**x** **Ctrl**+**j** (“j” for **j**ump to current clocked task).

If you practice the Pomodoro technique then this means that you work for lets say 25 minutes with full focus and then do a small break to maybe get a coffee or just get up and walk around for a short time.

```

*** TODO Clocking (aka time tracking)..... 0:10
SCHEDULED: <2020-06-25 Do>
:PROPERTIES:...
:CLOCKING:
CLOCK: [2020-06-25 Do 20:37]--[2020-06-25 Do 20:39] => 0:02
CLOCK: [2020-06-25 Do 20:27]--[2020-06-25 Do 20:33] => 0:06
CLOCK: [2020-06-25 Do 20:20]--[2020-06-25 Do 20:22] => 0:02
:END:
:LOGBOOK:...
- Only one timer at a time

```

Figure 5.1: Example for a clock summary display

So you clocked in at the start and clocked out after 25 minutes when your little break starts. After the break you want to work on that task again. In this case you can just press `Ctrl+c` `Ctrl+x` `Ctrl+x` and the clock will start again on the task that was last clocked.

Sometimes you switch tasks during the day, but practically spend time slices on just a few tasks. then you can prefix the clock restart command above to get a menu of the last tasks that were being clocked so you can easily select them. The full key combination for that is `Ctrl+u` `Ctrl+c` `Ctrl+x` `Ctrl+x`.

Show me the data

After you clocked your tasks for lets say a few days or a week you want to just see how much time some task already accumulated. In this case you press `Ctrl+c` `Ctrl+x` `Ctrl+d`. You will get a grayed overlay that shows the sum of all clock entries ¹ for a task.

In figure 5.1 you see how this works ² for my task to write this part of the book. I just have created some 3 random clock entries and after pressing `Ctrl+c` `Ctrl+x` `Ctrl+d` Org-mode will show me that I have 10 minutes in total worked on this task.

If you want to quit displaying the clocking information just type `Ctrl+c` `Ctrl+c`. Then the display of the clock information will be removed.

Also keep in mind, as mentioned in figure 5.1 that since your brain is a single core CPU only **one** clock can run at any given time. You cannot clock 2 tasks in parallel as well as you can not work on 2 things at the same time.

If a task is clocked already and you “clock in” on another task that means your previous task will be automatically “clocked out” and the clock starts on the current task that you last “clocked in”.

5.4 Column view

Learning goals for this lesson

You will learn how to view your Org-file in a different way.

¹In section 8.1 you will learn about clocktables that provide a great way to get detailed clocking information.

²Timestamps in this example show “Do” which is the abbreviation of the German word for “Donnerstag” or “Thursday” in English, so don’t get confused.

Why do I want to use column view?

Sometimes you want to have a special view on your Org-file. Column view offers to see the file in a sort of table structure.

The rows of this table are your headlines. The columns of the column view can be

- The headline of your item.
- The ToDo keyword of your item.
- The tags of that headline.
- The sum of all clock entries.
- User defined properties from the `PROPERTIES` drawer.

Defining columns for a column view

The definition can be done on a global level or a subtree level.

Global definitions are put at the start of the file with a line like this:

```
#+COLUMNS: %58ITEM(Task) %7TODO %6CLOCKSUM(Clock)
```

The number after the “%” character specifies the width of the column. After that there comes the keyword to specify what you want to see. The text between “(“ and “)” defines the header of that column in column view.

The column definition on a subtree level has almost the same syntax. The difference is that now you have to define a property called `COLUMNS` inside the `PROPERTIES` drawer. This defines the columns for the subtree below the headline to which that `PROPERTIES` drawer belongs.

Working with column view

To switch to column view use the key combination `Ctrl+c` `Ctrl+x` `Ctrl+c`. If the cursor is located before the first level 1 headline then column view is applied to the complete file. If the cursor is inside a subtree, then the column view is applied to this subtree only.

If you want to leave column view and switch back to the normal view, then press `q` to quit column view. Note that column view is a so called overlay, so you don’t have to fear that part of your Org-file will be stored in column view when you save it while column view is active.

In our example above we used the special item `CLOCKSUM` which shows the sum of all clock information for the item as described in section 5.3. If you want to sum only the clock items of the current day, then you can use `CLOCKSUM_T`.

Remember the template of our book reading list from section 4.4? Here you could create a special column view for the subtree which has columns for the user defined properties “Author” and “Title”. You could even create more properties like “Numpages” for the number of pages or “Genre” for the genre of that book. All that info then can be displayed in a table form with column view.

5.5 Effort estimates

Learning goals for this lesson

You will learn how to plan the estimated time you need for a task.

Better planning with efforts

So far we have created tasks in our system that we want to do, but there is not much information in the system how long this task will need to be completed. Some tasks might be short like “Check the oil in the cars engine”, others will take long like “Write a book”.

So it would be helpful to see how much time you have planned for a certain task. Org-mode offers a special property named **EFFORT** to keep track of you estimates.

Working with the **EFFORT** property

The simplest way to use the **EFFORT** property would be that you open your **PROPERTIES** drawer and add that property with an effort. Example:

```
:PROPERTIES:
:EFFORT:    0:10
:END:
```

In this example we defined an effort of 10 Minutes for a task.

A quicker and easier way is to put a definition line at the start of your Org-file that says:

```
#+PROPERTY: Effort_ALL 0:10 0:20 0:30 1:00 2:00 4:00 6:00 8:00
```

This creates a set of predefined effort estimates between 10 minutes and 8 hours³ in reasonable steps.

Now you also create a column view (see section 5.4) that might look like this:

```
#+COLUMNS: %58ITEM(Details) %8Effort(Time){:} %6CLOCKSUM(Clock)
```

So we have 3 columns, one with the task description, one with the effort and one with the sum of the clocking information. The “{:}” in the column definition means that you sum up the time. So if your column view is on a subtree level the parent line will get the sum of all child lines.

Now you switch to column view with **Ctrl**+**c** **Ctrl**+**x** **Ctrl**+**c** and you will probably see your task description and an empty column for “Time” and also an empty column for “Clock”.

³8 hours should be more than enough. If you have tasks that take several hours you should think about converting them into a project and create smaller subtasks.

Now place your cursor in the Time column. With $\uparrow + \rightarrow$ you can step forward through your predefined effort estimates, with $\uparrow + \leftarrow$ you step backwards through your predefined values.

That is a quick way to set rough effort estimates.

Since we have also a “Clock” column let us set a task to 10 minutes and then “clock-in” that task as we have learned it in section 5.3.

Press $\text{Ctrl} + \text{c}$ $\text{Ctrl} + \text{x}$ $\text{Ctrl} + \text{i}$ to start the clocking.

You will notice that now in your status bar the display is “(0:00/0:10)” which means that on the left side you see the current clock value and on the right side you see your effort estimate.

If your clock exceeds the effort estimate for that job you get a notification that says “The task should be finished by now” and the clock counter in your status line gets a red background.

Other uses of effort estimates

Just imagine you find yourself in a situation that you have 10 minutes of time, but instead of doing nothing you want to spend it productively. So in this case you could open an agenda view and use m to match for the effort property. With a match string like “EFFORT=“0:10” you will get an agenda view with all tasks that have an effort estimate of 10 minutes. So you can pick one and do that.

Chapter 6

Linking, attachments and more

6.1 Linking (internal)

Learning goals for this lesson

You will learn how to link to other items in your current Org-file.

What are internal links

Everyone that uses the World Wide Web should know about links. A link is a pointer to another location in the Internet. Orgmode also knows links, both internal (what we discuss in this section) and also external (which we will discuss later).

What are possible targets for internal links?

Since a link is a pointer to a destination, we need to know what targets we can use to link to.

A headline

Every headline in your Org-file can serve as a target for internal links.

Named targets

You can define a named target with a line that starts with `#+NAME:` and then names the target after the colon. Another way is to define targets with `«target»`, which seems to work better than the usage of `#+NAME:.`

Custom IDs

You can also create a property `CUSTOM_ID` with an unique (its up to you to make sure that it is unique) identifier.

Radio Targets

Radio targets are a bit special. You put them in a place in your file and

every other occurrence of the target string is automatically transformed into a link to that radio target.

How do links look like?

Since Org-mode is all in plain text you can enter links as text. Every link starts with two square left brackets. Then you name the target. If you don't want a link description then close the link with two right square brackets, if you want to add a description then just use one right square bracket, another square left bracket and then write the description and close it with two right square brackets.

Examples:

- This links to `[[a headline]]`
- This links to `[[Name target][Named target with a description text]]`
- This links to `[[#MyID123]]` an item with that CUSTOM_ID.

Radio targets

A radio target looks somewhat different:

```
<<<radio_target>>>
```

Radio targets are “activated” with `Ctrl+c` `Ctrl+c`. Once they are activated every occurrence of the target string will transform into a link to that target.

Key combinations

You can edit links with `Ctrl+c` `Ctrl+l` (lower case “L” for link).

You can follow a link with `Ctrl+c` `Ctrl+o` (like open link).

You can return to where you came from with `Ctrl+c` `&`.

6.2 Linking (external)

Learning goals for this lesson

You will learn how to link to external locations in your current Org-file.

What are external links

The main difference between internal and external links is that the external link points to something outside the current Org-file. That can be an item in another Org-file or just a location on the web.

The syntax in this case is that you have to provide a target that is usually in the form `protocol:location`. So if I would like to link to my blog the link would be typed like this:

```
[[http://koenig-haunstetten.de] [Rainer's Blog]]
```

The link in Org-mode would be shown as “Rainer’s Blog” and point to my blog.

Link to unique IDs

In section 6.1 we learned about the `CUSTOM_ID` property. The “problem” with that is that we had to ensure that all IDs were unique.

Orgmode also has the `ID` property. This is generated with the function `org-id-get-create` and creates a 25-digit UUID for the item that has the cursor.

If an item has an `ID` property you can link to it by using the target `ID:25-digit-UUID`. For this type of link it is not necessary that the target is in the same file as the link. Emacs keeps track of what UUID is stored in what Org-file. The location of that housekeeping file is in the variable `org-id-locations-file`. Usually you don’t need to customize this, the file will be a hidden file usually in your “`/.emacs.d`” directory. But imagine you want to use Orgmode on several computers and you synchronize them with lets say Dropbox¹, then its a good idea to put this file also into your Dropbox directory so that it gets synchronized as well.

How do I create an ID

If you want to create an ID for any item place your cursor on that item and then type `[Alt]+[x]org-id-get-create[Enter]` and your `PROPERTIES` drawer will then get the ID. Example:

```
:PROPERTIES:
:ID:         63428fb9-4452-48d4-890f-ceae609aa623
:END:
```

Automate the ID creation

Yes, it will be boring to always have to type `[Alt]+[x]org-id-get-create[Enter]` but hey, the great advantage of Emacs and Open Source is that you can add some lines of code and make things much easier. We have 2 possible approaches that can be put into the Emacs startup file:

Adding IDs to all headlines when saving the file

One method is to define a small function that automatically adds the `ID` property with an unique ID to all headlines that have none when you save your current Org-file.

The code for this approach looks like this:

¹Dropbox is a cloud storage service where you define a directory that then is kept in sync over all computers connected to your Dropbox account.

```
(defun my/org-add-ids-to-headlines-in-file ()
  "Add ID properties to all headlines in the current file which
  do not already have one."
  (interactive)
  (org-map-entries 'org-id-get-create))

(add-hook 'org-mode-hook
  (lambda ()
    (add-hook 'before-save-hook
      'my/org-add-ids-to-headlines-in-file nil 'local)))
```

Explanation is simple. On top we define a function that calls our *org-id-get-create* function for all headlines. That means, if a headline already has an ID property nothing will happen, if it has none then a new UUID will be created.

Then we add this function to the *before-save-hook* which is executed every time we save our Org-file. So in this case before saving we add IDs to all headlines.

The problem with this way of automation is that it will work on **all** your Org-files, so imagine the journal with the datetree, that is probably a file where you don't need IDs. But you can't turn it off for that file, so every journal entry will have at least 3 additional lines for the **PROPERTIES** drawer.

Create an ID and copy the UUID to the clipboard

This approach defines a function that will be bound to a function key. When pressing this key the item where the cursor is will be treated with the *org-id-get-create* function which as explained above will change existing IDs, but create new ones if an item has no ID yet. Then the 25-digit UUID will be copied to your clipboard. This means, if you need to construct a link to this item, then you have the UUID already present to be pasted into your link target.

The code looks like this:

```
(defun my/copy-id-to-clipboard() "Copy the ID property value
  to killring, if no ID is there then create a new unique ID.
  This function works only in org-mode buffers.
```

The purpose of this function is to easily construct id:-links to org-mode items. If its assigned to a key it saves you marking the text and copying to the killring."

```
(interactive)
(when (eq major-mode 'org-mode) ; do this only in org-mode buffers
  (setq mytmpid (funcall 'org-id-get-create))
  (kill-new mytmpid)
  (message "Copied %s to killring (clipboard)" mytmpid)
  ))

(global-set-key (kbd "<f5>") 'my/copy-id-to-clipboard)
```

The function is bound to the **F5** key with the last line. The function will only work inside org-buffers, so if you for example work with Emacs on a simple

text file and press `F5` nothing will happen.

6.3 Attachments

Learning goals for this lesson

You will learn how to add attachments to your items. Attachments are external files on your hard disk.

Important: The attachment handling has been re-factored in Org version 9.3 so be sure to have at least version 9.3 installed for this lesson.

Where do attachments go?

Attachments will be stored on your computers hard-disk. You have two possibilities to specify where attachments should end up:

Use ID property

If the node that gets a file attached has an ID property then your file will be stored under a directory that is named after the UUID assigned to this item. You can customize the variable *org-attach-id-dir* to set a path to where those ID related directories should be created. The default setting for this variable is “data/”, so your attachments will be stored in directory “data/*UUID*/”.

Use DIR property

If the note that gets a file attached has an property named *DIR* then this will specify the directory where attachments are stored. The DIR setting has precedence over the ID setting, so if an entry has both ID and DIR properties, then DIR will win.

Be careful, if you set DIR to something static like “attachments” then every node with this DIR setting will see all attachments in this directory. So when you want to have an attachment structure that is closely bound to your items, you should go with the ID method.

Invoking Org Attach

If you place your cursor on an item and press `Ctrl + c` `Ctrl + a` you will see the Org attach menu. On top of the menu you probably see something like that:

```
Attachment folder:
/home/joe/org/data/de/d63a31-bb34-4352-8388-58051dfc1fa7
(Not yet created)
```

You may notice that the “data” directory also has a subdirectory de. This are the first to digits of the UUID, the complete UUID used for this example was:

```
:PROPERTIES:
:ID:          ded63a31-bb34-4352-8388-58051dfc1fa7
:END:
```

Attaching files

Next you see a list of keys you can use to attach files:

a

You will be prompted for a file and then the file will be attached by using the method configured in the variable *org-attach-method*. You need to customize this variable to set your preferred attach method or you use one of the other command keys.

c

You will be prompted for a file and this file will then be **copied** to the attachment directory.

m

You will be prompted for a file and this file will then be **moved** to the attachment directory. **Caution:** Move means it will be gone at its original location!

l

You will be prompted for a file and this file will then be **hard-linked** to a similar file in the attachment directory. This will only work if the file is stored on the same filesystem as your attachment directory. You cannot create hard links over filesystem borders.

y

You will be prompted for a file and then a **symbolic link** to this file will be created in the attachment directory.

u

You will be prompted for an URL and then the file from that URL will be downloaded and stored in the attachment directory. **Note:** This will only work for download URLs like <https://example.org/download/file.pdf>, but if you for example would like to download from my Wordpress blog with and URL like <https://koenig-haunstetten.de/download/11/> you will get an error message because the URL does not end with a filename. So be careful when you use this method.

b

You will be prompted for an active buffer in your Emacs session. The current contents of this buffer will then be stored in the attachment directory. Note that if you change the buffer later the changes will probably not end up in the file that was stored in the attachment directory.

n

You will be prompted for the filename of a new attachment and then you end up in an empty buffer where you can write that attachment and save it to the attachment directory.

z

If you manually added files to the attachment directory then you can now synchronize this with the current node. Note that in my tests I was able to see the attachments immediately even without that synching.

Viewing attachments

The next part of command keys is for viewing your attachments and the attachment directory:

o

Opens the attachment(s). If only one attachment is attached it will open immediately, otherwise you will be prompted for the name of the attachment. Opening can mean, that an external viewer can be invoked.

↑ + o

Opens the attachment(s) forced in Emacs. If more than one attachment exists you will be prompted for a file name.

f

Opens a view to the node attachment directory.

↑ + f

Force open the attachment directory using Dired in Emacs.

Once an item has an attachment the item will also get a tag named `:ATTACH:.`

Deleting attachments

The next command block is for deleting attachments:

d

Deletes **one** attachment. You will be prompted for the filename.

↑ + d

Deletes **all** attachments. You will be prompted if you are sure to delete all attachments. A probably safer way is to open the attachment directory in Dired and remove the the files there.

Admin commands

The last block is for some admin stuff:

s

Sets a specific attachment directory for this entry. The `DIR` property is set.

↑ + s

Unsets the `DIR` property. In my case also the attachment directory got removed when I did this, so be careful.

q

Abort Org Attach if you accidentally invoked it.

Attachment links

One problem in viewing attachments is that you always have to invoke Org Attach to see what is there. A solution for this problem are attachment links. They have the form:

```
[[attachment:file.txt][description]]
```

So you can place a direct link to your attachment in your Org-file which makes accessing the attachment much easier.

Attachment inheritance

The Org-mode manual has some stuff on attachment inheritance, but honestly I had problems to reproduce it. Simple example:

```
* Orgmode Course
** TODO Write section attachments          :ATTACH:
   :PROPERTIES:
   :ID:         ded63a31-bb34-4352-8388-58051dfc1fa7
   :END:
*** TODO Record video
   :PROPERTIES:
   :ID:         6e813555-6802-4419-aa4c-32665e397a20
   :END:
```

In this setting attachments should be inherited between “Write section on attachments” and “Record video”. To make this work you need to customize two variables:

org-attach-use-inheritance

This determines if inheritance should be used or not. Problem is that the customization interface shows it as “on” and you can toggle between “on” and “off” but there seems to be a third possible value “selective” which needs the next variable.

org-use-property-inheritance

This variable needs to be set to “t”, if its “nil” inheritance did not work, even if the variable *org-attach-use-inheritance* was set to “t”.

Be careful if you are using attachment inheritance, this also means, that adding attachments to any child will store them in the attachment directory of the parent.

6.4 Priorities

Learning goals for this lesson

You will learn which tools Org-mode offers to define priorities for your tasks.



Figure 6.1: Eisenhower's priority matrix

Some thoughts on priorities

Before we look at what tools Org-mode provides to manage priorities let us have a look at priorities in general. There is that quote from Karen Martin that says “*When everything is a priority, nothing is a priority*”. This should remind us that priorities will not automatically make us happy and our lives easier. We tend to prioritize things to quickly see... what? If you say “highest priority” what does that mean. Is it the most important thing or the most urgent.

One way to look at it could be the “Eisenhower priority matrix”.

When you look at figure 6.1 you see that this matrix has 2 dimensions, **urgency** and **importance**. The matrix gives also some guidelines on how to treat those tasks.

- A task from quadrant 1 (high importance, high urgency) should be done right now. Don't procrastinate it.
- A task from quadrant 2 (high importance, low urgency) should be planed, decide when to do that.
- A task from quadrant 3 (low importance, but high urgency) should still be done immediately, but you can think if you can delegate this task to someone else.
- A task from quadrant 4 (low importance, low urgency) is a task that you really should consider to dump. One of the basic rules of personal productivity is that it is not only important to do the things right, but also to do the right things. A task from this quadrant is hardly a “right thing”.

Now let us be honest and serious. If you have a well maintained Org-mode system, if you do your weekly review and capture everything, then the chances are high that you don't need extra priorities. Yes, I said at the beginning that the purpose of the system is that you brain can forget about your tasks because they are in the system. But really, if something is important or urgent you will

still remember and know this. And if you apply schedules and deadlines (see section 2.4) then you have taken care of the tasks in quadrant 3 already before they boil towards “urgent” and you are still able to “act” instead of “react”.

I’m using Org-mode now for several years now, I feel that I’m very productive and always in control, and I really don’t use priorities.

But if you still want to put priorities on items, here we go.

The standard priorities

Orgmode offers either “no priority” or a set of 3 priorities named “A”, “B”, and “C”.

You can cycle through those priorities when you place the cursor on a headline and press `⏶+↑` or `⏶+↓`. You can also directly set the priority with the key combination `Ctrl+c`, which will prompt you then for the priority to use.

The current priority will be displayed in square brackets at the start of the headline or directly after the `ToDo` keyword. Example:

```
* Priority example
** TODO [#B] Do with medium priority
** TODO Do with no priority
** TODO [#A] Do with high priority
```

That is how priorities will be represented in your Org-file. If you create an Agenda view on this subtree, the agenda view will look like this:

```
private:  TODO [#A] Do with high priority
private:  TODO [#B] Do with medium priority
private:  TODO Do with no priority
```

So in our agenda view we see the highest priority on top, and tasks with no priority setting on the bottom.

Custom priority definitions

If you don’t like the “A”, “B” and “C” scheme for priorities and want to set a range lets say from 1 to 5 you can write a line at the beginning of your Org-file that says:

```
#+PRIORITIES: 1 5 3
```

The tree numbers mean “highest” (1) “lowest” (5) and “default” (3) priority. The “default” is the value that is displayed when an item has not yet a priority and you start cycling through the priorities with `⏶+↑` or `⏶+↓`.

6.5 Tables

Learning goals for this lesson

You will learn about the table editor inside Org-mode.

What are tables

Org-mode supports basic text tables, that are arranged like a spreadsheet. You can present some data in table format, and unlike column view (see section 5.4) it's not just an overlay but real text in your file.

Tables can be used to organize reference material like inventory lists.

Structure of a table

A table consists of rows separated by “|” and columns. Example:

Name	Year of birth	Age
Rainer	1961	
Joe	1995	
Mike	1937	

Basic table editor commands

When entering a table you will see, that the column width gets automatically adjusted to the width of the content of this column.

If you need a horizontal line then type “|-” and press `↩`.

If you want to move columns left or right you can do this with `Alt+←` or `Alt+→`.

If you want to delete a column position the cursor in it and then press `Alt+↑+←`.

If you want to insert a column right of the current column then press `Alt+↑+→`.

If you want a new horizontal line press `Ctrl+c -`.

If you want to move rows up or down use `Alt+↑` or `Alt+↓`.

The Org-mode table editor has some additional features that you can explore under the `Tbl` menu.

Table formulas

Wouldn't it be nice if Org-mode could calculate the Age in the table above? Yes, this is possible. We add some more things to our table:

```
#+CONSTANTS: year=2020
| Name | Year of birth | Age |
|-----+-----+-----|
| Rainer | 1961 | |
| Joe | 1995 | |
| Mike | 1937 | |
#+TBLFM: $3=$year-$2
```

It is **important** that you don't have empty lines. On top of the table we define a constant named “year” with the value 2020. Constants always start with “#+CONSTANTS:”.

On the bottom we define a formula. Table formulas start with “#+TBLFM:”. In our example the formula says: The value of column 3 is the value of “year” (our constant above) and subtracted by value of column 2. Columns are counted 1..n from the left and are referenced with “\$”. If you would need to reference a row then you should use “@”.

So now when we place our cursor on the line with “#+TBLFM:” and press `Ctrl`+`c` `Ctrl`+`c` the formula gets applied to the table.

```
#+CONSTANTS: year=2020
| Name      | Year of birth | Age |
|-----+-----+-----|
| Rainer    |          1961 |  59 |
| Joe       |          1995 |  25 |
| Mike      |          1937 |  83 |
#+TBLFM: $3=$year-$2
```

Of course the tables in Org-mode are not a fully replacement for spreadsheet programs like LibreOffice Calc, but for simple formulas it works as well.

Chapter 7

Exporting and publishing

7.1 Exporting

Learning goals for this lesson

You will learn how to export your files in HTML or PDF format.

How do I export files?

For exporting you have to type `Ctrl + c` `Ctrl + e` to invoke the export dispatcher.

Figure 7.1 shows what export possibilities you have. For this lesson we will only focus on HTML and PDF export.

Exporting to HTML

Figure 7.1 shows what you see after pressing `Ctrl + c` `Ctrl + e`. The next possible keys you can press are highlighted in red. For HTML you have to press the `h` key.

Then you have 3 options:

`↑` + `h`

Create an HTML buffer.

`h`

Create an HTML file. The filename will be the same as for your Org file, but the extension will be “.html” instead of “.org”.

`o`

Create a file and open it.

Exporting to PDF

Exporting to PDF is done by invoking `LATEX`, so you should have a working `LATEX` installation on your machine.

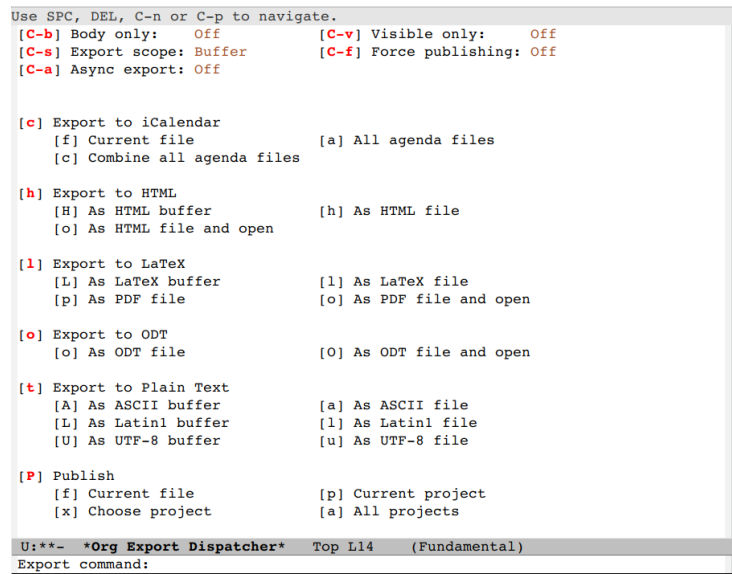


Figure 7.1: Org Export Dispatcher

For $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ export you first have to press **I** and then you have the following options:

↑ + **L**

Creates a $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ buffer.

I

Creates a $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ file.

P

Creates a PDF file with the same name as your Org-file but with the extension “.pdf” instead of “.org”.

o

Creates a PDF file and opens it.

Customizing things

You can adjust what gets exported by adding an line with options like this:

```
#+OPTIONS: d:t \n:t p:t todo:t
```

Explanation of the options:

d:t

Export also the contents of drawers like LOGBOOK.

\n:t

Preserve line breaks in the exported format.

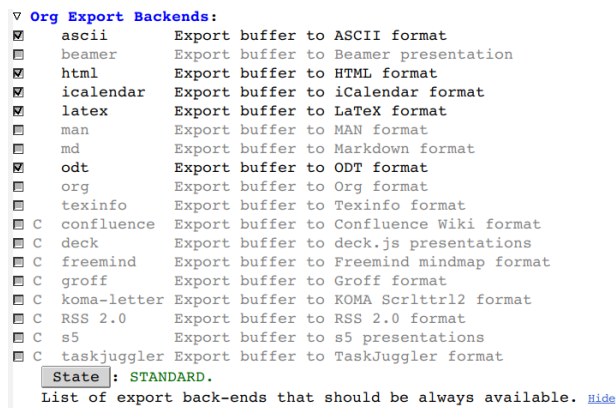


Figure 7.2: Customizing org-export-backends

p:t

Export planning information, so the timestamps for SCHEDULE and DEADLINE are shown in the exported file.

todo:t

Include todo keywords in the exported text.

All export functions offer an option “export and open”. Normally those files are opened in Emacs. If you want to use another program to view the exported files you have to customize the variable *org-file-apps*. Here you can for example specify that you want to use Evince as a PDF viewer. That means you have to switch the viewer for PDF files to “command” and use a command like “evince %s”. The “%s” is needed to hand over the filename of the file to be viewed.

7.2 Advanced exporting

Learning goals for this lesson

You will learn how to export your files in more formats and with more options.

Enabling more export formats

To enable additional export formats you have to customize the variable *org-export-backends*.

Figure 7.2 shows what backends are available. The backends with a “C” after the checkbox are “community backends” and not part of the “official” Org-mode distribution.

Let us just for fun enable the beamer backend. Click on the checkbox for the beamer backend so that it is “checked” and then click on “Apply & save”.

Use the beamer backend

So you want to make your next presentation on some stuff, but you are sick of PowerPoint and want to prepare the presentation with Org-mode? Let us create an Org-file for that purpose, let us name it “presentation.org”.

We start with some options for the export backend.

```
#+OPTIONS: toc:nil ^:nil tags:t f:t
#+AUTHOR: Rainer König
#+TITLE: Getting yourself organized with OrgMode
#+SUBTITLE: Advanced exporting
#+DESCRIPTION: More advanced exporting examples. Some more options to play around with.
#+BEAMER_THEME: Berlin
#+BEAMER_FONT_THEME: professionalfonts
```

The “#+OPTIONS: “ line we know already from section 7.1, now we define some new options

toc

Export table of contents. Since we have set it to “nil” no table of contents will be exported.

^

Enables or disables sub/superscripting

tags

Enables or disables exporting of tags.

f

Enables or disables exporting of footnotes. We will use this later so we enable it with “t”.

The other definitions are this:

#+AUTHOR:

Names the author of that presentation. That will be displayed in the slides.

#+TITLE:

Defines the title for the presentation.

#+SUBTITLE:

Defines a subtitle for the presentation.

#+DESCRIPTION:

This is text for your convenience, it will describe what the presentation is about, but not exported to the slides.

#+BEAMER_THEME:

Selects a theme for the beamer class.

#+BEAMER_FONT_THEME:

Selects a font for the beamer class.

Now let us write the contents of the first slide:

```
* Customize new backends          :CUSTOMIZE:
- Customoze group "org-export" and there "org-export-backends"
  + Enable "beamer" for LaTeX beamer support[fn:1]
  + Enable "odt" for export to OpenDocument Text
```

```
[fn:1] This file was created with beamer export.
```

The level 1 headline defines the title of that slide.

Then we have a “bullet” list with “-” and “+” which will be transformed to bullets with the correct indentation.

You also see a construct

fn : 1

that defines a footnote. The text for the footnote can be found at the end of the slide definition.

You can add a third slide with a table now:

```
* Useful options
```

```
Additional ##+OPTIONS:* (with "nil" or "t")
```

Option	What does it do?
-----+-----	-----
toc	Enable Table of contents
^	enable sub/superscripting
	(used it here for BEAMER_THEME)
tags	Export tags
f	Export footnotes (used on slide 2)
@@latex:\textbackslash n@@	Toggle line break preservation

More options can be found in the manual.

Those are the options. That very strange “@@latex:\textbackslash n@@” entry will later transform into “\n.”

So now let us export this to the beamer backend. If you press **Ctrl**+**c** **Ctrl**+**e** you will invoke the Org export dispatcher as we have learned already in section ???. But after enabling the beamer backend we got more options for the `LaTeX` exporter.

Some more keys are added under the “Export to LaTeX” option:

↑ + **b**

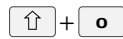
Exports the `LaTeX`code into a new Emacs buffer.

b

Exports the `LaTeX`code into a file.

↑ + **p**

Exports to a PDF file using the beamer class.



Exports to a PDF file using the beamer class and opens this file.

So if you press  +  your PDF viewer should start and display the presentation slides.

A final thought on the beamer backend

The beamer backend is basically transforming an outline document into slides, but it is lacking of design and images. So there is a good chance (or risk) that your presentation will be boring.

If you want to learn about how to make presentations that are not boring and will really impress your audience I highly recommend that you read the book “Presentation Zen” from Garr Reynolds[Rey08].

7.3 Publishing

Learning goals for this lesson

You will learn how to use Org-mode as a little CMS (Content Management System) for your web site.

Preparing the stage

For this lesson we setup a web sever on our local Linux machine and enable the “public_html” directory inside the user’s home directories.

On my openSUSE Leap 15.1 system I installed “nginx” with the following command:

```
# zypper install nginx
```

In the default configuration the “public_html” directories are not enabled. So you have to change the config file.

In my case I looked for the “server” definition that listens on port 80 and runs on localhost. There I added those lines:

```
location ~ ^/^(.+?)(/.*)?$ {
    alias /home/$1/public_html$2;
    index index.html index.htm;
    autoindex on;
}
```

Then I restarted the web server and now you can access HTML files with `http://localhost/~joe/` which will make nginx send the contents of the file `/home/joe/public_html/index.html` to your browser.

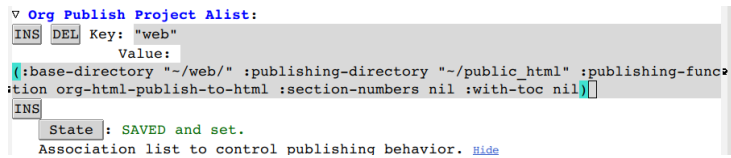


Figure 7.3: Customizing org-publish-project-alist

Creating some content

Next step is to create some really basic content for our experiment. First we create a directory “web” and inside that we create 2 files for the moment. The first file we call “index.org” and it should look like this:

```

#+TITLE: Welcome to my homepage
#+AUTHOR: Joe the test user

* My home page

** What do you find here
- [[file:about.org][Infos about me]]

```

The next file is called “about.org” and looks like this:

```

#+TITLE: Infos about me

* Infos about me

I am what I am.

```

Those are the 2 files we want to publish in this lesson. Publishing means, that the files should be converted to HTML, placed inside the “public_html” directory of our test user and the link from “index.html” should then point to “about.html”.

Describing our project

The next step is to “tell” Org-mode about our project. For that we have to customize the variable *org-publish-project-alist* which is a list of project definition.

When you call the customization editor it can happen, that you can’t find this option on the first shot. If this is the case invoke the export dispatcher with **Ctrl**+**c** **Ctrl**+**e**. After that the option should be ready to customize and you might see something like figure 7.3. This is a simple list that consists of a key (in our case “web”) and a set of properties that describe the project. Every property begins with a colon and the property name, then the value. We used the following properties:

base-directory

describes where your source files for the publishing project are located. In our case it is the directory “web” inside the users home directory.

publishing-directory

describes where the exported files should end up. In our case it is “public_html” inside the users directory.

publishing-function

describes which function should be used to transform the Org-files to HTML. We use “org-html-publish-to-html”.

section-numbers

defines if section numbers should be exported to HTML. We use “nil”, so no section numbers

with-toc

defines if we need a table of contents or not. We use “nil”, so no table of contents in our example.

Publish the project

Now it is time to do the publishing. First we have to invoke the Org export dispatcher by pressing `Ctrl+c` `Ctrl+e`. If you have many export backends enabled (see section 7.2) then there is a good chance that the list is too long and you don’t see the publishing options. In that case press the (space bar) to scroll the menu down.

The publishing commands are invoked with `↑+p`. Then you have 4 possibilities:

`f`

publishes the current file.

`p`

publishes the current project.

`x`

if you have several projects defined, then you can chose the *key* of the project that you want to publish.

`a`

publishes all projects.

Publishing timestamps

When you publish our project and then publish it again, you will see the following messages in the “Messages” buffer:

```
Skipping unmodified file /home/joe/web/about.org
Skipping unmodified file /home/joe/web/index.org
```

Org-mode tracks the timestamps of the source file that were used for publishing. So if the file was not modified since the last publishing action, then the file will be skipped, which means that only changed files will be published.

The location where those timestamps are saved is the directory “.org-timestamps” in the user’s home directory. So if you want to force Org-mode into publishing all files again, the easiest way is to delete the files in this directory.

Chapter 8

More advanced topics

8.1 Dynamic blocks

Learning goals for this lesson

You will learn how to

- Use a clocktable to get detailed clocking reports
- Use dynamic blocks to capture column views

Work with clocktables

Remember our clocking example from section 5.3. There we learned about a method to provide a quick overview of the accumulated time.

Now imagine yourself as a free-lancer that needs to write bills to his clients for the amount of time spent on their projects in the last month. In such a scenario I would recommend, that you have a separate Org-file for every client where you write down the tasks you're doing for them and of course, where you are clocking the time spent on their issues.

Inserting a clocktable

For our example we create a level 1 headline at the end of the client file and call it "Clocktable". Then we position the cursor in the next line and press `Ctrl` + `c` `Ctrl` + `x` `x` `clocktable` `↵`.

This will insert a new dynamic block with a clocktable that looks like this:

```
#+BEGIN: clocktable :scope subtree :maxlevel 2
#+CAPTION: Clock summary at [2020-07-23 Do 18:54]
| Headline      | Time      |
|-----+-----|
| *Total time* | *0:00*    |
#+END:
```

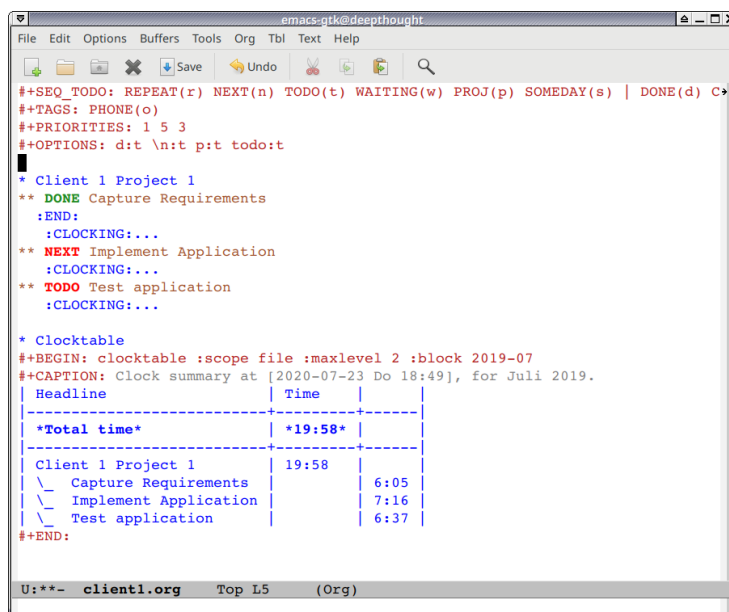


Figure 8.1: Example for a clocktable

The `#+BEGIN:` and `#+END:` mark the start and the end of the dynamic block. The word after `#+BEGIN:` describes what kind of dynamic block we have here, it is a “clocktable”. Then we have a list of properties:

:scope

describes the scope to consider. “subtree” is the default, but in our case we change it to “file” which means to look for all clocking entries in the current buffer.

:maxlevel

describes the maximum depth to which times are listed in the table. Clocks at deeper levels are summed into the upper level.

:block

Defines the time block to consider for this clocktable. In my example file I have clock entries from June and July 2019, so let us write “2019-07” for July 2019.

Updating the clocktable

After changing the begin line of our clocktable to

```
#+BEGIN: clocktable :scope file :maxlevel 2 :block 2019-07
```

we press `Ctrl+c` `Ctrl+c` to update the clock table.

In figure 8.1 we see the example clocktable for July¹ 2019. We see that we

¹Remember, my Emacs is running in a German environment, so you see “Juli” instead of “July”.

have spent 19:58 hours on “Client 1 Project 1” in July which is distributed to the 3 actions we have setup.

What are my options for time blocks

In the example above we have set the time to consider to “2019-07”. Now, if you place the cursor on the begin line of the block you can press `↑+←` or `↑+→` to change the month. to “2019-06” or “2019-08”. If you do this, then your clocktable will be immediately updated and will show the values for the new range.

The definition of `:block` can also use the following things:

'2020-07-23' a specific date.

'2019-07' a specific month.

'2020' a specific year.

'2020-Q2' 2nd quarter of 2020.

'2019-W48' ISO-week 48 of 2019.

'today', 'yesterday', 'today-N' a relative day.

'thisweek', 'lastweek', 'thisweek-N' a relative week.

'thismonth', 'lastmonth', 'thismonth-N' a relative month.

'thisyear', 'lastyear', 'thisyear-N' a relative year.

'untilnow' all clocked time ever.

If you don't define a `:block` property, then also all clock entries will be in the clock table.

Capturing column views

In section 5.4 you learned that the column view is an overlay on the text buffer. That also means, that you cannot export a columnview that is showed in your buffer because of that. But there is a way to add a dynamic block that holds the columnview as “normal” text, so that it can be exported.

Inserting a columnview

For our example we create a level 1 headline at the end of the client file and call it “Columnview”. Then we position the cursor in the next line and press `Ctrl+c` `Ctrl+x` `columnview` `Ctrl+j`.

Note: If Org-mode complains that there is “no such dynamic block: columnview” then press `Ctrl+c` `Ctrl+x` `Ctrl+c` once to enable column view and then leave that column view with `q`. After that the inserting of the dynamic block should work.

The next question you have ask is on the bottom of your emacs window: "Capture columns (local, global, entry with :ID: property) [local]:"

Here you define what to capture. If you placed the column view at the end of your file then type "global" which means that the whole buffer will be captured as column view.

Other values for the `:id` property of this dynamic block are:

'local' use the tree where the dynamic block is located.

'global' use the whole file with all headings.

'LABEL' define an unique ID for the ID property of an item.

'file:FILENAME' run columnview on top of FILENAME file.

You can also insert a property `:indent` and set it to `'t'` so that the hierarchy of the headlines is shown in the columnview dynamic block.

There is also a property `:hlines` with a number, which means that you get a horizontal line before each headline with a level smaller or equal that number.

Working with columnview dynamic block

When you want to update the dynamic block you press `Ctrl+c` `Ctrl+c`.

Since this columnview is "normal text" you can export it with the export functions.

Also note, that you can capture a column view from a totally different file by giving that filename as the `:id` property. Just remember the "track-read-books.org" archive file from section 3.5. Now I could create a temporary Org file and insert a dynamic columnview block like this:

```
#+BEGIN: columnview :hlines 1 :id "file:track-read-books.org"

#+END:
```

Now define a `#+COLUMNS:` header in that archive file. Then all you have to do is update the dynamic block to get a list of all the books you've read.

8.2 Tracking habits

Learning goals for this lesson

You will learn how Org-mode can help you to implement and track habits in your daily life.

What are habits?

Habits are recurring tasks that you do frequently. Examples:

- Brush your teeth every day

- Water your plants in the house every second day
- Go to the gym every week

Once a habit is *installed* you usually don't need a reminder to do them. Or do you need someone to remind you to brush your teeth every day?

How can Org-mode help me?

Orgmode has a module to help you track how good you are in doing your habits.

What do I need to track habits?

First you have to enable the *habits* module by customizing *org-modules*.





Your habits need to be tasks with a recurring ToDo keyword, e.g. "REPEAT".

The tasks need to have a property named **STYLE** with the value "habit".

The syntax for repetitions is a bit different from what we learned in section 2.5. Example:

```
* Habits
** REPEAT Practice Bass every 2nd or at least every 3rd day
   SCHEDULED: <2020-07-25 Sa .+2d/3d>
   :PROPERTIES:
   :STYLE: habit
   :END:
```

The repetition definition ".+2d/3d" means you plan to do the task every second day, or at least every 3 days.

If you track habits you need the completion timestamps for every iteration of your repeating task, so its is important, that logging of DONE state changes is enabled and you really log timestamps and don't kill the log entry with  +   + .

Habits tracker in action

Figure 8.2 shows the habits tracker in action. In the upper window you see the example habits task and a bunch of completion timestamps.

In the lower window you see the weekly agenda created on July 25 which is also the date that is scheduled for the habit.

Because we have set the **STYLE** property to "habit" you now see a colored bar in the agenda view for this task. Its overlaying the task description.

The "!" marks the current date in that bar. Every "*" marks a completed task on that day in the past (left of the "!").

The background color of the bar also has some meanings:

Blue

The task was to be done on that day.

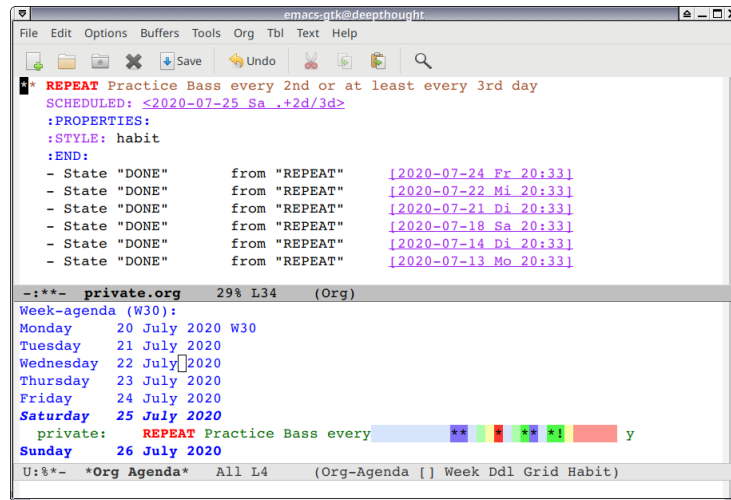


Figure 8.2: Example for tracking habits

Green

The task could have been done on that day.

Yellow

The task will be overdue the next day.

Red

The task is overdue on that day.

So if you look at the graph you see two "*" which represents July 13 and 14. Then on July 16 the background is green, indicating that the task could have been don an that day. July 17 is yellow, indicating that the task will be overdue on the next day. July 18 gets an "*", we did the task, but it was overdue anyway.

Additional notes

If you want to see that habit graph not only on the current day then you need to customize the variable *org-habit-show-habits-only-for-today*.

Habit tracking tracks just the successful repetition of your habit tasks. So if I'm doing my project to practice bass playing then I probably need other tasks that say "Practice lesson 1" and "Practice lesson 2" and so on.

Habit tracking relies on the recorded timestamps. Sometimes it will take a long while to install a habit, so you should shorten the list of the timestamps (delete old ones) from time to time. Just keep enough timestamps so that your bar graph is filled.

Once a habit is successfully installed you should delete the habit tracker task.

If you want to read more on habits I can recommend the books "The Power of Habit"[Duh13] or "Atomic habits"[Cle18].

8.3 Bulk agenda actions

Learning goals for this lesson

You will learn how to mark items in the agenda view and perform bulk actions on them.

Practical examples

The question that might come to your mind is “why shall I use bulk actions?”. So here are some examples how I use this feature.

In my case I use it during my weekly review. Usually I capture lots of things during the week and by using my capture templates those items all get a tag “:NEW:”. So all items with the “:NEW:” tag are items that I captured since the last weekly review. And at the start of my weekly review checklist there is a checklist item that says:

```
WR BOOTUP
=====
- [ ] Remove all NEW tags from the tasks in the backlog
```

So I have created an agenda view that shows me all items with that “:NEW:” tag. Then I mark all those items and do a bulk agenda action that says “remove tag”. This removes the tag from all items and I don’t need to browse through my file to do this manually on each item.

The other use is at the end of my weekly review. There is a checklist item that reminds me to archive all finished tasks. As you learned in section 3.5 you can have different archive files for every subtree. In my case I defined an agenda view that shows me all finished tasks. Then I mark all of them and do a bulk archive action, which means that all items will end up in their defined archive location. After that I do a quick `Ctrl`+`x` `Ctrl`+`s` in the now empty agenda view to force writing all modified buffers to disk.

How do I mark agenda items?

There are some keys to mark agenda items:

- `m` marks entry at cursor position.
- `u` un-marks item at cursor position.
- `*` mark all agenda entries.
- `↑` + `u` un-marks all agenda entries.
- `Alt` + `m` toggles mark at cursor position.
- `Alt` + `*` toggles all marks in the agenda.
- `%` mark entries based on a regular expression.

Marking entries based on a regular expression could be helpful if for example you have a lot of tasks that say “call xyz” and you want to attach a “:PHONE:” tag to every of those tasks. Then you could mark every entry with “call” as a regular expression and bulk assign the “:PHONE:” tag to them.

Executing bulk agenda actions

After you have marked your items in the agenda view you press **↑** + **b** to get a menu for the bulk agenda actions. You can also use prefixing by pressing **Ctrl** + **u** before you press the **b** key. The choices you now have are:

p

toggle between persistent and non persistent marks. The default is “non persistent”, that means after your bulk action the marked items will be unmarked. With the **p** you make the marks persistent, so that you can perform multiple bulk actions.

\$

Archive all marked entries to their archive files.

↑ + **a**

Archive all marked entries by moving them to their respective siblings.

t

Change ToDo keyword for the marked items.

+

Assign a tag to all marked items.

-

Remove a tag from all marked items.

s

Schedule all marked items to a specific date. You can also shift the dates by relative values like “++8d”. If you used **prefixing** for the bulk action then all schedule information will be removed from the marked items.

d

Set deadline for all marked item to a specific date. If you used **prefixing** for the bulk agenda action then all deadline information will be removed from the marked items.

r

Refile the marked entries.

↑ + **s**

Scatter the marked items over a given time period (usually 7 days). This might help for scheduling. If you used **prefixing** for the bulk agenda action then the tasks are scattered only across weekdays and weekends are excluded.

f

apply a function to the marked entries. See the Org-mode manual for details.

Some notes on the scatter function

If you are using the scatter function to reschedule tasks then the variable *org-log-reschedule* should not be configured to take notes, otherwise scattering is aborted with an error message after rescheduling the first entry.

8.4 Google calendar import

Learning goals for this lesson

In this lesson I will show you how you can make your Google calendar entries also show up in your Org-mode agenda views.

Calendar and Org-mode how do they coexist?

I'm using Google calendar frequently, every appointment is stored in my Google calendar. On the other hand, all my actionable items are stored in Org-mode, and when doing the planning it would be nice to see not only the list of tasks that are scheduled to be done, but also a list of appointments for a given day.

So my goal was to create a one way road from Google calendar to my Org-mode agenda view. Yes, I know that there are probable more sophisticated approaches to integrate Org-mode and Google calendar, but for my usage scenario I don't need to export from Org-mode to Calendar.

Both tools serve different purposes for me. Org-mode is the GTD system that helps me to be highly productive, the calendar is a tool to remind me about upcoming appointments and meetings. And yes, thanks to my smartphone and smart watch I have the calendar always with me, so reminders won't be missed.

Import from Google calendar

My approach is to use a cron job² for downloading all of my and my families Google calendars to ".ics" files. This cron job is performed every 2 hours. So in the worst case you need to wait 2 hours to get the latest additions to your Google calendar.

This is done in a Bash script. After the download is complete Emacs is called in batch mode with a Emacs Lisp script that imports the downloaded ".ics" files into a diary file. That diary file is then copied to the ".emacs.d" directory. This file is included in the Emacs "diary" file.

Org-mode needs to be customized. You need to set the variable *org-agenda-include-diary* to "t". Then all diary entries will show up in your Org-mode agenda views.

You also need to customize the variable *diary-list-entries-hook* in the diary group so that other files can be included in the diary.

²A job that is automatically executed on your system in a defined time interval.

The scripts

The download shell script

The following code comes from the down loader script. This is a script for the bash shell.

```
#!/bin/bash

WGET=/usr/bin/wget
DIARYDIR=~/.emacs.d/

mkdir -p /tmp/calimport
cd /tmp/calimport

# All URLs of calendars that we want to download
CALURL=( "https://calendar.google.com/calendar/ical/.../basic.ics" \
         "https://calendar.google.com/calendar/ical/.../basic.ics" \
         "https://calendar.google.com/calendar/ical/.../basic.ics" )

# Because every URL above ends in basic.ics
# we map those to the real calendar names
ICSNAME=( cal-main.ics cal-aux1.ics cal-aux2.ics )

# Get the calendar ICS files
for ((i=0;i<${#CALURL[@]};i++))
do
    $WGET -O ${ICSNAME[$i]} ${CALURL[$i]};
done

# create an empty diary file
rm diary-google
# Do the transformation into orgfiles
emacs --batch -l ~/bin/icsdiary.el

# copy all org files to your .emacs.d directory
cp diary-google $DIARYDIR

# Cleanup the mess that you did
cd /tmp
# rm -r /tmp/calimport
```

Note that I removed the private information from the Google calendar URLs and replaced them with “...” before the final “basic.ics”.

Since every calendar ends up with “basic.ics” we map the vector with the calendar URLs (CALURL) to a vector ICSNAME which holds the names of the downloaded files. So the first calendar maps to “cal-main.ics”, the second to “cal-aux1.ics” and the third to “cal-aux2.ics”.

Downloading is done with `wget` where we supply the real names with the “-O” parameter.

Then we call Emacs in batch mode with the `icsdiary.el` script. This creates a diary file named “diary-google” in our working directory (which we created in the “tmp” filesystem). We copy this to “emacs.d” and cleanup the “tmp” directory that we used.

The Emacs batch script

The “icsdiary.el” script is simple and straightforward:

```
(icalendar-import-file "/tmp/calimport/cal-main.ics"
"/tmp/calimport/diary-google")

(icalendar-import-file "/tmp/calimport/cal-aux1.ics"
"/tmp/calimport/diary-google")

(icalendar-import-file "/tmp/calimport/cal-aux2.ics"
"/tmp/calimport/diary-google")
```

We take every of our downloaded “.ics” files and append it to the “diary-google” file by using the function *icalendar-import-file*.

The cron tab entry

The cron tab entry looks like this:

```
# Import Google calendars to diary file
0 */2 * * * $HOME/bin/importcalendars >/dev/null 2>&1
```

The script will run at every even hour (because of the “*/2” for the hour). If you need a higher frequency for updates (or even a lower) adjust the time settings.

8.5 Working with source code blocks

Learning goals for this lesson

In this lesson I will show you how to utilize org-babel to write and execute small source code snippets inside your Org file.

Customizing org-babel

The first thing you need to do to use source code in your Org files is to customize the variable *org-babel-load-languages*. The list of languages that org-babel supports is long, so for this lesson we just want to enable 2 of them:

shell is used for shell code snippets

dot is used for GraphViz code. GraphViz is a tool that lets you describe graphs in text form.

Anatomy of a source code block

A source code block usually looks like this:

```
##+BEGIN_SRC
.
.
##+END_SRC
```

Everything between beginning and end of that block is written as source code for the selected language.

A simple shell script example

A while ago I was using an accounting program that was running on a web appliance. Every month I scheduled an accounting session and my capture template to schedule this looked like this:

```
TODO Accounting for %^{Month} [0/2]
- [ ] Do the accounting
- [ ] Backup the accounting database
##+BEGIN_SRC sh :dir /ssh:root@example.org:backup :results silent
pg_dump -h localhost -U kivitendo fy2019 > backup-%\1.sql
##+END_SRC
##+BEGIN_SRC sh :dir ~/Backups/FY2019 :results silent
scp root@example.org:backup/backup-%\1.sql .
##+END_SRC
```

So I had a checklist that says “do the accounting” and then “Backup the accounting database”. To do this backup more or less automatically I inserted 2 code blocks with simple shell code.

The first block opens a SSH connection³ to my appliance server, switches to the backup directory there and runs a “pg_dump” to dump the database to a backup file named backup and the month that I entered when creating that task with the capture template.

The second code block retrieves that backup file from the remote server and downloads it on my system via secure copy.

So now, when I wanted those shell code to execute all I had to do is positioning the cursor in the code block and press `Ctrl`+`c` `Ctrl`+`c` which executes the code.

At the end of each block start you see a property `:results`. This can have the following values:

raw shows the raw results.

table shows the results as a table.

list shows the results as a list.

silent does not show any results.

³SSH was enabled through RSA keys, so there was no need to know passwords.

A simple GraphViz example

Lets write down some dot code to describe the state changes in our GTD system. The code block looks like this:

```
#+BEGIN_SRC dot :file graph.png :cmdline -Kdot -Tpng
digraph G {
#      rankdir=LR;
      new_idea [ label="CAPTURE" shape=box];
someday [color = "red"];
{rank=same; done, cancelled, forwarded}
next [color="red"];
todo [color="red"];
repeat [color="red"];
next -> repeat;
next -> delegated;
todo -> repeat;
waiting [color="red"];
delegated [color="red"];
done [color="green"];
cancelled [color="green"];
forwarded [color="green"];
new_idea -> someday;
# new_idea -> next;
new_idea -> todo;
# new_idea -> repeat;
repeat -> repeat;
someday -> todo -> next;
someday -> next;
next -> waiting;
waiting -> next;
todo -> done;
next -> done;
repeat -> done;
waiting -> done;
todo -> cancelled;
next -> cancelled;
waiting -> cancelled;
someday -> cancelled;
todo -> delegated;
delegated -> done;
delegated -> cancelled;
todo -> forwarded;
}
#+END_SRC

#+RESULTS:
```

You see that some lines are commented out with “#” at the beginning. You also see a line `#+RESULTS:` at the bottom. When we execute the dot code with

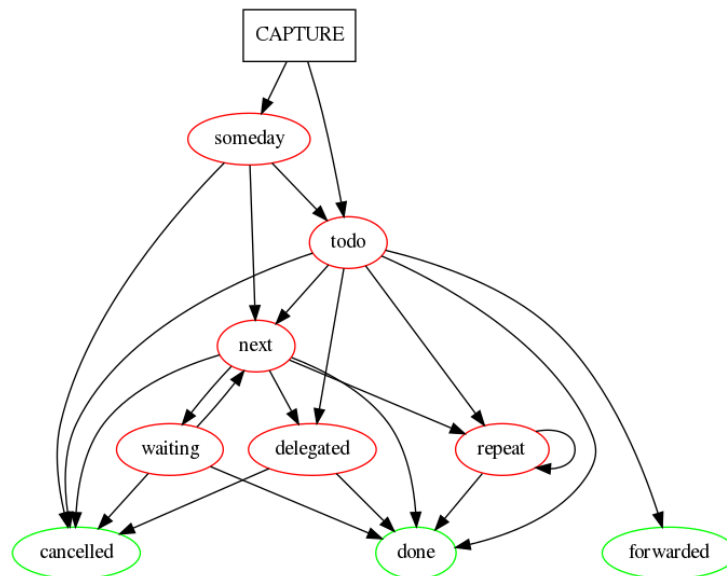


Figure 8.3: State changes in our GTD system

`Ctrl` + `c` `Ctrl` + `c` then a file named “graph.png” will be created and under the `#+RESULTS:` line we find a file link to this file.

Figure 8.3 shows the graph as we see it when we click on the file link under `#+RESULTS:`.

In our start line of the source code block we passed the `:file` property with the name of the file we want our GraphViz output to be and we also passed a small command line after the property `:cmdline`. In this way we instruct GraphViz that we need to call the “dot” utility and that our output should be in PNG format.

8.6 Goal setting and tracking

Learning goals for this lesson

In this lesson you will learn about a way to set and track goals with Org-mode.

What are goals?

To start with goals let us have a look at some goals:

Make America great again.

(Donald J. Trump)

I believe this nation should commit itself to achieving the goal, before this decade is out, of landing a man on the moon and returning him

safely to Earth.

(John F. Kennedy)

Now in direct comparison you see, that Donald Trump's goal is not very clear:

- What does “great” mean? Great in size, which will hardly work out without conquering some of the neighbor states? Great as a leading nation?
- What is the time frame of this goal? Next week? 2021? 2099? Nobody knows.

Kennedy on the other hand has set a deadline (before this decade is out) and a clear goal (land a man on the moon and return him safely to Earth).

So when it comes to your specific goals you should at least specify better goals than Donald Trump.

If you think about goals they can be split up into sort of two categories:

Destination goals

You want to achieve this goal like traveling to some destination and once you're there you check this goal off as achieved. An example goal of this category could be “Travel to a the North Cape in Norway on June 21st and experience the midnight sun.”

Journey goals

Those goals are different, they don't describe a destination to reach, they describe habits and your value. Journey goals are things like “Go to the gym at least twice a week” or “Practice piano every day”. Or if we quote the author Ralph Waldo Emerson: “*Life is a journey, not a destination.*”

So for your journey goals you can use the habit tracking as we have seen in section 8.2 or even schedule tasks that repeat in the frequency that you want.

Destination goals should be tracked differently. I created a capture template that makes use of the SMART scheme for goals. SMART stands for:

Specific describes what your goal is.

Measurable quantify or at least suggest an indicator for progress.

Actionable defines what actions you should do to achieve this goal.

Resources defines what resources you want to invest to achieve this goal.

Time bound defines your deadline but also how much time you want to spend each week or month to achieve this goal.

So my approach was to define the goal in a capture template:

```
* GOAL %^{Describe your goal}
  Added on %U - Last reviewed on %U
  :SMART:
```

```

:Sense:      %^{What is the sense of this goal?}
:Measurable: %^{How do you measure it?}
:Actions:    %^{What actions are needed?}
:Resources:  %^{Which resources do you need?}
:Timebox:    %^{How much time are you spending for it?}
:END:

```

So the template introduces a new `ToDo` keyword “GOAL”. This means your Org-file should have a line like this:

```
#+TODO: GOAL(G) | ACHIEVED(a@) MISSED(m@)
```

So as long as we are not hitting the deadline the goal is open, once we have reached the due date of this goal, we have to decide (according to our metrics that we defined for measuring the success if we switch the status to “ACHIEVED” or “MISSED”.

The template also creates a custom drawer that is called **SMART** which holds all the SMART criteria. Here I used the “S” for “Sense”, which makes me think on the “Why” I want to achieve this goal.

We also have 2 timestamps, one that tells us when we added this goal, and one that we can update manually whenever we review our goals.

Workflow for goals

Define a goal

The first step is to define a new goal. I use my capture template for this. Lets say we want a goal like read at least 1 book every month.

So after applying my template I see something like that:

```

* Goals
** GOAL Read at least 1 book every month
   Added on [2020-01-01 Mo 10:17] - Last reviewed on [2020-08-07 Fr 20:20]
   :SMART:
   :Sense:      Train my brain, learn something new, gain knowledge
   :Measurable: Check if I read at least 12 books ate the end of the year
   :Actions:    Read frequently, best on a daily schedule
   :Resources:  Time to read, interesting books
   :Timebox:    30 minutes up to 1 hour per day
   :END:

```

The “Goals” headline is at the start of my Org-file so that I can find it quickly when I want to have a look at my current goals.

So, my goal is set. Time for another quote:

“A goal without a plan is just a wish.”

(Antoine de Saint-Exupery)

Make a plan

So my next step is to assign a tag to my goal. Since I have set this goal for 2020 I use something like “G@a2020_reading” as a tag, so that I can easily spot this as a Goal for 2020.

Now lets go to my “Books to read” headline where all my captured books are stored below. There I assign the same tag, so that every book that is in the tree below gets this tag as an inherited tag.

Review my goals

Whenever I want to quickly review my goals I just have to go on top of my Org-file where the “Goals” headline is. Then I expand the subtree to see all my goals.

Now if I click on one of the assigned tags, I get an agenda view with all tasks that have this goal assigned. On top of that agenda list I see my goal again, then in my book reading example I see the level 1 headline “Books to read” and below that all books that are in my “to read” list. So I can really quickly check if I’m running out of books or not.

This helps me to see all the actions that I have planned to achieve that specific goal. If I don’t see actions there its time to think about what I need to schedule to go forward towards my goal.

8.7 Presenting my system

Learning goals for this lesson

In this lesson I will present you the my system as I use it in summer 2020. The system has developed over the past years and thanks to Org-mode being Open Source I was able to adapt any new idea to my system instead of a system that forces me to adapt workflows to the limitations of the system.

Background about my situation

To understand the system the basic idea is to distinguish the “hats” I’m wearing in my life. As everyone in this world I have to fit in different roles, so I structured it down to 3 “hats”:

- The private life in my little kingdom⁴, being a husband, father, musician and all the things I do in my private free time. So this hat is a crown.
- The work life as a Linux software engineer. At the time of writing this I’m evaluating new opportunities because my old job got lost after a shutdown of our site. Nevertheless, work is symbolized by the red hat (even if I’m using openSUSE) and stands for the part of your life that generates income to pay the bills.

⁴If you don’t know, my name “König” is German for “king”.

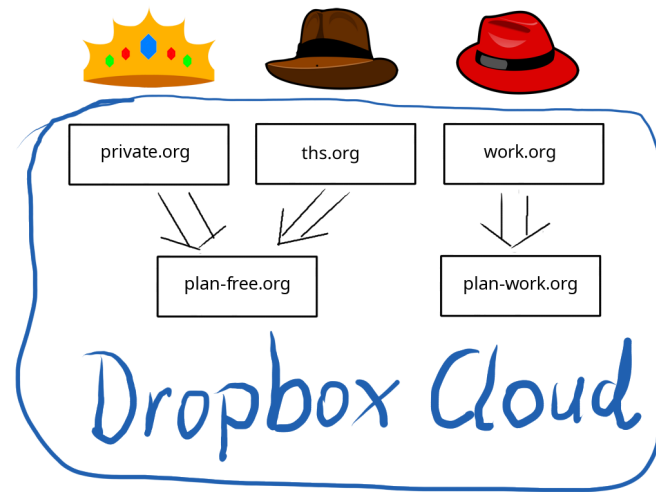


Figure 8.4: My life “hats” and the rough file structure

- The third hat I’m wearing is being the president of a local shooting club. This can be compared with managing a small business, just with the difference that I don’t get paid to do all the bureaucracy and service.

File structure

All the files I’m using for my system are stored in a folder on my Dropbox drive. This enables me to use the system on different computers, e.g. on my desktop PC and also on my notebook computer. To keep the Org-mode settings and capture templates, agenda views synchronized between the different machines I have created a file “org-global.el” that is also stored in the Dropbox and loaded on every computer.

Figure 8.4 shows that for every of my “hats” I have a backlog file that holds all the items that are relevant for that role.

private.org is for my private life.

work.org is for the work life.

ths.org is for the shooting club stuff.

For every “hat” there is also a “journal-*hat*.org” file, like e.g. “journal-private.org”. Those are datetrees to collect journal entries that are not related to a task item.

Finished items from those files end up in different archive files, depending on projects or overall areas, e.g. “track-read-books.org” or “maintenance-archive.org”.

For agenda views I use the hack described in section 4.2 to select the agenda files based on a “focus”. I have the following custom commands:

org-focus-private sets agenda files to “private.org”.

org-focus-work sets agenda files to “work.org”.

org-focus-ths sets agenda files to “ths.org”.

org-focus-free sets agenda files to “ths.org” and “private.org”.

org-focus-all sets agenda files to “ths.org”, “private.org” and “work.org”.

I also have 2 datetree files called “plan-free.org” and “plan-work.org” which I use every morning to create a plan for that day.

More ToDo keywords

I also extended the set of ToDo keywords by 2 new keywords:

DELEGATED

This is used for a task that I delegated to someone els (e.g. one of my kids or a coworker), but I still need to monitor the progress (especially when teenagers are involved). So tasks that are **DELEGATED** are still “open”, but I know that I have no action in them except looking if the one that should do it really works on it.

FORWARDED

This is used for tasks that hit my system, but where I’m not the person that should work on this. Lets say a customer found out my mail address and sends me a question on prices of whatever, then I forward this question to my colleague in the sales department and mark this task as **FORWARDED**. That means for me this task is considered as “done” and there is no need for me to monitor if my colleague in sales answers this or not.

Workflows

Creating my daily plan

First thing in the morning is to create my daily plan. The daily plan file is a datetree and gets the current day appended by applying a capture template. That means, I have a collection of all daily plans over the year and thanks to the progress indicators for the checklist that each daily plan represents I can easily see how good I was able to stick to my plan.

To create the daily plan I apply my capture template and then I open a second emacs window with `Ctrl+x 5 2` where I open the agenda view for the given focus area (private or work) and the current day.

There I might see some scheduled items for today as well as some overdue items. Doesn’t matter. First thing is to look for the 3 most important tasks for this day. When I want to enter them in my daily plan all I need to do is to

- Place the cursor in the line of the agenda view and press `F5`.
- Place the cursor in the daily plan and press `Ctrl+y`.

To make this work I extended the code in section 6.2 so that pressing `F5` creates a link in the clipboard that links to the ID of that agenda item and has the item text as the link text.

So my daily plan can be done quickly, first the 3 most important task, then a bunch of tasks that are scheduled or overdue and at the end I also have checkboxes for some chores I need to do, even if they are not in the system. If the basket with laundry cloths is full, I know that I have to start a washing machines, no need to track this in Orgmode. But it will end up as a chore on my daily plan that says “do the laundry”.

At this point you probably want to ask: “But why are you doing an extra plan and not just using the agenda view?”. This has several reasons:

- An overfull agenda view can be distracting, disappointing and demotivating. With this approach I see the mess only once in a day and after creating the daily plan, this is the view I have on my screen. No bad feelings about tasks that didn’t make it on todays plan.
- When doing the scheduling during the weekly review I’m usually optimistic about what I can do on a day. But on the morning of every day I really can estimate, how much energy I have for this day. I might have a hangover, feel sick or whatever, so I can limit my daily plan to the things I feel that I’m able to achieve on this day. This is practically the final decision on how I want to use this day.
- It is very satisfying top check off items from the daily plan. So the plan is giving me small rewards during the day.

So when I finished a job on my daily plan, I can follow the link to the org file from the daily plan and mark this task as “DONE” in my backlog Org-file as well. If I’m just working on that task, but didn’t complete it, I can check off the item on the checklist, and maybe reschedule the task in the backlog file for another day. For example, while I’m writing these lines I have an item “Write about my system” on my daily plan, but I won’t finish it today, because I need to create some illustrations and that I want to do tomorrow. But I already spent time, focus and energy on this text, so it is legit to check that item off the daily plan for today.

The weekly review

Every of my three “hats” has its own weekly review, because the topics usually don’t overlap and so I can focus on one role instead of three.

The weekly review is scheduled as a repeating task and has a checklist in it.

```
WR BOOTUP
=====
- [ ] Remove all NEW tags from the tasks in the backlog

GET CLEAN
=====
Inbox Zero (Create new tasks if needed):
```

```

- [ ] Snail Mail
- [ ] Email
- [ ] Journal
- [ ] Voicemail, Others
- [ ] Meeting minutes of meetings last week
- [ ] Other reference material
Braindump 1:
- [ ] Set a timer for 5 minutes and write down what's in your mind

CHECK CURRENT
=====
Check WAITING actions and do this for each item:
- [ ] Did something get delivered? If so change status to NEXT
- [ ] Postpone reminder if still waiting and its overdue
Check DELEGATED actions and do this for each item:
- [ ] Is the task finished? If so mark it as DONE
- [ ] Is the task overdue? If so schedule a supportive action
Check your projects and tasks in detail:
- [ ] Review Project List, is something stuck?
- [ ] See your goals list and the associated actions
- [ ] Browse through your actions and mark finished things DONE or CANCELLED
- [ ] Schedule tasks with no date
- [ ] See your schedule for the next 2 weeks an reschedule if its too full
Rate your performance, energy and write a journal entry:
- [ ] Check last weeks daily plan on how good the performace was
- [ ] Create short Journal entry of what was bad & good last week

FUTURE ACTIONS
=====
Prepare for the future:
- [ ] Is a Someday/Maybe becoming a task/project?
- [ ] If a Someday is no longer of interest set it to "Cancelled".
Braindump 2:
- [ ] Set timer to 5 minutes and start dreaming
- [ ] Put the results into Actions/Someday/Maybe

WR SHUTDOWN & ARCHIVING
=====
- [ ] Go through REPEAT actions and shorten the logbook if needed
- [ ] Select finished tasks in agenda view and do a bulk archive action

```

So when I do the weekly review I open a second window and extend it to almost the full screen size on my left monitor. The checklist is open on my right monitor and I start doing the small actions and then check off every item of this checklist.

At the end I mark this repeating task as “DONE” which, thanks to a small snippet of Emacs Lisp code resets all checkboxes so that my checklist is cleared for the next weeks iteration and the date is scheduled again.

Code snippets

As mentioned above, I did some small Emacs-Lisp code that makes my life easier. Here is the code.

Create link in clipboard

The following code creates a link based on the ID of an item and writes it to the clipboard. It will be activated by pressing **F5** in an Org-mode or agenda buffer and you can paste the link then to the daily plan.

(**defun** my/copy-idlink-to-clipboard() "Copy an ID link with the headline to killring, if no ID is there then create a new unique ID. This function works only in org-mode or org-agenda buffers.

The purpose of this function is to easily construct id:links to org-mode items. If its assigned to a key it saves you marking the text and copying to the killring."

```
(interactive)
(when (eq major-mode 'org-agenda-mode)
  (org-agenda-show)
  (org-agenda-goto))
(when (eq major-mode 'org-mode) ; do this only in org-mode buffers
  (setq mytmphead (nth 4 (org-heading-components)))
  (setq mytmpid (funcall 'org-id-get-create))
  (setq mytmplink (format "[[id:%s][%s]]" mytmpid mytmphead))
  (kill-new mytmplink)
  (message "Copied %s to killring (clipboard)" mytmplink)
))

(global-set-key (kbd "<f5>") 'my/copy-idlink-to-clipboard)
```

In my case this code is stored in org-global.el in my Dropbox, so that it is available on all PCs.

Clear all checkboxes when done

The following code was taken from one of the community sites. The purpose is to clear all checkboxes of a checklist when the repeating task is marked as done. In my case I use it for my weekly review checklist.

```
(defun org-reset-checkbox-state-maybe ()
  "Reset all checkboxes in an entry
  if the `RESET_CHECK_BOXES' property is set"
  (interactive "*")
  (if (org-entry-get (point) "RESET_CHECK_BOXES")
      (org-reset-checkbox-state-subtree)))

(defun org-checklist ()
  (when (member org-state org-done-keywords)
    (org-reset-checkbox-state-maybe)))

(defun my-clockfiles ()
  (append org-agenda-files
    (file-expand-wildcards "~/Dropbox/org/track*.org")))
```

```
(add-hook 'org-after-todo-state-change-hook 'org-checklist)
```

To make this code work the task with the checklist has to get a property `RESET_CHECK_BOXES`. In my weekly review checklist the `PROPERTIES` drawer looks like this:

```
:PROPERTIES:
:ID:          2d47045a-c960-489f-8d71-de35ea036a19
:Effort: 0:30
:LOGGING: nil
:RESET_CHECK_BOXES: t
:LAST_REPEAT: [2020-07-26 So 20:24]
:END:
```

You also see that even if logging is turned off by setting the `LOGGING` to “nil” the last repeat of this task is recorded in the `LAST_REPEAT` property.

Chapter 9

Congratulations, you're done

So now you reached the end of this course and I hope, that you learned something that you can use in your daily life. For me Org-mode has become my trusted system. As you saw in section 8.7 I use it for all of my roles and I plan every day with it.

Of course it is up to you how you use it for your needs. Never forget, that are free to adapt the system to your needs, you don't need to adapt yourself to a system that puts pressure on you to do things like the software developer thought you need to do the things, even if you have a complete different opinion on how to do things.

Org-mode is still under development, so follow the news and read the release notes when new versions come out.

Bibliography

- [AF15] David Allen and James Fallows. *Getting Things Done - The Art of Stress-Free Productivity*. Revised. New York: Penguin, 2015. ISBN: 978-0-698-16186-3.
- [Cle18] James Clear. *Atomic Habits - The life-changing million copy best-seller*. New York: Random House, 2018. ISBN: 978-1-473-53780-4.
- [Duh13] Charles Duhigg. *The Power of Habit - Why We Do what We Do and how to Change*. New York: Random House, 2013. ISBN: 978-1-847-94624-9.
- [MDP14] Dr Theo Compernelle MD.PhD. *Brainchains - Your Thinking Brain Explained in Simple Terms. Full of Practical Tools, Tips and Tricks to Improve Your Efficiency, Creativity and Health. How to Cope Better with Ict, Being Always Connected, Multitasking, Email, Social Media, Lack of Sleep and Stress*. 1. Aufl. Compuplications, 2014. ISBN: 978-9-082-20580-0.
- [Rey08] Garr Reynolds. *Presentation Zen - Simple ideas on presentation design and delivery*. New Riders, 2008. ISBN: 978-0-321-62565-9.

Index

- Agenda view, 12
 - advanced, 20
 - bulk actions, 77
 - customized, 23
 - priorities, 60
 - restriction, 29
- Archiving, 27
 - target, 28
- Attachments, 55
- Capture
 - template, 35, 39
- Checklists, 16
- Clocking, 45
- Column view, 47
- Customization
 - diary-list-entries-hook, 79
 - org-agenda-custom-commands, 23
 - org-agenda-dim-blocked-tasks, 44
 - org-agenda-files, 13, 34
 - org-agenda-include-diary, 79
 - org-agenda-overriding-header, 25
 - org-agenda-span, 21, 25
 - org-archive-location, 28
 - org-attach-id-dir, 55
 - org-attach-method, 56
 - org-attach-use-inheritance, 58
 - org-babel-load-languages, 81
 - org-capture-templates, 38
 - org-clock-into-drawer, 46
 - org-deadline-warning-days, 13
 - org-enforce-todo-checkbox-dependencies, 44
 - org-enforce-todo-depedencies, 44
 - org-export-backends, 65
 - org-file-apps, 65
 - org-habit-show-habits-only-for-today, 76
 - org-hide-leading-stars, 9
 - org-id-locations-file, 53
 - org-log-done, 33
 - org-log-into-drawer, 26
 - org-log-repeat, 16
 - org-log-reschedule, 33, 79
 - org-modules, 75
 - org-publish-project-alist, 69
 - org-refile-allow-creating-parent-nodes, 34
 - org-refile-targets, 34
 - org-refile-use-outline-path, 34
 - org-track-ordered-property-with-tag, 44
 - org-use-property-inheritance, 58
- datetree, 39
- Deadline, 12, 78
- Drawers, 26
- Dropbox, 53
- Dynamic blocks, 71
 - clocktable, 71
 - columnview, 73
- Effort estimates, 49
- Eisenhower
 - priority matrix, 59
- ELPA, 6
- Emacs Lisp, 24, 35
- Exporting, 63
 - Advanced, 65
 - HTML, 63
 - PDF, 63
- Follow mode, 13
- Goals, 84
 - SMART, 85
- Google calendar, 79
- GraphViz, 81
- Getting Things Done, i, 2

- Habits, 74
- Headline, 7, 51
- Journaling, 39
- Key bindings
 - agenda view, 13
 - my/copy-id-to-clipboard, 54
 - org-capture, 36
- Linking
 - attachments, 58
 - external, 52
 - internal, 51
- Logging, 26
 - automatic logging, 31
- My system, 87
- nginx, 68
- Ordered tasks, 43
- Package management, 6
- Pomodoro technique, 2, 46
- prefixing, 21, 44, 78
- Priorities, 58
- Progress indicator, 16
- Project, 10
- PROPERTIES, 26, 40
 - COLUMNS, 48
 - CUSTOM_ID, 51
 - DIR, 55
 - EFFORT, 49
 - ID, 53, 55
 - LAST_REPEAT, 93
 - LOGGING, 32, 93
 - ORDERED, 43
 - RESET_CHECK_BOXES, 93
 - STYLE, 75
- Publishing, 68
- Quick notes, 26
- Refiling, 9, 33, 78
- Regular expression, 22
- Reschedule, 14
- Scatter, 78
- Schedule, 12, 78
- Source Code Blocks, 81
- Splitting files, 33
- Tables, 60
- Tags, 19
 - inheritance, 20
 - search, 22
- Targets
 - Named targets, 51
 - Radio targets, 51
- Timers, 44
- Timestamp, 12, 27, 45, 70
- ToDo keywords, 9, 14
- Version, 5
- Weekly review, 11, 44, 77

Appendix A

Key combinations from all lessons

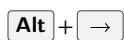
2.2 Headlines & outline mode



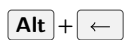
local visibility cycling.



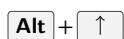
global visibility cycling.



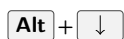
increase level by one.



decrease level by one.



move block upwards.



move block downwards.



refile block.



re-read and activate configuration line.

2.3 ToDo keywords



cycle right through the defined ToDo keywords.



cycle left through the defined ToDo keywords.








display a menu with all ToDo keywords and their hotkeys to change the ToDo keyword of an item.

2.4 Schedule, deadlines & agenda view

Ctrl + **c** **Ctrl** + **s**
create a **SCHEDULED**: timestamp.

Ctrl + **c** **Ctrl** + **d**
create a **DEADLINE**: timestamp.

 **plus** , ,  or 
move around in the calendar view to pick a date.



Ctrl + **c** **a**
invoke the agenda dispatcher menu¹.

f or **b**
move forward or backward in the agenda view.

 + **f**
enable follow mode in the agenda view.


2.6 Checklists

Ctrl + **c** **Ctrl** + **c**
check off a checklist item or update the progress indicator (depending on cursor position).

Alt +  + 
quick entry of new checklist line start.

3.1 Tags

Ctrl + **c** **Ctrl** + **q**
assign a tag to a headline.


enter a free tag when assigning tags.

3.2 Advanced agenda views

Ctrl + **u**
prefix an agenda command.

3.4 Drawers, logging and quick notes

Ctrl + **c** **Ctrl** + **z**
capture a quick note.

Ctrl + **c** **Ctrl** + **c**
finish capturing the quick note and store it the Org-file.

¹This is a custom key binding that needs to be defined in your Emacs startup file

3.5 Archiving

Ctrl + **c** **Ctrl** + **x** **a**
toggle `:ARCHIVE:` tag.

Ctrl + **c** **Ctrl** + **x** **Ctrl** + **s** or **Ctrl** + **c** **\$**
archive subtree.

4.1 Automatic logging of status changes

Ctrl + **c** **Ctrl** + **c**
finish log entry and store it in the Org-file. item(**Ctrl** + **c** **Ctrl** + **k**)
cancel log entry and don't save it in the Org-file.

4.2 Splitting your system into several files

Ctrl + **c** **Ctrl** + **w**
move item to another location and remove the original item.

Ctrl + **c** **Alt** + **w**
copy item to another location and leave the original unchanged.

4.3 The first capture template

F6
open the capture menu².

↑ + **c**
when in capture menu, then enter the capture template editor.

5.1 Ordered tasks

Ctrl + **c** **Ctrl** + **x** **↑** + **o**
toggle the `ORDERED` property.

5.2 Timers

Ctrl + **c** **Ctrl** + **x** **;**
start a countdown timer. Prefix this with **Ctrl** + **u** to define a time.

Ctrl + **c** **Ctrl** + **x** **0**
start a relative timer.

Ctrl + **c** **Ctrl** + **x** **,**
pause or restart a timer.

Ctrl + **u** **Ctrl** + **c** **Ctrl** + **x** **,**
stop the current timer.

²This is a custom key binding and needs to be defined in your Emacs startup file.

Ctrl + **c** **Ctrl** + **x** **.**
insert a simple timestamp.

Ctrl + **c** **Ctrl** + **x** **-**
insert a description timestamp.

5.3 Clocking

Ctrl + **c** **Ctrl** + **x** **Ctrl** + **i**
“clock in” a task.

Ctrl + **c** **Ctrl** + **x** **Ctrl** + **i**
“clock out” a task.

Ctrl + **c** **Ctrl** + **x** **Ctrl** + **q**
cancel a clock.

Ctrl + **c** **Ctrl** + **x** **Ctrl** + **j**
jump to the task that is currently clocked.

Ctrl + **c** **Ctrl** + **x** **Ctrl** + **x**
restart the last used clock.

Ctrl + **u** **Ctrl** + **c** **Ctrl** + **x** **Ctrl** + **x**
display a menu of the last clocked task to restart one of them.

Ctrl + **c** **Ctrl** + **x** **Ctrl** + **d**
show the sum of all clock entries for every task (this creates an overlay).

5.4 Column view

Ctrl + **c** **Ctrl** + **x** **Ctrl** + **c**
switches to column view.

q
leaves the column view.

5.5 Effort estimates

↑ + **→**
step forward through the predefined efforts.

↑ + **←**
step backward through the predefined efforts.

6.1 Linking (internal)

Ctrl + **c** **Ctrl** + **c**
activate radio target.

Ctrl + **c** **Ctrl** + **l**
edit link.

Ctrl + **c** **Ctrl** + **o**

follow a link.

Ctrl + **c** **&**

return to previous position.

6.2 Linking (external)

F5

Fetch ID for the item at the cursor position and copy it to the clipboard.
If no ID exists create a new one³.

6.3 Attachments

Ctrl + **c** **Ctrl** + **a**

Invoke Org Attach and display the command menu. The next keys refer to the possible commands.

a

attach file by using the method configured in the variable *org-attach-method*.

c

attach file by copying it to the attachment directory.

m

attach file by moving it to the attachment directory.

l

attach file by creating a hard-link.

y

attach file by creating a symbolic link.

u

download file from URL and store it in the attachment directory.

b

store the contents of an active buffer in the attachment directory.

n

create a new buffer with an attachment.

z

synchronize attachment directory if you added files manually.

o

Opens the attachment(s)

⇧ + **o**

Opens the attachment(s) forced in Emacs.

³This is a custom key binding that needs to be defined in your Emacs startup file.

f

Opens a view to the node attachment directory.

↑ + **f**

Force open the attachment directory using Dired in Emacs.

d

Deletes one attachment.

↑ + **d**

Deletes all attachments.

s

Sets a specific attachment directory for this entry. The DIR property is set.

↑ + **s**

Unsets the DIR property.

q

Abort.

6.4 Priorities

↑ + **↑**

increase priority of a task.

↑ + **↓**

decrease priority of a task.

Ctrl + **c** **,**

select a priority for a task.

6.5 Tables

| - **↔**

inserts a horizontal line.

Ctrl + **c** **-**

inserts a horizontal line.

Alt + **←**

move column to the left.

Alt + **→**

move column to the right.

Alt + **↑**

move row up.

Alt + **↓**

move row down.

7.1 Exporting

Ctrl + **c** **Ctrl** + **e**

Invoke org export dispatcher. After that you have the following options:

h **↑** + **h**

Create an HTML buffer.

h **h**

Create an HTML file. The filename will be the same as for your Org file, but the extension will be “.html” instead of “.org”.

h **o**

Create a file and open it.

l **↑** + **L**

Creates a \LaTeX buffer.

l **l**

Creates a \LaTeX file.

l **p**

Creates a PDF file with the same name as your Org-file but with the extension “.pdf” instead of “.org”.

l **o**

Creates a PDF file and opens it.

7.2 Advanced exporting Additional keys for the \LaTeX “beamer” backend:

↑ + **b**

Exports the \LaTeX code into a new Emacs buffer.

b

Exports the \LaTeX code into a file.

↑ + **p**

Exports to a PDF file using the beamer class.

↑ + **o**

Exports to a PDF file using the beamer class and opens this file.

7.3 Publishing Invoke the publishing backend with **↑** + **p**:

f

publishes the current file.

p

publishes the current project.


x


chose a project to publish.



a



publishes all projects.

8.1 Dynamic blocks

Ctrl + **c** **Ctrl** + **x** **x** clocktable 
inserts a clocktable dynamic block.

Ctrl + **c** **Ctrl** + **x** **x** columnview 
inserts a columnview dynamic block.

 + 
decrement date block, e.g from 2020-06 to 2020-05.

 + 
increment date block, e.g from 2020-06 to 2020-07.


Ctrl + **c** **Ctrl** + **c**
update dynamic block.

8.3 Bulk agenda actions Marking agenda items:

m marks entry at cursor position.

u un-marks item at cursor position.



***** mark all agenda entries.

 + **u** un-marks all agenda entries.

Alt + **m** toggles mark at cursor position.

Alt + ***** toggles all marks in the agenda.


% mark entries based on a regular expression.

Invoking bulk agenda:  + **b** or **Ctrl** + **u**  + **b**.

Bulk agenda actions:

p
toggle between persistent and non persistent marks.

\$
Archive all marked entries to their archive files.

 + **a**
Archive all marked entries by moving them to their respective siblings.

t
Change ToDo keyword for the marked items.

+
Assign a tag to all marked items.

-
Remove a tag from all marked items.

- s Schedule all marked items to a specific date. You can also shift the dates by relative values like “++8d”. If you used **prefixing** for the bulk action then all schedule information will be removed from the marked items.

- d Set deadline for all marked item to a specific date. If you used **prefixing** for the bulk agenda action then all deadline information will be removed from the marked items.

- r Refile the marked entries.

- ↑ + s Scatter the marked items over a given time period (usually 7 days). This might help for scheduling. If you used **prefixing** for the bulk agenda action then the tasks are scattered only across weekdays and weekends are excluded.

- f apply a function to the marked entries. See the Org-mode manual for details.

Appendix B

Acknowledgments

First I have to thank the team behind Org, especially Carsten Dominik and Bastien Guerry, but also many others, for providing such a great tool to the community. Started in the year 2003 Org is still maintained and evolving.

Special thanks go to my friend Anders Raynar Karlsson. Many years ago, somewhere between 2005 and 2010 Anders was our Technical Account Manager at Red Hat and one day we met in a bar to have a drink. I was a proud user of some ToDo list on my Blackberry smartphone at this time and Anders said he's doing GTD with Org-mode. That made me curious, so when I was back home I searched for this "Org-mode" thing and got infected.

The next one to thank is my former colleague Werner Merkl. It was in 2013 when I was hit by the "shiny new objects" syndrome and tried out other fancy GTD tools on my brand new state-of-the-art Android smartphone. He reminded me that Org-mode was still developing new features, so I had another look at it and switched back from IQTELL at this time to Org-mode. And never looked at another shiny new tool since then. In fact, Org-mode is the only tool I know that does not force me to adapt to a special workflow that the tool enforces, but let me adapt the tool to my workflow.

The next people to thank are the members of the LinkedIn GTD group. We had some discussions on GTD tools and I told them, that I do GTD in a text editor. If they wouldn't have started to get curious on that, I would not have started to "quickly record a video to show them". Which then led to a series of tutorial videos on YouTube.

I also have to say thanks to almost 3400 subscribers of my YouTube channel that motivated me by posting comments and asking questions. That feedback was very important for me to find out what I didn't explain well enough and what needs to be clarified. So many positive and motivating comments to go on. It showed me that Org-mode might be a product for a relatively small user base, but there is demand for tutorials.

Thanks also to Alex Koval for pointing me to videos about "Literate Devops" which became very useful when exploring the work with source code blocks.

A special thanks to Sacha Chua with her weekly Emacs news web site where my videos were featured many times and brought me many subscribers. Also the people on Reddit who recommended my videos.

Thanks to Heiko Westphal who ordered some guitar strings for me to say thank you for my tutorials. That was really a big surprise. Also thanks to Mia Andal one that donated 10 Dollars via PayPal, didn't make me rich, but made me smile a lot.

Finally I have to thank Eduardo Mercovich who recently linked to my YouTube tutorials from the Worg community manual.