Org Mode - Organize Your Life In Plain Text!

Bernt Hansen (IRC:Thumper_ on freenode)*

October 15, 2011

Contents

*bernt@norang.ca

Org-mode is a fabulous organizational tool built by Carsten Dominik that operates on plain text files. Org-mode is part of Emacs.

This document assumes you've had some exposure to org-mode already so concepts like the agenda, capture mode, etc. won't be completely foreign to you. More information about org-mode can be found in the Org-Mode Manual and on the Worg Site.

I have been using org-mode as my personal information manager for years now. I started small with just the default `TODO` and `DONE` keywords. I added small changes to my workflow and over time it evolved into what is described by this document.

I still change my workflow and try new things regularly. This document describes mature workflows in my current org-mode setup. I tend to document changes to my workflow 30 days after implementing them (assuming they are still around at that point) so that the new workflow has a chance to mature.

Some of the customized Emacs settings described in this document are set at their default values. This explicitly shows the setting for important org-mode variables used in my workflow and to keep my workflow behaviour stable in the event that the default value changes in the future.

1 Getting Started

I use `org-mode` in most of my emacs buffers.

1.1 Org-Mode Setup

The following setup in my .emacs enables org-mode for most buffers. `org-mode` is the default mode for `.org`, `.org_archive`, and `.txt` files.

```
;;;
;;; Org Mode
;;;
(add-to-list 'load-path (expand-file-name "~/git/org-mode/lisp"))
(add-to-list 'auto-mode-alist '("\\.\\(org\\|org_archive\\|txt\\)$" . org-n
(require 'org-install)
;;
;; Standard key bindings
(global-set-key "\C-cl" 'org-store-link)
(global-set-key "\C-ca" 'org-agenda)
(global-set-key "\C-cb" 'org-iswitchb)
```

`orgstruct++-mode` is enabled in `Gnus` message buffers to aid in creating structured email messages.

```
(add-hook 'message-mode-hook 'orgstruct++-mode 'append)
(add-hook 'message-mode-hook 'turn-on-auto-fill 'append)
(add-hook 'message-mode-hook 'bbdb-define-all-aliases 'append)
(add-hook 'message-mode-hook 'orgtbl-mode 'append)
(add-hook 'message-mode-hook 'turn-on-flyspell 'append)
(add-hook 'message-mode-hook '(lambda () (setq fill-column 72)) 'append)
(add-hook 'message-mode-hook '(lambda () (local-set-key (kbd "C-c␣M-o") 'or
```

`flyspell-mode` is enabled for almost everything to help prevent creating documents with spelling errors.

```
;; flyspell mode for spell checking everywhere
(add-hook 'org-mode-hook 'turn-on-flyspell 'append)

;; Disable C-c [ and C-c ] in org-mode
(add-hook 'org-mode-hook
          (lambda ()
            ;; Undefine C-c [ and C-c ] since this breaks my
            ;; org-agenda files when directories are include It
            ;; expands the files in the directories individually
            (org-defkey org-mode-map "\C-c["    'undefined)
            (org-defkey org-mode-map "\C-c]"    'undefined)) 'append)

(add-hook 'org-mode-hook
          (lambda ()
            (local-set-key (kbd "C-c M-o") 'bh/mail-subtree)) 'append)

(defun bh/mail-subtree ()
  (interactive)
  (org-mark-subtree)
  (org-mime-subtree))

;; Enable abbrev-mode
(add-hook 'org-mode-hook (lambda () (abbrev-mode 1)))
```

1.2 Organizing Your Life Into Org Files

Tasks are separated into logical groupings or projects. Use separate org files for large task groupings and subdirectories for collections of files for multiple projects that belong together.

Here are sample files that I use.

The following org files collect non-work related tasks:

Filename	Description
todo.org	Personal tasks and things to keep track of
gsoc2009.org	Google Summer of Code stuff for 2009
farm.org	Farm related tasks
mark.org	Tasks related to my son Mark
org.org	Org-mode related tasks
git.org	Git related tasks
bzflag.org	BZFlag related tasks

The following org-file collects org capture notes and tasks:

Filename	Description
refile.org	Capture task bucket

The following work-related org-files keep my business notes (using fictitious client names)

Filename	Description
norang.org	Norang tasks and notes
XYZ.org	XYZ Corp tasks and notes
ABC.org	ABC Ltd tasks
ABC-DEF.org	ABC Ltd tasks for their client DEF Corp
ABC-KKK.org	ABC Ltd tasks for their client KKK Inc
YYY.org	YYY Inc tasks

Org-mode is great for dealing with multiple clients and client projects. An org file becomes the collection of projects, notes, etc. for a single client or client-project.

Client ABC Ltd. has multiple customer systems that I work on. Separating the tasks for each client-customer into separate org files helps keep things logically grouped and since clients come and go this allows entire org files to be added or dropped from my agenda to keep only what is important visible in agenda views.

Other org files are used for publishing only and do not contribute to the agenda. See Publishing for more details.

1.3 Agenda Setup

Here is my current `org-agenda-files` setup.

```
(setq org-agenda-files (quote ("~/git/org"
                               "~/git/org/client1"
```

```
"~/git/org/bzflag"
"~/git/client2")))
```

`org-mode` manages the `org-agenda-files` variable automatically using `C-c [` and `C-c ]` to add and remove files respectively. However, this replaces my directory list with a list of explicit filenames instead and is not what I want. If this occurs then adding a new org file to any of the above directories will not contribute to my agenda and I will probably miss something important.

I have disabled the `C-c [` and `C-c ]` keys in `org-mode-hook` to prevent messing up my list of directories in the `org-agenda-files` variable. I just add and remove directories manually in my `.emacs` file. Changing the list of directories in `org-agenda-files` happens very rarely since new files in existing directories are automatically picked up.

In the example above I have `~/git/client2` in a separate git repository from ~/git/org. This gives me the flexibility of leaving confidential information at the client site and having all of my personal information available everywhere I use org-mode. I synchronize my personal repositories on multiple machines and skip the confidential info on the non-client laptop I travel with. `org-agenda-files` on this laptop does not include the `~/git/client2` directory.

1.4 Org File Structure

Most of my org files are set up with level 1 headings as main categories only. Tasks and projects normally start as level 2.

Here are some examples of my level 1 headings in `todo.org`:

- Special Dates

  Includes level 2 headings for

  - Birthdays
  - Anniversaries
  - Holidays

- Finances
- Health and Recreation
- House Maintenance

- Lawn and Garden Maintenance
- Notes
- Tasks
- Vehicle Maintenance
- Passwords

`norang.org`:

- System Maintenance
- Payroll
- Accounting
- Finances
- Hardware Maintenance
- Tasks
- Research and Development
- Notes
- Purchase Order Tracking
- Passwords

Each of these level 1 tasks normally has a `property drawer` specifying the category for any tasks in that tree. Level 1 headings are set up like this:

```
* Health and Recreation
  :PROPERTIES:
  :CATEGORY: Health
  :END:
  ...
* House Maintenance
  :PROPERTIES:
  :CATEGORY: House
  :END:
```

1.5 Key bindings

I live in the agenda. To make getting to the agenda faster I mapped `F12` to the sequence `C-c a` since I'm using it hundreds of times a day.

I have the following custom key bindings set up for my emacs (sorted by frequency).

Key	For	Used
F12	Agenda (1 key less than C-c a)	Very Often
C-c b	Switch to org file	Very Often
F11	Goto currently clocked item	Very Often
C-M-r	Capture a task	Very Often
C-F11	Clock in a task (show menu with prefix)	Often
f9 g	Gnus - I check mail regularly	Often
f5	Show todo items for this subtree	Often
S-f5	Widen	Often
f9 b	Quick access to bbdb data	Often
f9 c	Calendar access	Often
C-S-f12	Save buffers and publish current project	Often
C-c l	Store a link for retrieval with C-c C-l	Often
f8	Go to next org file in org-agenda-files	Sometimes
f9 r	Boxquote selected region	Sometimes
f9 t	Insert inactive timestamp	Sometimes
f9 v	Toggle visible mode (for showing/editing links)	Sometimes
C-f9	Previous buffer	Sometimes
C-f10	Next buffer	Sometimes
C-x n r	Narrow to region	Sometimes
f9 f	Boxquote insert a file	Sometimes
f9 i	Info manual	Sometimes
f9 I	Punch Clock In	Sometimes
f9 O	Punch Clock Out	Sometimes
f9 o	Switch to org scratch buffer	Sometimes
f9 s	Switch to scratch buffer	Sometimes
C-c r	Capture a task (from my mobile phone)	Rare
f9 h	Hide other tasks	Rare
f7	Toggle line truncation/wrap	Rare
f9 u	Untabify region	Rare
C-c a	Enter Agenda (minimal emacs testing)	Rare

Here is the keybinding setup in lisp:

```
;; Custom Key Bindings
(global-set-key (kbd "<f12>") 'org-agenda)
(global-set-key (kbd "<f5>") 'bh/org-todo)
(global-set-key (kbd "<S-f5>") 'bh/widen)
(global-set-key (kbd "<f7>") 'bh/set-truncate-lines)
(global-set-key (kbd "<f8>") 'org-cycle-agenda-files)
(global-set-key (kbd "<f9> b") 'bbdb)
(global-set-key (kbd "<f9> c") 'calendar)
(global-set-key (kbd "<f9> f") 'boxquote-insert-file)
(global-set-key (kbd "<f9> g") 'gnus)
(global-set-key (kbd "<f9> h") 'bh/hide-other)
(global-set-key (kbd "<f9> i") 'info)

(global-set-key (kbd "<f9> I") 'bh/punch-in)
(global-set-key (kbd "<f9> O") 'bh/punch-out)

(global-set-key (kbd "<f9> o") 'bh/make-org-scratch)

(global-set-key (kbd "<f9> r") 'boxquote-region)
(global-set-key (kbd "<f9> s") 'bh/switch-to-scratch)

(global-set-key (kbd "<f9> t") 'bh/insert-inactive-timestamp)
(global-set-key (kbd "<f9> u") 'bh/untabify)

(global-set-key (kbd "<f9> v") 'visible-mode)
(global-set-key (kbd "<f9> SPC") 'bh/clock-in-last-task)
(global-set-key (kbd "C-<f9>") 'previous-buffer)
(global-set-key (kbd "C-x n r") 'narrow-to-region)
(global-set-key (kbd "C-<f10>") 'next-buffer)
(global-set-key (kbd "<f11>") 'org-clock-goto)
(global-set-key (kbd "C-<f11>") 'org-clock-in)
(global-set-key (kbd "C-s-<f12>") 'bh/save-then-publish)
(global-set-key (kbd "C-M-r") 'org-capture)
(global-set-key (kbd "C-c r") 'org-capture)

(defun bh/hide-other ()
  (interactive)
  (save-excursion
    (org-back-to-heading)
    (org-shifttab)
```

```
    (org-reveal)
    (org-cycle)))

(defun bh/set-truncate-lines ()
  "Toggle value of truncate-lines and refresh window display."
  (interactive)
  (setq truncate-lines (not truncate-lines))
  ;; now refresh window display (an idiom from simple.el):
  (save-excursion
    (set-window-start (selected-window)
                      (window-start (selected-window)))))

(defun bh/make-org-scratch ()
  (interactive)
  (find-file "/tmp/publish/scratch.org")
  (gnus-make-directory "/tmp/publish"))

(defun bh/switch-to-scratch ()
  (interactive)
  (switch-to-buffer "*scratch*"))

(defun bh/untabify ()
  (interactive)
  (untabify (point-min) (point-max)))
```

The main reason I have special key bindings (like `F11`, and `F12`) is so that the keys work in any mode. If I'm in the Gnus summary buffer then `C-u C-c C-x C-i` doesn't work, but the `C-F11` key combination does and this saves me time since I don't have to visit an org-mode buffer first just to clock in a recent task.

2 Tasks and States

I use one set of TODO keywords for all of my org files. Org-mode lets you define TODO keywords per file but I find it's easier to have a standard set of TODO keywords globally so I can use the same setup in any org file I'm working with.

The only exception to this is this document :) since I don't want `org-mode` hiding the `TODO` keyword when it appears in headlines. I've set up a dummy `#+SEQ_TODO: FIXME FIXED` entry at the top of this file just to

leave my TODO keyword untouched in this document.

2.1 TODO keywords

I use a light colour theme in emacs. I find this easier to read on bright sunny days.

Here are my TODO state keywords and colour settings:

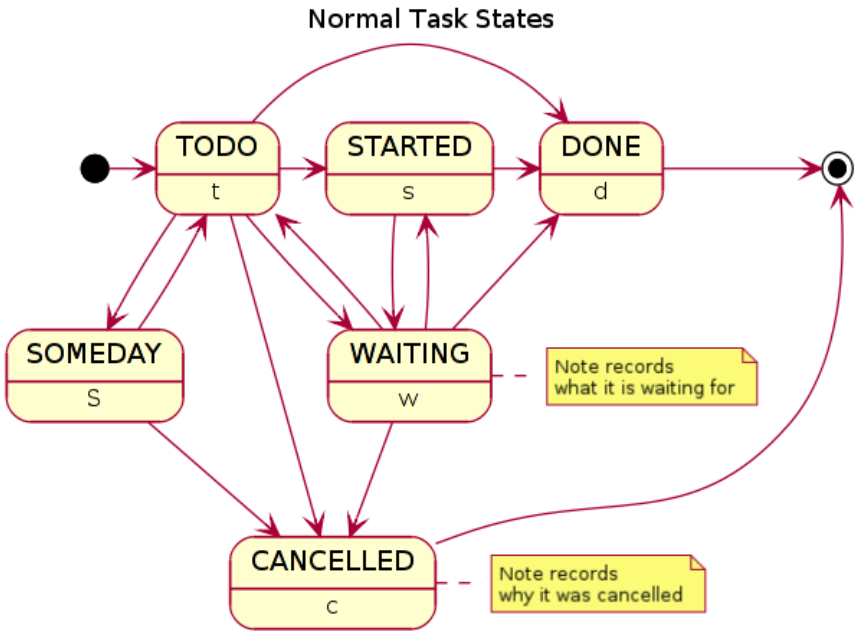
```
(setq org-todo-keywords
      (quote ((sequence "TODO(t)" "NEXT(n)" "STARTED(s)" "|" "DONE(d!/!)")
              (sequence "WAITING(w@/!)" "SOMEDAY(S!)" "|" "CANCELLED(c@/!)"
              (sequence "OPEN(O!)" "|" "CLOSED(C!)"))))

(setq org-todo-keyword-faces
      (quote (("TODO" :foreground "red" :weight bold)
              ("NEXT" :foreground "blue" :weight bold)
              ("STARTED" :foreground "blue" :weight bold)
              ("DONE" :foreground "forest green" :weight bold)
              ("WAITING" :foreground "orange" :weight bold)
              ("SOMEDAY" :foreground "magenta" :weight bold)
              ("CANCELLED" :foreground "forest green" :weight bold)
              ("OPEN" :foreground "blue" :weight bold)
              ("CLOSED" :foreground "forest green" :weight bold)
              ("PHONE" :foreground "forest green" :weight bold))))
```

2.1.1 Normal Task States

Normal tasks go through the sequence TODO -> DONE.

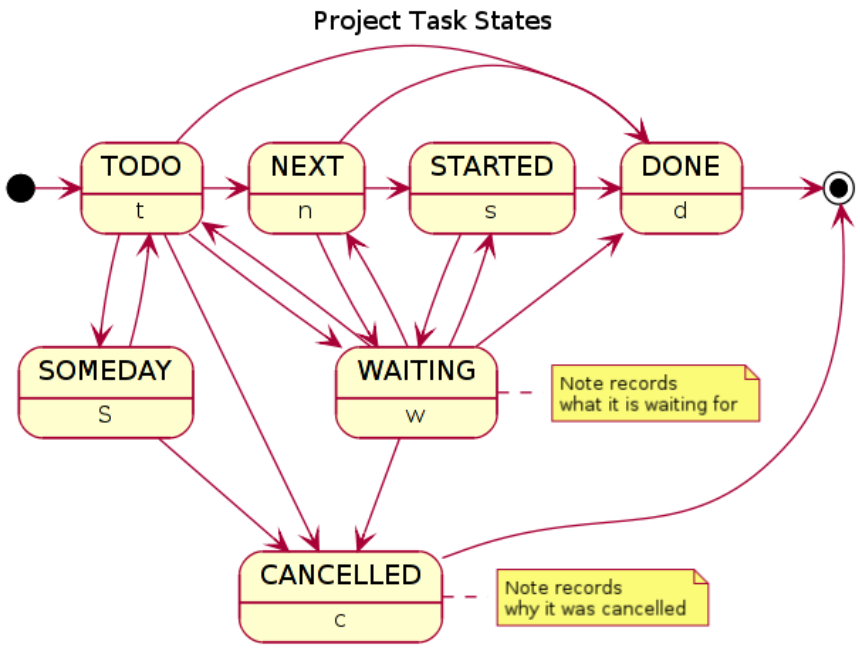
The following diagram shows the possible state transitions for a task.



2.1.2 Project Task States

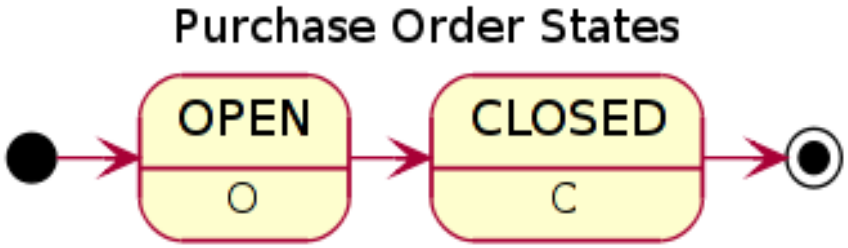
I use a lazy project definition. I don't like to bother with manually stating 'this is a project' and 'that is not a project'. For me a project definition is really simple. If a task has subtasks with a todo keyword then it's a project. That's it.

Projects can be defined at any level - just create a task with a todo state keyword that has at least one subtask also with a todo state keyword and you have a project. Projects use the same todo keywords as regular tasks with one exception - one subtask of a project needs to be marked `NEXT` or `STARTED` so the project is not on the stuck projects list.



2.1.3 Purchase Order Task States

Paying projects have a `Purchase Order` associated with it which is used for billing the client. The following states track purchase orders.



2.1.4 Phone Calls

Telephone calls are special. They are created in a done state by a capture task. The time of the call is recorded for as long as the capture task is active. If I need to look up other details and want to close the capture task early I can just `C-c C-c` to close the capture task (stopping the clock) and then `f9 SPC` to resume the clock in the phone call while I do other things.

2.2 Fast Todo Selection

Fast todo selection allows changing from any task todo state to any other state directly by selecting the appropriate key from the fast todo selection key menu. This is a great feature!

```
(setq org-use-fast-todo-selection t)
```

Changing a task state is done with

```
C-c C-t KEY
```

where `KEY` is the appropriate fast todo state selection key as defined in `org-todo-keywords`.

The setting

```
(setq org-treat-S-cursor-todo-selection-as-state-change nil)
```

allows changing todo states with S-left and S-right skipping all of the normal processing when entering or leaving a todo state. This cycles through the todo states but skips setting timestamps and entering notes which is very convenient when all you want to do is fix up the status of an entry.

2.3 TODO state triggers

I have a few triggers that automatically assign tags to tasks based on state changes. If a task moves to `CANCELLED` state then it gets a `CANCELLED` tag. Moving a `CANCELLED` task back to `TODO` removes the `CANCELLED` tag. These are used for filtering tasks in agenda views which I'll talk about later.

The triggers break down to the following rules:

- Moving a task to `CANCELLED` adds a `CANCELLED` tag

- Moving a task to `WAITING` adds a `WAITING` tag
- Moving a task to `SOMEDAY` adds a `WAITING` tag
- Moving a task to a done state removes a `WAITING` tag
- Moving a task to `TODO` removes `WAITING` and `CANCELLED` tags
- Moving a task to `NEXT` removes a `WAITING` tag
- Moving a task to `STARTED` removes a `WAITING` tag
- Moving a task to `DONE` removes `WAITING` and `CANCELLED` tags

The tags are used to filter tasks in the agenda views conveniently.

```
(setq org-todo-state-tags-triggers
      (quote (("CANCELLED" ("CANCELLED" . t))
              ("WAITING" ("WAITING" . t))
              ("SOMEDAY" ("WAITING" . t))
              (done ("WAITING"))
              ("TODO" ("WAITING") ("CANCELLED"))
              ("NEXT" ("WAITING"))
              ("STARTED" ("WAITING"))
              ("DONE" ("WAITING") ("CANCELLED")))))
```

3 Adding New Tasks Quickly with Org Capture

Org Capture mode replaces remember mode for capturing tasks and notes.

To add new tasks efficiently I use a minimal number of capture templates. I used to have lots of capture templates, one for each org-file. I'd start org-capture with C-M-r and then pick a template that filed the task under `* Tasks` in the appropriate file. This binding of C-M-r overrides the default emacs reverse regexp search but I rarely use that and can invoke it from the `M-x` command line if I really need it. I like `C-M-r` better than `C-c r` since it feels like a single keystroke instead of two separate keys and I've been using this so long that my fingers just do the right thing without really thinking about it.

I found I still needed to refile these capture tasks again to the correct location within the org-file so all of these different capture templates weren't really helping at all. Since then I've changed my workflow to use a minimal number of capture templates – I create the new task quickly and refile it

once. This also saves me from maintaining my org-capture templates when I add a new org file.

3.1 Capture Templates

When a new task needs to be added I categorize it into one of a few things:

- A phone call (p)
- A new task (t)
- A new note (n)
- An interruption (j)
- A new habit (h)

and pick the appropriate capture task.
Here is my setup for org-capture

```
(setq org-default-notes-file "~/git/org/refile.org")

;; I use C-M-r to start capture mode
(global-set-key (kbd "C-M-r") 'org-capture)
;; I use C-c r to start capture mode when using SSH from my Android phone
(global-set-key (kbd "C-c r") 'org-capture)

;; Capture templates for: TODO tasks, Notes, appointments, phone calls, and
(setq org-capture-templates
      (quote (("t" "todo" entry (file "~/git/org/refile.org")
               "* TODO %?\n%U\n%a\n  %i" :clock-in t :clock-resume t)
              ("n" "note" entry (file "~/git/org/refile.org")
               "* %? :NOTE:\n%U\n%a\n  %i" :clock-in t :clock-resume t)
              ("j" "Journal" entry (file+datetree "~/git/org/diary.org")
               "* %?\n%U\n  %i" :clock-in t :clock-resume t)
              ("w" "org-protocol" entry (file "~/git/org/refile.org")
               "* TODO Review %c\n%U\n  %i" :immediate-finish t)
              ("p" "Phone call" entry (file "~/git/org/refile.org")
               "* PHONE %? :PHONE:\n%U" :clock-in t :clock-resume t)
              ("h" "Habit" entry (file "~/git/org/refile.org")
               "* NEXT %?\n%U\n%a\nSCHEDULED: %t .+1d/3d\n:PROPERTIES:\n:ST
```

The `%i` in the templates inserts any text in the kill ring as part of the capture task. This is intentionally indented from the rest of the capture task details so that I can include text that starts with '* ' in column 1 without generating a new headline.

Capture mode now handles automatically clocking in and out of a capture task. This all works out of the box now without special hooks. When I start a capture mode task the task is clocked in as specified by `:clock-in t` and when the task is filed with `C-c C-c` the clock resumes on the original clocking task.

The quick clocking in and out of capture mode tasks (often it takes less than a minute to capture some new task details) can leave empty clock drawers in my tasks which aren't really useful. Since I remove clocking lines with 0:00 length I end up with a clock drawer like this:

```
* TODO New Capture Task
  :LOGBOOK:
  :END:
  [2010-05-08 Sat 13:53]
```

I have the following setup to remove these empty `LOGBOOK` drawers if they occur.

```
;; Remove empty LOGBOOK drawers on clock out
(defun bh/remove-empty-drawer-on-clock-out ()
  (interactive)
  (save-excursion
    (beginning-of-line 0)
    (org-remove-empty-drawer-at "LOGBOOK" (point))))

(add-hook 'org-clock-out-hook 'bh/remove-empty-drawer-on-clock-out 'append)
```

3.2 Separate file for Capture Tasks

I have a single org file which is the target for my capture templates.

I store notes, tasks, phone calls, and org-protocol tasks in `refile.org`. I used to use multiple files but found that didn't really have any advantage over a single file.

Normally this file is empty except for a single line at the top which creates a `REFILE` tag for anything in the file.

The file has a single permanent line at the top like this

```
#+FILETAGS: REFILE
```

3.3 Capture Tasks is all about being FAST

Okay I'm in the middle of something and oh yeah - I have to remember to do that. I don't stop what I'm doing. I'm probably clocking a project I'm working on and I don't want to lose my focus on that but I can't afford to forget this little thing that just came up.

So what do I do? Hit `C-M-r` to start capture mode and select `t` since it's a new task and I get a buffer like this:

```
** TODO
   [2010-08-05 Thu 21:06]
   [[file:~/git/org-mode-doc/org-mode.org::*Capture%20Tasks%20is%20all%20about%20being
```

Enter the details of the TODO item and `C-c C-c` to file it away in refile.org and go right back to what I'm really working on secure in the knowledge that that item isn't going to get lost and I don't have to think about it anymore at all now.

The amount of time I spend entering the captured note is clocked. The capture templates are set to automatically clock in and out of the capture task. This is great for interruptions and telephone calls too.

4 Refiling Tasks

Refiling tasks is easy. After collecting a bunch of new tasks in my refile.org file using capture mode I need to move these to the correct org file and topic. All of my active org-files are in my `org-agenda-files` variable and contribute to the agenda.

I collect capture tasks in refile.org for up to a week. These now stand out daily on my block agenda and I usually refile them during the day. I like to keep my refile task list empty.

4.1 Refile Setup

To refile tasks in org you need to tell it where you want to refile things.

In my setup I let any file in `org-agenda-files` and the current file contribute to the list of valid refile targets.

I've recently moved to using IDO to complete targets directly. I find this to be faster than my previous complete in steps setup. At first I didn't like IDO but after reviewing the documentation again and learning about `C-SPC` to limit target searches I find it is much better than my previous complete-in-steps setup. Now when I want to refile something I do `C-c C-w` to start

the refile process, then type something to get some matching targets, then `C-SPC` to restrict the matches to the current list, then continue searching with some other text to find the target I need. `C-j` also selects the current completion as the final target. I like this a lot.

I now exclude DONE state tasks as valid refile targets. This helps to keep the refile target list to a reasonable size.

Here is my refile configuration:

```
; Targets include this file and any file contributing to the agenda - up to
(setq org-refile-targets (quote ((nil :maxlevel . 3)
                                 (org-agenda-files :maxlevel . 3))))

; Stop using paths for refile targets - we file directly with IDO
(setq org-refile-use-outline-path nil)

; Targets complete directly with IDO
(setq org-outline-path-complete-in-steps nil)

; Allow refile to create parent tasks with confirmation
(setq org-refile-allow-creating-parent-nodes (quote confirm))

; Use IDO for both buffer and file completion and ido-everywhere to t
(setq org-completion-use-ido t)
(setq ido-everywhere t)
(setq ido-max-directory-size 100000)
(ido-mode (quote both))

;;;; Refile settings
; Exclude DONE state tasks from refile targets
(defun bh/verify-refile-target ()
  "Exclude todo keywords with a done state from refile targets"
  (not (member (nth 2 (org-heading-components)) org-done-keywords)))

(setq org-refile-target-verify-function 'bh/verify-refile-target)
```

To refile a task to my `norang.org` file under `System Maintenance` I just put the cursor on the task and hit `C-c C-w` and enter `nor C-SPC sys RET` and it's done. IDO completion makes locating targets a snap.

4.2 Refiling Tasks

Tasks to refile are in their own section of the block agenda. To find tasks to refile I run my agenda view with `F12 SPC` and scroll down to second section of the block agenda: `Tasks to Refile`. This view shows all tasks (even ones marked in a `done` state). Alternatively I just use `F12 r` on my slower Eee PC.

Bulk refiling in the agenda works very well for multiple tasks going to the same place. Just mark the tasks with `m` and then `B r` to refile all of them to a new location. Occasionally I'll also refile tasks as subtasks of the current clocking task using `C-2 C-c C-w` from the `refile.org` file.

Refiling all of my tasks tends to take less than a minute so I normally do this a couple of times a day.

4.3 Refiling Notes

I keep a `* Notes` headline in most of my org-mode files. Notes have a `NOTE` tag which is created by the capture template for notes. This allows finding notes across multiple files easily using the agenda search functions.

Notes created by capture tasks go first to `refile.org` and are later refiled to the appropriate project file. Some notes that are project related get filed to the appropriate project instead of under the catchall `* NOTES` task. Generally these types of notes are specific to the project and not generally useful – so removing them from the notes list when the project is archived makes sense.

4.4 Refiling Phone Calls

Phone calls are handled using capture mode. I time my calls using the capture mode template settings to clock in and out the capture task while the phone call is in progress.

Phone call tasks collect in `refile.org` and are later refiled to the appropriate location. Some phone calls are billable and we want these tracked in the appropriate category.

5 Custom agenda views

I now have one block agenda view that has everything on it. I also keep separate single view agenda commands for use on my slower Eee PC - since it takes prohibitively long to generate my block agenda on that slow machine.

I'm striving to simplify my layout with everything at my fingertips in a single agenda on my workstation which is where I spend the bulk of my time.

Most of my old custom agenda views were rendered obsolete when filtering functionality was added to the agenda in newer versions of `org-mode` and now with block agenda functionality I can combine everything into a single view.

Custom agenda views are used for:

- Single block agenda shows the following
  - overview of today
  - Finding tasks to be refiled
  - Finding stuck projects
  - Finding NEXT tasks to work on
  - Reviewing projects
  - Show all TODO state tasks
  - Finding tasks waiting on something
  - Findings tasks to be archived
- Finding notes
- Viewing habits

If I want just today's calendar view then `F12 a` is still faster than generating the block agenda - especially if I want to view a week or month's worth of information. In that case the extra detail on the block agenda view is never really needed and I don't want to spend time waiting for it to be generated.

5.1 Setup

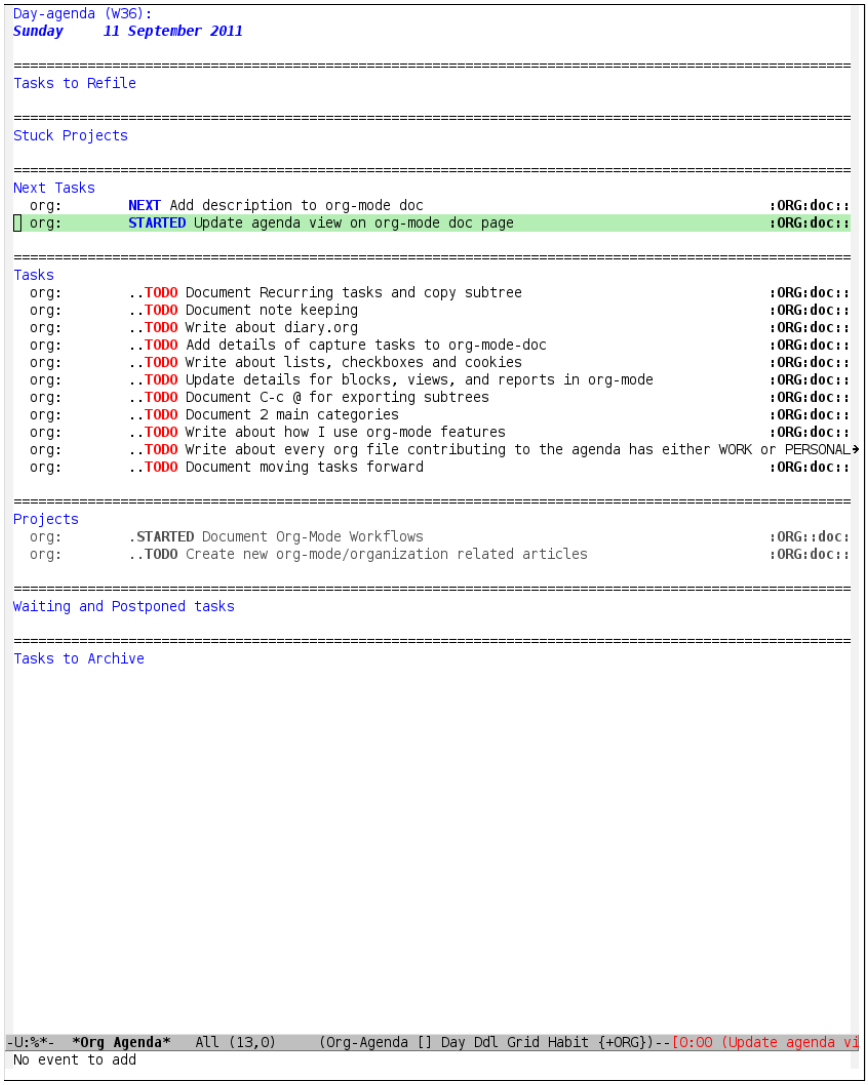
```
;; Do not dim blocked tasks
(setq org-agenda-dim-blocked-tasks nil)

;; Custom agenda command definitions
(setq org-agenda-custom-commands
      (quote (("N" "Notes" tags "NOTE"
                ((org-agenda-overriding-header "Notes")
                 (org-tags-match-list-sublevels t)))
```

```
("h" "Habits" tags-todo "STYLE=\"habit\""
 ((org-agenda-overriding-header "Habits")
  (org-agenda-sorting-strategy
   '(todo-state-down effort-up category-keep))))
(" " "Agenda"
 ((agenda "" nil)
  (tags "REFILE"
        ((org-agenda-overriding-header "Notes and Tasks to R
         (org-agenda-overriding-header "Tasks to Refile")))
  (tags-todo "-CANCELLED/!"
             ((org-agenda-overriding-header "Stuck Projects")
              (org-tags-match-list-sublevels 'indented)
              (org-agenda-skip-function 'bh/skip-non-stuck-p
  (tags-todo "-WAITING-CANCELLED/!NEXT|STARTED"
             ((org-agenda-overriding-header "Next Tasks")
              (org-agenda-skip-function 'bh/skip-projects-and
              (org-agenda-todo-ignore-scheduled t)
              (org-agenda-todo-ignore-deadlines t)
              (org-tags-match-list-sublevels t)
              (org-agenda-sorting-strategy
               '(todo-state-down effort-up category-keep))))
  (tags-todo "-REFILE-CANCELLED/!-NEXT-STARTED-WAITING"
             ((org-agenda-overriding-header "Tasks")
              (org-agenda-skip-function 'bh/skip-projects-and
              (org-tags-match-list-sublevels 'indented)
              (org-agenda-todo-ignore-scheduled t)
              (org-agenda-todo-ignore-deadlines t)
              (org-agenda-sorting-strategy
               '(category-keep))))
  (tags-todo "-CANCELLED/!"
             ((org-agenda-overriding-header "Projects")
              (org-agenda-skip-function 'bh/skip-non-projects
              (org-tags-match-list-sublevels 'indented)
              (org-agenda-sorting-strategy
               '(category-keep))))
  (todo "WAITING|SOMEDAY"
        ((org-agenda-overriding-header "Waiting and Postponed
         (org-agenda-skip-function 'bh/skip-projects-and-hab
  (tags "-REFILE/"
        ((org-agenda-overriding-header "Tasks to Archive")
```

```
          (org-agenda-skip-function 'bh/skip-non-archivable-ta
 nil)
("r" "Tasks to Refile" tags "REFILE"
 ((org-agenda-overriding-header "Notes and Tasks to Refile")
  (org-agenda-overriding-header "Tasks to Refile")))
("#" "Stuck Projects" tags-todo "-CANCELLED/!"
 ((org-agenda-overriding-header "Stuck Projects")
  (org-tags-match-list-sublevels 'indented)
  (org-agenda-skip-function 'bh/skip-non-stuck-projects)))
("n" "Next Tasks" tags-todo "-WAITING-CANCELLED/!NEXT|STARTED
 ((org-agenda-overriding-header "Next Tasks")
  (org-agenda-skip-function 'bh/skip-projects-and-habits)
  (org-agenda-todo-ignore-scheduled t)
  (org-agenda-todo-ignore-deadlines t)
  (org-tags-match-list-sublevels t)
  (org-agenda-sorting-strategy
   '(todo-state-down effort-up category-keep))))
("R" "Tasks" tags-todo "-REFILE-CANCELLED/!-NEXT-STARTED-WAI
 ((org-agenda-overriding-header "Tasks")
  (org-agenda-skip-function 'bh/skip-projects-and-habits)
  (org-tags-match-list-sublevels 'indented)
  (org-agenda-sorting-strategy
   '(category-keep))))
("p" "Projects" tags-todo "-CANCELLED/!"
 ((org-agenda-overriding-header "Projects")
  (org-agenda-skip-function 'bh/skip-non-projects)
  (org-tags-match-list-sublevels 'indented)
  (org-agenda-sorting-strategy
   '(category-keep))))
("w" "Waiting Tasks" todo "WAITING|SOMEDAY"
 ((org-agenda-overriding-header "Waiting and Postponed tasks"
 (org-agenda-skip-function 'bh/skip-projects-and-habits))
("A" "Tasks to Archive" tags "-REFILE/"
 ((org-agenda-overriding-header "Tasks to Archive")
  (org-agenda-skip-function 'bh/skip-non-archivable-tasks))))
```

My block agenda view looks like this:



I generally work top-down on the agenda. Things with deadlines and scheduled dates (planned to work on today or earlier) show up in the agenda at the top. When searching for tasks in the agenda I disable display of child tasks with the following setting:

```
(setq org-tags-match-list-sublevels nil)
```

This keeps the list of tasks I'm looking at to a reasonable size. I can always display child tasks for any specific task I want simply by visiting it in the org buffer.

My day goes generally like this:

- Punch in (this starts the clock on the default task)
- Look at the agenda and make a mental note of anything important to deal with today
- Read email and news
  - create notes, and tasks for things that need responses with org-capture
- Check refile tasks and respond to emails
- Look at my agenda and work on important tasks for today
  - Clock it in
  - Work on it until it is `DONE` or it gets interrupted
- work on tasks
- Make journal entries (`C-M-r j`) for interruptions
- Punch out for lunch and punch back in after lunch
- work on more tasks
- Refile tasks to empty the list
  - Tag tasks to be refiled with `m` collecting all tasks for the same target
  - Bulk refile the tasks to the target location with `B r`
  - Repeat (or refile individually with `C-c C-w`) until all refile tasks are gone
- Mark habits done today as DONE
- Punch out at the end of the work day

5.2 What do I work on next?

Start with deadlines and tasks scheduled today or earlier from the daily agenda view. Then move on to tasks in the `Next Tasks` list in the block agenda view.

When I look for a new task to work on I generally hit `F12 SPC` to get the block agenda and follow this order:

- Pick something off today's agenda
  - deadline for today (do this first - it's not late yet)
  - deadline in the past (it's already late)
  - a scheduled task for today (or in the past)
  - deadline that is coming up soon
- pick a NEXT task
- If you run out of items to work on look for a NEXT task in the current context pick a task from the Tasks list of the current project.

5.2.1 Why keep it all on the `NEXT` list?

I've moved to a more GTD way of doing things. Now I just use a `NEXT` list. Only projects get tasks with `NEXT` keywords since stuck projects initiate the need for marking or creating `NEXT` tasks. A `NEXT` task is something that is available to work on *now*, it is the next logical step in some project.

I used to have a special keyword `ONGOING` for things that I do a lot and want to clock but never really start/end. I had a special agenda view for `ONGOING` tasks that I would pull up to easily find the thing I want to clock.

Since then I've moved away from using the `ONGOING` todo keyword. Having an agenda view that shows `NEXT` tasks makes it easy to pick the thing to clock - and I don't have to remember if I need to look in the `ONGOING` list or the `NEXT` list when looking for the task to clock-in. The `NEXT` list is basically 'what is current' - any task that moves a project forward. I want to find the thing to work on as fast as I can and actually do work on it - not spend time hunting through my org files for the task that needs to be clocked-in.

To drop a task off the `NEXT` list simply move it back to the `TODO` state.

5.3 Reading email, newsgroups, and conversations on IRC

When reading email, newsgroups, and conversations on IRC I just let the default task (normally `** Organization`) clock the time I spend on these tasks. To read email I go to Gnus and read everything in my inboxes. If there are emails that require a response I use org-capture to create a new task with a heading of 'Respond to <user>' for each one. This automatically links to the email in the task and makes it easy to find later. Some emails are quick to respond to and some take research and a significant amount of time to complete. I clock each one in it's own task just in case I need that clocked time later.

Next, I go to my newly created tasks to be refiled from the block agenda with `F12 a` and clock in an email task and deal with it. Repeat this until all of the 'Respond to <user>' tasks are marked `DONE`.

I read email and newgroups in Gnus so I don't separate clocked time for quickly looking at things. If an article has a useful piece of information I want to remember I create a note for it with `C-M-r n` and enter the topic and file it. This takes practically no time at all and I know the note is safely filed for later retrieval. The time I spend in the capture buffer is clocked with that capture note.

5.4 Filtering

So many tasks, so little time. I have hundreds of tasks at any given time (373 right now). There is so much stuff to look at it can be daunting. This is where agenda filtering saves the day.

It's 11:53AM and I'm in work mode just before lunch. I don't want to see tasks that are not work related right now. I also don't want to work on a big project just before lunch... so I need to find small tasks that I can knock off the list.

How do we do this? Get a list of NEXT tasks from the block agenda and then narrow it down with filtering. Tasks are ordered in the NEXT agenda view by estimated effort so the short tasks are first – just start at the top and work your way down. I can limit the displayed agenda tasks to those estimates of 10 minutes or less with `/ + 1` and I can pick something that fits the minutes I have left before I take off for lunch.

5.4.1 Automatically removing context based tasks with / RET

`/ RET` in the agenda is really useful. This awesome feature was added to org-mode by John Wiegley. It removes tasks automatically by filtering based on a user-provided function.

I work from home and set up my day as follows:

- On weekdays 8am-12am, 1pm-5pm I'm working (@office)
- My son (Mark) is available on weekdays before school 8am-9am and after school to bedtime 4pm-8pm (MARK), and weekends 10am-8pm
- Personal tasks are done outside working hours (PERSONAL)
- Work tasks are done during working hours (WORK)

I have the following setup to allow `/ RET` to filter tasks based on what the computer determines my current context to be at the time I run the `/ RET` filter command.

```
(defun bh/weekday-p ()
  (let ((wday (nth 6 (decode-time))))
    (and (< wday 6)
         (> wday 0))))

(defun bh/working-p ()
  (let ((hour (nth 2 (decode-time))))
    (and (bh/weekday-p)
         (or (and (>= hour 8) (<= hour 11))
             (and (>= hour 13) (<= hour 16))))))

(defun bh/org-auto-exclude-function (tag)
  "Automatic task exclusion in the agenda with / RET"
  (and (cond
        ((string= tag "@farm")
         t)
        ((or (string= tag "@errand") (string= tag "phone"))
         (let ((hour (nth 2 (decode-time))))
           (or (< hour 8) (> hour 21))))
        (t
         (if (bh/working-p)
             (setq tag "PERSONAL")
           (setq tag "WORK"))
         (unless (member (concat "-" tag) org-agenda-filter)
           tag)))
       (concat "-" tag)))

(setq org-agenda-auto-exclude-function 'bh/org-auto-exclude-function)
```

This lets me filter tasks with just `/ RET` on the agenda which removes tasks I'm not supposed to be working on now from the list of returned results.

This helps to keep my agenda clutter-free.

6 Time Clocking

Okay, I admit it. I'm a clocking fanatic.

I clock everything (well almost everything). Org-mode makes this really easy. I'd rather clock too much stuff than not enough so I find it's easier to get in the habit of clocking everything.

As an example of what I mean my clock data for April 20, 2009 shows 14 hours 19 minutes of clocked time (which included 3 hours and 17 minutes of painting my basement.) My clocked day started at 6:57AM and ended at 23:11PM. I have only a few holes in my clocked day (where I wasn't clocking anything):

Missing Clock Data
16:14-16:53
16:55-17:19
18:00-18:52

This makes it possible to look back at the day and see where I'm spending too much time, or not enough time on specific projects.

Without clocking data it's hard to tell what you did after the fact.

I now use the concept of punching in and punching out at the start and end of my work day. This defines a default task to clock time on whenever the clock would normally stop. I found that with the default org-mode setup I would lose clocked minutes during the day, a minute here, a minute there, and that all adds up. This is especially true if you write notes when moving to a DONE state - in this case the clock normally stops before you have composed the note – and good notes take a few minutes to write.

My clocking setup basically works like this:

- Punch in (start the clock)
  - This identifies a task that is the default task to clock in whenever the clock normally stops
- Clock in tasks normally, and let moving to a DONE state clock out
  - clocking out automatically clocks time on a new task
- Continue clocking whatever tasks you work on
- Punch out (stop the clock)

I'm free to change the default task multiple times during the day. If I punch-in with a prefix on a task in `Project X` then that task automatically becomes the default task and all clocked time goes on that project until I either punch out or punch in some other task.

My org files look like this:
`todo.org`:

```
#+FILETAGS: PERSONAL
...
* Tasks
** Organization
:PROPERTIES:
:CLOCK_MODELINE_TOTAL: today
:ID:       eb155a82-92b2-4f25-a3c6-0304591af2f9
:END:
...
```

If I am working on some task, then I simply clock in on the task. Clocking out moves the clock up to a parent task with a todo keyword (if any) which keeps the clock time in the same subtree. If there is no parent task with a todo keyword then the clock moves back to the default clocking task until I punch out or clock in some other task. When an interruption occurs I start a capture task which keeps clocked time on the interruption task until I close it with C-c C-c.

This works really well for me.

For example, consider the following org file:

```
* TODO Project A
** NEXT TASK 1
** TODO TASK 2
** TODO TASK 3
* Tasks
** TODO Some miscellaneous task
```

I'll work on this file in the following sequence:

1. I punch in with `F9-I` at the start of my day

   That clocks in the `Organization` task by id in my `todo.org` file.

2. `F12-SPC` to review my block agenda

   Pick 'TODO Some miscellaneous task' to work on next and clock that in with `I` The clock is now on 'TODO Some miscellaneous task'

3. I complete that task and mark it done with `C-c C-t d`

   This stops the clock and moves it back to the `Organization` task.

4. Now I want to work on `Project A` so I clock in `Task 1`

   I work on Task 1 and mark it `DONE`. This clocks out `Task 1` and moves the clock to `Project A`. Now I work on `Task 2` and clock that in.

The entire time I'm working on and clocking some subtask of `Project A` all of the clock time in the interval is applied somewhere to the `Project A` tree. When I eventually mark `Project A` done then the clock will move back to the default organization task.

6.1 Clock Setup

To get started we need to punch in which clocks in the default task and keeps the clock running. This is now simply a matter of punching in the clock with `F9 I`. You can do this anywhere. Clocking out will now clock in the parent task (if there is one with a todo keyword) or clock in the default task if not parent exists.

Keeping the clock running when moving a subtask to a `DONE` state means clocking continues to apply to the project task. I can pick the next task from the parent and clock that in without losing a minute or two while I'm deciding what to work on next.

I keep clock times, state changes, and other notes in the `:LOGBOOK:` drawer.

I have the following org-mode settings for clocking:

```
;;
;; Resume clocking tasks when emacs is restarted
(org-clock-persistence-insinuate)
;;
;; Small windows on my Eee PC displays only the end of long lists which isn
(setq org-clock-history-length 10)
;; Resume clocking task on clock-in if the clock is open
(setq org-clock-in-resume t)
;; Change task to STARTED when clocking in
(setq org-clock-in-switch-to-state 'bh/clock-in-to-started)
;; Separate drawers for clocking and logs
(setq org-drawers (quote ("PROPERTIES" "LOGBOOK")))
;; Save clock data and state changes and notes in the LOGBOOK drawer
(setq org-clock-into-drawer t)
;; Sometimes I change tasks I'm clocking quickly - this removes clocked tas
(setq org-clock-out-remove-zero-time-clocks t)
;; Clock out when moving task to a done state
```

```
(setq org-clock-out-when-done t)
;; Save the running clock and all clock history when exiting Emacs, load it
(setq org-clock-persist (quote history))
;; Enable auto clock resolution for finding open clocks
(setq org-clock-auto-clock-resolution (quote when-no-clock-is-running))
;; Include current clocking task in clock reports
(setq org-clock-report-include-clocking-task t)

(setq bh/keep-clock-running nil)

(defun bh/clock-in-to-started (kw)
  "Switch task from TODO or NEXT to STARTED when clocking in.
Skips capture tasks."
  (if (and (member (org-get-todo-state) (list "TODO" "NEXT"))
           (not (and (boundp 'org-capture-mode) org-capture-mode)))
      "STARTED"))

(defun bh/find-project-task ()
  "Move point to the parent (project) task if any"
  (let ((parent-task (save-excursion (org-back-to-heading) (point))))
    (while (org-up-heading-safe)
      (when (member (nth 2 (org-heading-components)) org-todo-keywords-1)
        (setq parent-task (point))))
    (goto-char parent-task)
    parent-task))

(add-hook 'org-agenda-mode-hook '(lambda () (org-defkey org-agenda-mode-map

(defun bh/set-agenda-restriction-lock (arg)
  "Set restriction lock to current subtree or file if prefix is specified"
  (interactive "p")
  (let* ((pom (org-get-at-bol 'org-hd-marker))
         (tags (org-with-point-at pom (org-get-tags-at))))
    (let ((restriction-type (if (equal arg 4) 'file 'subtree)))
      (cond
       ((equal major-mode 'org-agenda-mode)
        (org-with-point-at pom
          (org-agenda-set-restriction-lock restriction-type)))
       ((and (equal major-mode 'org-mode) (org-before-first-heading-p))
        (org-agenda-set-restriction-lock 'file))
```

```
        (t
         (org-with-point-at pom
           (org-agenda-set-restriction-lock restriction-type)))))))

(defun bh/punch-in (arg)
  "Start continuous clocking and set the default task to the
selected task.  If no task is selected set the Organization task
as the default task."
  (interactive "p")
  (setq bh/keep-clock-running t)
  (if (equal major-mode 'org-agenda-mode)
      ;;
      ;; We're in the agenda
      ;;
      (let* ((marker (org-get-at-bol 'org-hd-marker))
             (tags (org-with-point-at marker (org-get-tags-at))))
        (if (and (eq arg 4) tags)
            (org-agenda-clock-in '(16))
          (bh/clock-in-organization-task-as-default)))
    ;;
    ;; We are not in the agenda
    ;;
    (save-restriction
      (widen)
      ; Find the tags on the current task
      (if (and (equal major-mode 'org-mode) (not (org-before-first-heading-
          (org-clock-in '(16))
        (bh/clock-in-organization-task-as-default)))))

(defun bh/punch-out ()
  (interactive)
  (setq bh/keep-clock-running nil)
  (when (org-clock-is-active)
    (org-clock-out))
  (org-agenda-remove-restriction-lock))

(defun bh/clock-in-default-task ()
  (save-excursion
    (org-with-point-at org-clock-default-task
      (org-clock-in))))
```

```
(defun bh/clock-in-parent-task ()
  "Move point to the parent (project) task if any and clock in"
  (let ((parent-task))
    (save-excursion
      (save-restriction
        (widen)
        (while (and (not parent-task) (org-up-heading-safe))
          (when (member (nth 2 (org-heading-components)) org-todo-keywords-
            (setq parent-task (point))))
        (if parent-task
            (org-with-point-at (or parent-task)
              (org-clock-in))
          (when bh/keep-clock-running
            (bh/clock-in-default-task)))))))

(defvar bh/organization-task-id "eb155a82-92b2-4f25-a3c6-0304591af2f9")

(defun bh/clock-in-organization-task-as-default ()
  (interactive)
  (save-restriction
    (widen)
    (org-with-point-at (org-id-find bh/organization-task-id 'marker)
      (org-clock-in '(16)))))

(defun bh/clock-out-maybe ()
  (when (and bh/keep-clock-running
             (not org-clock-clocking-in)
             (marker-buffer org-clock-default-task)
             (not org-clock-resolving-clocks-due-to-idleness))
    (bh/clock-in-parent-task)))

(add-hook 'org-clock-out-hook 'bh/clock-out-maybe 'append)
```

I used to clock in tasks by ID using the following function but with the new punch-in and punch-out I don't need these as much anymore. `f9-SPC` calls `bh/clock-in-last-task` which switches the clock back to the previously clocked task.

```
(require 'org-id)
(defun bh/clock-in-task-by-id (id)
```

```
  "Clock in a task by id"
  (save-restriction
    (widen)
    (org-with-point-at (org-id-find id 'marker)
      (org-clock-in nil))))

(defun bh/clock-in-last-task (arg)
  "Clock in the interrupted task if there is one
Skip the default task and get the next one.
A prefix arg forces clock in of the default task."
  (interactive "p")
  (let ((clock-in-to-task
         (cond
          ((eq arg 4) org-clock-default-task)
          ((and (org-clock-is-active)
                (equal org-clock-default-task (cadr org-clock-history)))
           (caddr org-clock-history))
          ((org-clock-is-active) (cadr org-clock-history))
          ((equal org-clock-default-task (car org-clock-history)) (cadr org
          (t (car org-clock-history)))))
    (org-with-point-at clock-in-to-task
      (org-clock-in nil))))
```

6.2 Clocking in

When I start or continue working on a task I clock it in with any of the following:

- `C-c C-x C-i`
- `I` in the agenda
- `I` speed key on the first character of the heading line
- `f9 I` while on the task in the agenda
- `f9 I` while in the task in an org file

6.2.1 Setting a default clock task

I have a default `** Organization` task in my todo.org file that I tend to put miscellaneous clock time on. This is the task I clock in on when I punch

in at the start of my work day with `F9-I`. While reorganizing my org-files, reading email, clearing my inbox, and doing other planning work that isn't for a specific project I'll clock in this task. Punching-in anywhere clocks in this Organization task as the default task.

If I want to change the default clocking task I just visit the new task in any org buffer and clock it in with `C-u C-u C-c C-x C-i`. Now this new task that collects miscellaneous clock minutes when the clock would normally stop.

You can quickly clock in the default clocking task with `C-u C-c C-x C-i d`. Another option is to repeatedly clock out so the clock moves up the project tree until you clock out the top-level task and the clock moves to the default task.

6.2.2 Using the clock history to clock in old tasks

You can use the clock history to restart clocks on old tasks you've clocked or to jump directly to a task you have clocked previously. I use this mainly to clock in whatever got interrupted by something.

Consider the following scenario:

- You are working on and clocking `Task A` (Organization)
- You get interrupted and switch to `Task B` (Document my use of org-mode)
- You complete `Task B` (Document my use of org-mode)
- Now you want to go back to `Task A` (Organization) again to continue

This is easy to deal with.

1. Clock in `Task A`, work on it
2. Go to `Task B` (or create a new task) and clock it in
3. When you are finished with `Task B` hit `C-u C-c C-x C-i i`

This displays a clock history selection window like the following and selects the interrupted `[i]` entry.

Clock history selection buffer for C-u C-c C-x C-i

```
Default Task
[d] norang          Organization                          <-- Task B
The task interrupted by starting the last one
[i] norang          Organization                          <-- Task B
Current Clocking Task
[c] org             NEXT Document my use of org-mode      <-- Task A
Recent Tasks
[1] org             NEXT Document my use of org-mode      <-- Task A
[2] norang          Organization                          <-- Task B
...
[Z] org             DONE Fix default section links        <-- 35 clock task entries ag
```

6.3 Clock Everything - Create New Tasks

In order to clock everything you need a task for everything. That's fine for planned projects but interruptions inevitably occur and you need some place to record whatever time you spend on that interruption.

To deal with this we create a new capture task to record the thing we are about to do. The workflow goes something like this:

- You are clocking some task and an interruption occurs
- Create a quick capture task journal entry `C-M-r j`
- Type the heading
- go do that thing (eat lunch, whatever)
- file it `C-c C-c`, this restores the clock back to the previous clocking task
- clock something else in or continue with the current clocking task

This means you can ignore the details like where this task really belongs in your org file layout and just get on with completing the thing. Refiling a bunch of tasks later in a group when it is convenient to refile the tasks saves time in the long run.

If it's a one-shot uninteresting task (like a coffee break) I create a capture journal entry for it that goes to the diary.org date tree. If it's a task that actually needs to be tracked and marked done, and applied to some project then I create a capture task instead which files it in refile.org.

6.4 Finding tasks to clock in

To find a task to work on I use one of the following options (generally listed most frequently used first)

- Use the clock history C-u C-c C-x C-i Go back to something I was clocking that is not finished
- Pick something off today's block agenda `SCHEDULED` or `DEADLINE` items that need to be done soon
- Pick something off the `NEXT` tasks agenda view Work on some unfinished task to move to completion
- Pick something off the other task list
- Use an agenda view with filtering to pick something to work on

Punching in on the task you select will restrict the agenda view to that project so you can focus on just that thing for some period of time.

6.5 Editing clock entries

Sometimes it is necessary to edit clock entries so they reflect reality. I find I do this for maybe 2-3 entries in a week.

Occassionally I cannot clock in a task on time because I'm away from my computer. In this case the previous clocked task is still running and counts time for both tasks which is wrong.

I make a note of the time and then when I get back to my computer I clock in the right task and edit the start and end times to correct the clock history.

To visit the clock line for an entry quickly use the agenda log mode. `F12 a l` shows all clock lines for today. I use this to navigate to the appropriate clock lines quickly. F11 goes to the current clocked task but the agenda log mode is better for finding and visiting older clock entries.

Use `F12 a l` to open the agenda in log mode and show only logged clock times. Move the cursor down to the clock line you need to edit and hit `TAB` and you're there.

To edit a clock entry just put the cursor on the part of the date you want to edit (use the keyboard not the mouse - since the clicking on the timestamp with the mouse goes back to the agenda for that day) and hit the `S-<up arrow>` or `S-<down arrow>` keys to change the time.

The following setting makes time editing use discrete minute intervals (no rounding) increments:

```
(setq org-time-stamp-rounding-minutes (quote (1 1)))
```

Editing the time with the shift arrow combination also updates the total for the clock line which is a nice convenience.

I always check that I haven't created task overlaps when fixing time clock entries by viewing them with log mode on in the agenda. There is a new view in the agenda for this – just hit `v c` in the daily agenda and clock gaps and overlaps are identified.

I want my clock entries to be as accurate as possible.

The following setting shows 1 minute clocking gaps.

```
(setq org-agenda-clock-consistency-checks
      (quote (:max-duration "4:00"
              :min-duration 0
              :max-gap 0
              :gap-ok-around ("4:00"))))
```

6.6 Automatically clocking tasks

I spend time on an open source project called BZFlag. During work for releases I want to clock the time I spend testing the new BZFlag client. I have a key binding in my window manager that runs a script which starts the clock on my testing task, runs the BZFlag client, and on exit resumes the clock on the previous clocking task.

The testing task has an ID property of `dcf55180-2a18-460e-8abb-a9f02f0893be` and the following elisp code starts the clock on this task.

```
(defun bh/clock-in-bzflagt-task ()
  (interactive)
  (bh/clock-in-task-by-id "dcf55180-2a18-460e-8abb-a9f02f0893be"))
```

This is invoked by a bash shell script as follows:

```
#!/bin/sh
emacsclient -e '(bh/clock-in-bzflagt-task)'
~/git/bzflag/trunk/bzflag/src/bzflag/bzflag -directory ~/git/bzflag/trunk/b
emacsclient -e '(bh/resume-clock)'
```

The resume clock function just returns the clock to the previous clocking task

```
(defun bh/resume-clock ()
  (interactive)
  (if (marker-buffer org-clock-interrupted-task)
      (org-with-point-at org-clock-interrupted-task
        (org-clock-in))
    (org-clock-out)))
```

If no task was clocking `bh/resume-clock` just stops the clock.

7 Time reporting and tracking

7.1 Billing clients based on clocked time

At the beginning of the month I invoice my clients for work done last month. This is where I review my clocking data for correctness before billing for the clocked time.

Billing for clocked time basically boils down to the following steps:

1. Verify that the clock data is complete and correct
2. Use clock reports to summarize time spent
3. Create an invoice based on the clock data

   I currently create invoices in an external software package based on the org-mode clock data.
4. Archive complete tasks so they are out of the way.

   See Archiving for more details.

7.1.1 Verify that the clock data is complete and correct

Since I change tasks often (sometimes more than once in a minute) I use the following setting to remove clock entries with a zero duration.

```
;; Sometimes I change tasks I'm clocking quickly - this removes clocked tas
(setq org-clock-out-remove-zero-time-clocks t)
```

This setting just keeps my clocked log entries clean - only keeping clock entries that contribute to the clock report.

Before invoicing for clocked time it is important to make sure your clocked time data is correct. If you have a clocked time with an entry that is not closed (ie. it has no end time) then that is a hole in your clocked day and

it gets counted as zero (0) for time spent on the task when generating clock reports. Counting it as zero is almost certainly wrong.

To check for unclosed clock times I use the agenda-view clock check (`v c` in the agenda). This view shows clocking gaps and overlaps in the agenda.

To check the last month's clock data I use `F12 a v m b v c` which shows a full month in the agenda, moves to the previous month, and shows the clocked times only. It's important to remove any agenda restriction locks and filters when checking the logs for gaps and overlaps.

The clocked-time only display in the agenda makes it easy to quickly scan down the list to see if an entry is missing an end time. If an entry is not closed you can manually fix the clock entry based on other clock info around that time.

Use the following setup to get log mode in the agenda to only show clocked times:

```
;; Agenda log mode items to display (clock time only by default)
(setq org-agenda-log-mode-items (quote (clock)))
```

7.1.2 Using clock reports to summarize time spent

Billable time for clients are kept in separate org files.

To get a report of time spent on tasks for `XYZ.org` you simply visit the `XYZ.org` file and run an agenda clock report for the last month with `F12 < a v m b R`. This limits the agenda to this one file, shows the agenda for a full month, moves to last month, and generates a clock report.

My agenda org clock report settings show 5 levels of detail with links to the tasks.

```
;; Agenda clock report parameters
(setq org-agenda-clockreport-parameter-plist
      (quote (:link t :maxlevel 5 :fileskip0 t :compact t)))
```

I used to have a monthly clock report dynamic block in each project org file and manually updated them at the end of my billing cycle. I used this as the basis for billing my clients for time spent on their projects. I found updating the dynamic blocks fairly tedious when you have more than a couple of files for the month.

I have since moved to using agenda clock reports shortly after that feature was added. I find this much more convenient. The data isn't normally for consumption by anyone else so the format of the agenda clock report format is great for my use-case.

7.2 Task Estimates and column view

Estimating how long tasks take to complete is a difficult skill to master. Org-mode makes it easy to practice creating estimates for tasks and then clock the actual time it takes to complete.

By repeatedly estimating tasks and reviewing how your estimate relates to the actual time clocked you can tune your estimating skills.

7.2.1 Creating a task estimate with column mode

I use `properties` and `column view` to do project estimates.

I set up column view globally with the following headlines

```
; Set default column view headings: Task Effort Clock_Summary
(setq org-columns-default-format "%80ITEM(Task) %10Effort(Effort){:} %10CLO
```

This makes column view show estimated task effort and clocked times side-by-side which is great for reviewing your project estimates.

A property called `Effort` records the estimated amount of time a given task will take to complete. The estimate times I use are one of:

- 10 minutes
- 30 minutes
- 1 hour
- 2 hours
- 3 hours
- 4 hours
- 5 hours
- 6 hours
- 7 hours
- 8 hours

These are stored for easy use in `column mode` in the global property `Effort_ALL`.

```
; global Effort estimate values
(setq org-global-properties (quote (("Effort_ALL" . "0:10 0:30 1:00 2:00 3:
```

To create an estimate for a task or subtree start column mode with `C-c C-x C-c` and collapse the tree with `c`. This shows a table overlayed on top of the headlines with the task name, effort estimate, and clocked time in columns.

With the cursor in the `Effort` column for a task you can easily set the estimated effort value with the quick keys `1` through `9`.

After setting the effort values exit `column mode` with `q`.

7.2.2 Saving your estimate

For fixed price jobs where you provide your estimate to a client, then work to complete the project it is useful to save the original estimate that is provided to the client.

Save your original estimate by creating a dynamic clock report table at the top of your estimated project subtree. Entering `C-c C-x i RET` inserts a clock table report with your estimated values and any clocked time to date.

```
Original Estimate
#+BEGIN: columnview :hlines 1 :id local
| Task                        | Estimated Effort | CLOCKSUM |
|-----------------------------+------------------+----------|
| ** TODO Project to estimate |             5:40 |          |
| *** TODO Step 1             |             0:10 |          |
| *** TODO Step 2             |             0:10 |          |
| *** TODO Step 3             |             5:10 |          |
| **** TODO Step 3.1          |             2:00 |          |
| **** TODO Step 3.2          |             3:00 |          |
| **** TODO Step 3.3          |             0:10 |          |
| *** TODO Step 4             |             0:10 |          |
#+END:
```

I normally delete the `#+BEGIN:` and `#+END:` lines from the original table after providing the estimate to the client to ensure I don't accidentally update the table by hitting `C-c C-c` on the `#+BEGIN:` line.

Saving the original estimate data makes it possible to refine the project tasks into subtasks as you work on the project without losing the original estimate data.

7.2.3 Reviewing your estimate

`Column view` is great for reviewing your estimate. This shows your estimated time value and the total clock time for the project side-by-side.

Creating a dynamic clock table with `C-c C-x i RET` is a great way to save this project review if you need to make it available to other applications.

`C-c C-x C-d` also provides a quick summary of clocked time for the current org file.

8 Tags

Tasks can have any number of arbitrary tags. Tags are used for:

- filtering todo lists and agenda views
- providing context for tasks
- tagging notes
- tagging phone calls
- tagging tasks to be refiled
- tagging tasks in a WAITING state because a parent task is WAITING
- tagging cancelled tasks because a parent task is CANCELLED
- preventing export of some subtrees when publishing

I use tags mostly for filtering in the agenda. This means you can find tasks with a specific tag easily across your large number of org-mode files.

Some tags are mutually exclusive. These are defined in a group so that only one of the tags can be applied to a task at a time (disregarding tag inheritance). I use these types for tags for applying context to a task. (Work tasks have an `@office` tag, and are done at the office, Farm tasks have an `@farm` tag and are done at the farm – I can't change the oil on the tractor if I'm not at the farm... so I hide these and other tasks by filtering my agenda view to only `@office` tasks when I'm at the office.)

Tasks are grouped together in org-files and a `#+FILETAGS:` entry applies a tag to all tasks in the file. I use this to apply a tag to all tasks in the file. My norang.org file creates a NORANG file tag so I can filter tasks in the agenda in the norang.org file easily.

8.1 Tags

Here are my tag definitions with associated keys for filtering in the agenda views.

The startgroup - endgroup (`@XXX`) tags are mutually exclusive - selecting one removes a similar tag already on the task. These are the context tags - you can't be in two places at once so if a task is marked with @farm and you add @office then the @farm tag is removed automagically.

The other tags `QUOTE` .. `CANCELLED` are not mutually exclusive and multiple tags can appear on a single task. Some of those tags are created by todo state change triggers. The shortcut key is used to add or remove the tag using `C-c C-q` or to apply the task for filtering on the agenda.

I have both `FARM` and `@farm` tags. `FARM` is set by a `FILETAGS` entry and just gives me a way to filter anything farm related. The `@farm` tag signifies that the task as to be done *at the farm*. If I have to call someone about something that would have a `FARM` tag but I can do that at home on my lunch break. I don't physically have to be at the farm to make the call.

```
; Tags with fast selection keys
(setq org-tag-alist (quote ((:startgroup)
                            ("@errand" . ?e)
                            ("@office" . ?o)
                            ("@home" . ?h)
                            ("@farm" . ?f)
                            (:endgroup)
                            ("PHONE" . ?p)
                            ("QUOTE" . ?q)
                            ("WAITING" . ?w)
                            ("PERSONAL" . ?P)
                            ("WORK" . ?W)
                            ("FARM" . ?F)
                            ("ORG" . ?O)
                            ("NORANG" . ?N)
                            ("crypt" . ?E)
                            ("MARK" . ?M)
                            ("NOTE" . ?n)
                            ("BZFLAG" . ?B)
                            ("CANCELLED" . ?c)
                            ("FLAGGED" . ??))))

; Allow setting single tags without the menu
```

```
(setq org-fast-tag-selection-single-key (quote expert))

; For tag searches ignore tasks with scheduled and deadline dates
(setq org-agenda-tags-todo-honor-ignore-options t)
```

8.2 Filetags

Filetags are a convenient way to apply one or more tags to all of the headings in a file.

Filetags look like this:

```
#+FILETAGS: NORANG @office
```

I have the following `#+FILETAGS:` entries in my org-mode files:

8.2.1 Non-work related org-mode files

File	Tags
todo.org	PERSONAL
gsoc2009.org	GSOC PERSONAL
bzflag.org	BZFLAG @home PERSONAL
git.org	GIT WORK
org.org	ORG WORK
mark.org	MARK PERSONAL
farm.org	FARM PERSONAL

8.2.2 Work related org-mode files

File	Tags
norang.org	NORANG @office
ABC.org	ABC @office
XYZ.org	XYZ @office
ABC-DEF.org	ABC DEF @office
ABC-KKK.org	ABC KKK @office
YYY.org	YYY @office

8.2.3 Refile tasks

File	Tags
refile.org	REFILE

8.3 Trigger Tags

The following tags are automatically added or removed by todo state triggers described previously in ToDo state triggers

- `WAITING`
- `CANCELLED`

9 Handling Notes

Notes are little gems of knowledge that you come across during your day. They are just like tasks except there is nothing to do (except learn and memorize the gem of knowledge). Unfortunately there are way too many gems to remember and my head explodes just thinking about it.

org-mode to the rescue!

Often I'll find some cool feature or thing I want to remember while reading the org-mode and git mailing lists in Gnus. To create a note I use my note capture template `C-M-r n`, type a heading for the note and `C-c C-c` to save it. The only other thing to do is to refile it (later) to the appropriate project file.

I have an agenda view just to find notes. Notes are refiled to an appropriate project file and task. If there is no specific task it belongs to it goes to the catchall `* Notes` task. I generally have a catchall notes task in every project file. Notes are created with a `NOTE` tag already applied by the capture template so I'm free to refile the note anywhere. As long as the note is in a project file that contributes to my agenda (ie. in org-agenda-files) then I can find the note back easily with my notes agenda view by hitting the key combination `F12 N`. I'm free to limit the agenda view of notes using standard agenda tag filtering.

Short notes with a meaningful headline are a great way to remember technical details without the need to actually remember anything - other than how to find them back when you need them using `F12 N`.

Notes that are project related and not generally useful can be archived with the project and removed from the agenda when the project is removed.

So my org notes go in org.org and my git notes go in git.org both under the `* Notes` task. I'll forever be able to find those. A note about some work project detail I want to remember with the project is filed to the project task under the appropriate work org-mode file and eventually gets removed from the agenda when the project is complete and archived.

10 Handling Phone Calls

Phone calls are interruptions and I use capture mode to deal with these (like all interruptions). Most of the heavy lifting for phone calls is done by capture mode. I use a special capture template for phone calls combined with a custom function that replaces text with information from my `bbdb` addressbook database.

`C-M-r p` starts a capture task normally and I'm free to enter notes from the call in the template immediately. The cursor starts in the template normally where the name of the caller would be inserted. I can use a `bbdb` lookup function to insert the name with `f9-p` or I can just type in whatever is appropriate. If a `bbdb` entry needs to be created for the caller I can do that and replace the caller details with `f9-p` anytime that is convenient for me. I found that automatically calling the bbdb lookup function would interrupt my workflow during the call in cases where the information about the caller was not readily available. Sometimes I want to make notes first and get the caller details later during the call.

The phone call capture template starts the clock as soon as the phone rings and I'm free to lookup and replace the caller in bbdb anytime during or after the call. Capture mode starts the clock using the `:clock-in t` setting in the template.

When the phone call ends I simple do `C-c C-c` to close the capture buffer and stop the clock. If I have to close it early and look up other information during the call I just do `C-c C-c F9-SPC` to close the capture buffer (which stops the clock) and then immediately switch back to the last clocked item to continue the clock in the phone call task. When the phone call ends I clock out which normally clocks in my default task again (if any).

Here is my set up for phone calls. I would like to thank Gregory J. Grubbs for the original bbdb lookup functions which this version is based on.

Below is the partial capture template showing the phone call template followed by the phone-call related lookup functions.

```
;; Capture templates for: TODO tasks, Notes, appointments, phone calls, ana
(setq org-capture-templates
      (quote (...
              ("p" "Phone call" entry (file "~/git/org/refile.org")
               "* PHONE %? :PHONE:\n%U" :clock-in t :clock-resume t)
              ...)))

(require 'bbdb)
```

```
(require 'bbdb-com)

(global-set-key (kbd "<f9> p") 'bh/phone-call)

;;
;; Phone capture template handling with BBDB lookup
;; Adapted from code by Gregory J. Grubbs
(defun bh/phone-call ()
  "Return name and company info for caller from bbdb lookup"
  (interactive)
  (let* (name rec caller)
    (setq name (completing-read "Who is calling? "
                                (bbdb-hashtable)
                                'bbdb-completion-predicate
                                'confirm))
    (when (> (length name) 0)
      ; Something was supplied - look it up in bbdb
      (setq rec
            (or (first
                 (or (bbdb-search (bbdb-records) name nil nil)
                     (bbdb-search (bbdb-records) nil name nil)))
                name)))

    ; Build the bbdb link if we have a bbdb record, otherwise just return t
    (setq caller (cond ((and rec (vectorp rec))
                        (let ((name (bbdb-record-name rec))
                              (company (bbdb-record-company rec)))
                          (concat "[[bbdb:"
                                  name "][["
                                  name "]]"
                                  (when company
                                    (concat "  -  " company)))))
                       (rec)
                       (t "NameOfCaller")))
    (insert caller)))
```

11 GTD stuff

Most of my day is deadline/schedule driven. I work off of the agenda first and then pick items from the todo lists as outlined in What do I work on next?

11.1 Weekly Review Process

The first day of the week (usually Monday) I do my weekly review. I keep a list like this one to remind me what needs to be done.

To keep the agenda fast I set

```
(setq org-agenda-ndays 1)
```

so only today's date is shown by default. I only need the weekly view during my weekly review and this keeps my agenda generation fast.

I have a recurring task which keeps my weekly review checklist handy. This pops up as a reminder on Monday's. This week I'm doing my weekly review on Tuesday since Monday was a holiday.

```
** NEXT Weekly Review [0/8]
 \texttt{SCHEDULED:} <2009-05-18 Mon ++1w>

 LOGBOOK:...
 PROPERTIES:...

 What to review:

  - [ ] Check follow-up folder
  - [ ] Review weekly agenda F12 a w //
  - [ ] Check clocking data for past week v c b
  - [ ] Review clock report for past week R
    - Check where we spent time (too much or too little) and rectify this week

  - start work
    - daily agenda first - knock off items
    - then work on NEXT tasks
```

The first item [ ] Check follow-up folder makes me pull out the paper file I dump stuff into all week long - things I need to take care of but are in no particular hurry to deal with. Stuff I get in the mail etc. that I don't want

to deal with now. I just toss it in my `Follow-Up` folder in the filing cabinet and forget about it until the weekly review.

I go through the folder and weed out anything that needs to be dealt with. After that everything else is in `org-mode`. I tend to schedule tasks onto the agenda for the coming week so that I don't spend lots of time trying to find what needs to be worked on next.

This works for me. Your mileage may vary ;)

11.2 Project definition and finding stuck projects

I'm using a new lazy project definition to mark tasks as projects. This requires zero effort from me. Any task with a subtask using a todo keyword is a project. Period.

Projects are 'stuck' if they have no subtask with a `NEXT` or `STARTED` todo keyword task defined.

The org-mode stuck projects agenda view lists projects that have no `NEXT` task defined. Stuck projects show up on my block agenda and I tend to assign a `NEXT` task so the list remains empty. This helps to keep projects moving forward.

I disable the default org-mode stuck projects agenda view with the following setting.

```
(setq org-stuck-projects (quote ("" nil nil "")))
```

This prevents org-mode from trying to show incorrect data if I select the default stuck project view with `F12 #` from the agenda menu. My customized stuck projects view is part of my block agenda displayed with `F12 a`.

Projects can have subprojects - and these subprojects can also be stuck. Any project that is stuck shows up on the stuck projects list so I can indicate or create a `NEXT` task to move that project forward.

In the following example `Stuck Project A` is stuck because it has no subtask which is `NEXT`. `Project C` is not stuck because it has `NEXT` tasks `SubTask G` and `Task I`. `Stuck Sub Project D` is stuck because `SubTask E` is not `NEXT` and there are no other tasks available in this project.

```
* Category
** TODO Stuck Project A
*** TODO Task B
** TODO Project C
*** TODO Stuck Sub Project D
**** TODO SubTask E
*** TODO Sub Project F
```

```
**** NEXT SubTask G
**** TODO SubTask H
*** NEXT Task I
*** TODO Task J
```

All of the stuck projects and subprojects show up in the stuck projects list and that is my indication to assign or create `NEXT` tasks until the stuck projects list is empty. Occasionally some subtask is `WAITING` for something and the project is stuck until that condition is satisfied. In this case I leave it on the stuck project list and just work on something else. This stuck project 'bugs' me regularly when I see it on the block agenda and this prompts me to follow up on the thing that I'm waiting for.

I have the following helper functions defined for projects which are used by agenda views.

```
(defun bh/is-project-p ()
  "Any task with a todo keyword subtask"
  (let ((has-subtask)
        (subtree-end (save-excursion (org-end-of-subtree t)))
        (is-a-task (member (nth 2 (org-heading-components)) org-todo-keywor
    (save-excursion
      (forward-line 1)
      (while (and (not has-subtask)
                  (< (point) subtree-end)
                  (re-search-forward "^\*+ " subtree-end t))
        (when (member (org-get-todo-state) org-todo-keywords-1)
          (setq has-subtask t))))
    (and is-a-task has-subtask)))

(defun bh/is-subproject-p ()
  "Any task which is a subtask of another project"
  (let ((is-subproject)
        (is-a-task (member (nth 2 (org-heading-components)) org-todo-keywor
    (save-excursion
      (while (and (not is-subproject) (org-up-heading-safe))
        (when (member (nth 2 (org-heading-components)) org-todo-keywords-1)
          (setq is-subproject t))))
    (and is-a-task is-subproject)))

(defun bh/skip-non-stuck-projects ()
  "Skip trees that are not stuck projects"
```

```
  (let* ((next-headline (save-excursion (or (outline-next-heading) (point-m
         (subtree-end (save-excursion (org-end-of-subtree t)))
         (has-next (save-excursion
                     (forward-line 1)
                     (and (< (point) subtree-end)
                          (re-search-forward "^\\*+ \\(NEXT\\|STARTED\\) "
    (if (and (bh/is-project-p) (not has-next))
        nil ; a stuck project, has subtasks but no next task
      next-headline)))

(defun bh/skip-non-projects ()
  "Skip trees that are not projects"
  (let ((subtree-end (save-excursion (org-end-of-subtree t))))
    (if (bh/is-project-p)
        nil
      subtree-end)))

(defun bh/skip-project-trees-and-habits ()
  "Skip trees that are projects"
  (let ((subtree-end (save-excursion (org-end-of-subtree t))))
    (cond
     ((bh/is-project-p)
      subtree-end)
     ((org-is-habit-p)
      subtree-end)
     (t
      nil))))

(defun bh/skip-projects-and-habits ()
  "Skip trees that are projects and tasks that are habits"
  (let ((next-headline (save-excursion (or (outline-next-heading) (point-ma
    (cond
     ((bh/is-project-p)
      next-headline)
     ((org-is-habit-p)
      next-headline)
     (t
      nil))))

(defun bh/skip-non-subprojects ()
```

```
"Skip trees that are not projects"
(let ((next-headline (save-excursion (outline-next-heading))))
  (if (bh/is-subproject-p)
      nil
    next-headline)))
```

12 Archiving

12.1 Archiving Subtrees

My archiving procedure has changed. I used to move entire subtrees to a separate archive file for the project. Task subtrees in `FILE.org` get archived to `FILE.org_archive` using the `a y` command in the agenda.

I still archive to the same archive file as before but now I archive any done state todo task that is old enough to archive. Tasks to archive are listed automatically at the end of my block agenda and these are guaranteed to be old enough that I've already billed any time associated with these tasks. This cleans up my project trees and removes the old tasks that are no longer interesting. The archived tasks get extra property data created during the archive procedure so that it is possible to reconstruct exactly where the archived entry came from in the rare case where you want to unarchive something.

My archive files are huge but so far I haven't found a need to split them by year (or decade) :)

Archivable tasks show up in the last section of my block agenda when a new month starts. Any tasks that are done but have no timestamps this month or last month (ie. they are over 30 days old) are available to archive. Timestamps include closed dates, notes, clock data, etc - any active or inactive timestamp in the task.

Archiving is trivial. Just mark all of the entries in the block agenda using the `m` key and then archive them all to the appropriate place with `B $`. This normally takes less than 5 minutes once a month.

12.2 Archive Setup

I not longer use an `ARCHIVE` property in my subtrees. Tasks can just archive normally to the `Archived Tasks` heading in the archive file.

The following setting ensures that task states are untouched when they are archived. This makes it possible to archive tasks that are not marked

DONE. By default tasks are archived under the heading `* Archived Tasks` in the archive file.

```
(setq org-archive-mark-done nil)
(setq org-archive-location "%s_archive::* Archived Tasks")

(defun bh/skip-non-archivable-tasks ()
  "Skip trees that are not available for archiving"
  (let ((next-headline (save-excursion (or (outline-next-heading) (point-ma
    ;; Consider only tasks with done todo headings as archivable candidates
    (if (member (org-get-todo-state) org-done-keywords)
        (let* ((subtree-end (save-excursion (org-end-of-subtree t)))
               (daynr (string-to-int (format-time-string "%d" (current-time
               (a-month-ago (* 60 60 24 (+ daynr 1)))
               (last-month (format-time-string "%Y-%m-" (time-subtract (cur
               (this-month (format-time-string "%Y-%m-" (current-time)))
               (subtree-is-current (save-excursion
                                     (forward-line 1)
                                     (and (< (point) subtree-end)
                                          (re-search-forward (concat last-m
          (if subtree-is-current
              next-headline ; Has a date in this month or last month, skip
            nil))  ; available to archive
      (or next-headline (point-max)))))
```

12.3 Archive Tag - Hiding Information

The only time I set the ARCHIVE tag on a task is to prevent it from opening by default because it has tons of information I don't really need to look at on a regular basis. I can open the task with C-TAB if I need to see the gory details (like a huge table of data related to the task) but normally I don't need that information displayed.

12.4 When to Archive

Archiving monthly works well for me. I keep completed tasks around for at least 30 days before archiving them. This keeps current clocking information for the last 30 days out of the archives. This keeps my files that contribute to the agenda fairly current (this month, and last month, and anything that is unfinished). I only rarely visit tasks in the archive when I need to pull up ancient history for something.

Archiving keeps my main working files clutter-free. If I ever need the detail for the archived tasks they are available in the appropriate archive file.

13 Publishing and Exporting

I don't do a lot of publishing for other people but I do keep a set of private client system documentation online. Most of this documentation is a collection of notes exported to HTML.

Everything at http://doc.norang.ca/ is generated by publishing org-files. This includes the index pages on this site.

Org-mode can export to a variety of publishing formats including (but not limited to)

- ASCII (plain text - but not the original org-mode file)
- HTML
- LaTeX
- Docbook which enables getting to lots of other formats like ODF, XML, etc
- PDF via LaTeX or Docbook
- iCal

I haven't begun the scratch the surface of what org-mode is capable of doing. My main use case for org-mode publishing is just to create HTML documents for viewing online conveniently. Someday I'll get time to try out the other formats when I need them for something.

13.1 Org-babel Setup

Org-babel makes it easy to generate decent graphics using external packages like ditaa, graphviz, PlantUML, and others.

The setup is really easy. `ditaa` is provided with the org-mode source. You'll have to install the `graphviz` and `PlantUML` packages on your system.

```
(setq org-ditaa-jar-path "~/java/ditaa0_6b.jar")
(setq org-plantuml-jar-path "~/java/plantuml.jar")
```

```
(add-hook 'org-babel-after-execute-hook 'org-display-inline-images 'append)

(org-babel-do-load-languages
 (quote org-babel-load-languages)
 (quote ((emacs-lisp . t)
         (dot . t)
         (ditaa . t)
         (R . t)
         (python . t)
         (ruby . t)
         (gnuplot . t)
         (clojure . t)
         (sh . t)
         (ledger . t)
         (org . t)
         (plantuml . t)
         (latex . t))))

; Do not prompt to confirm evaluation
; This may be dangerous - make sure you understand the consequences
; of setting this -- see the docstring for details
(setq org-confirm-babel-evaluate nil)
```

Now you just create a `begin-src` block for the appropriate tool, edit the text, and build the pictures with `C-c C-c`. After evaluating the block results are displayed. You can toggle display of inline images with `C-c C-x C-v`

I disable startup with inline images because when I access my org-files from an SSH session without X this breaks (say from my Android phone) it fails when trying to display the images on a non-X session. It's much more important for me to be able to access my org files from my Android phone remotely than it is to see images on startup.

```
;; Don't enable this because it breaks access to emacs from my Android phon
(setq org-startup-with-inline-images nil)
```

13.2 Playing with ditaa

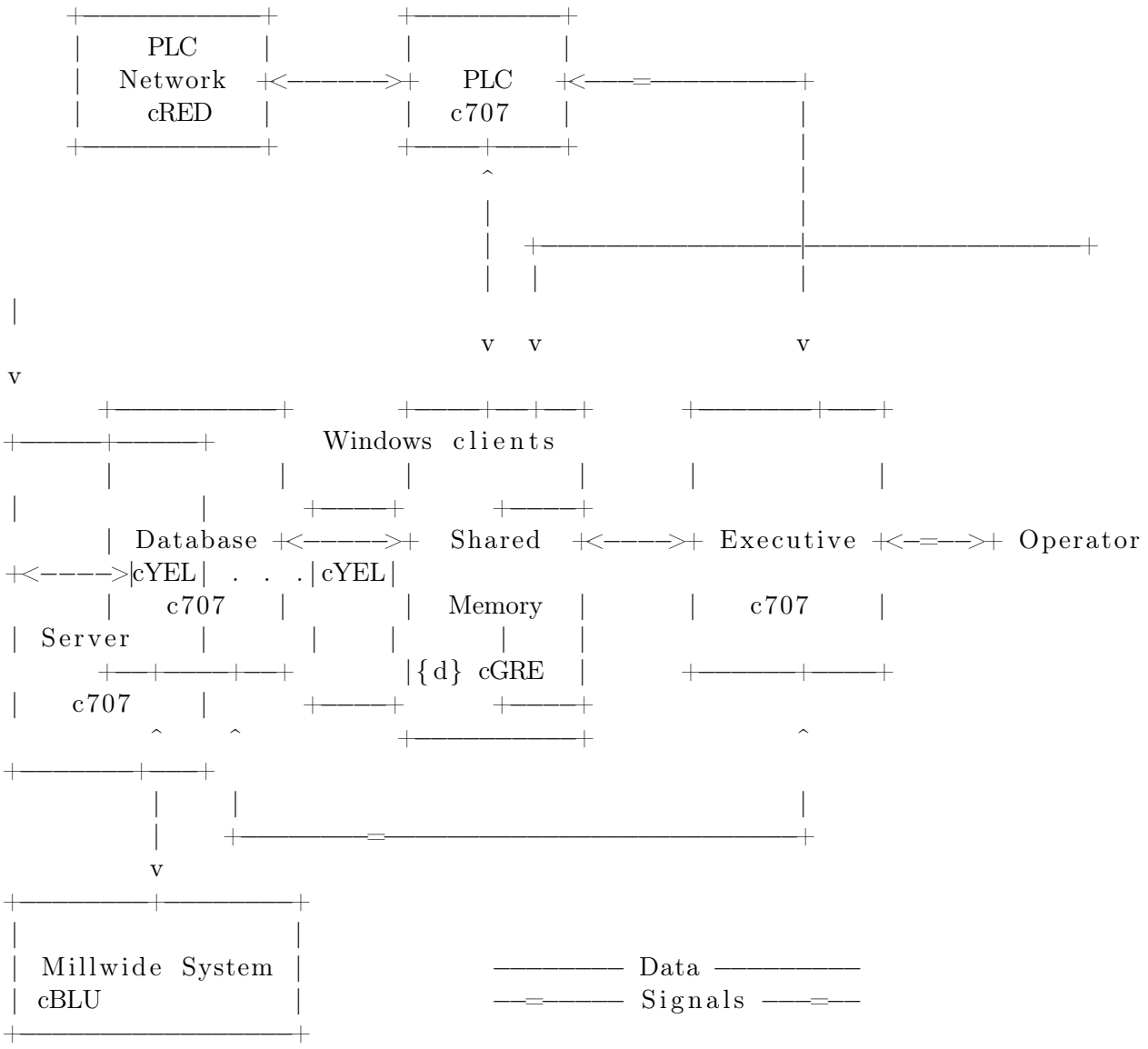
ditaa is a great tool for quickly generating graphics to convey ideas and `ditaa` is distributed with org-mode! All of the graphics in this document are automatically generated by org-mode using plain text source.

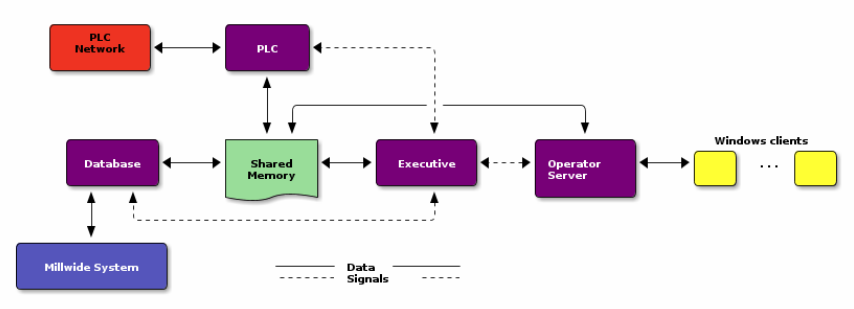
Artist mode makes it easy to create boxes and lines for ditaa graphics.

The source for a ditaa graphic looks like this in org-mode:

```
#+begin_src ditaa :file some_filename.png :cmdline -r -s 0.8
  <context of ditaa source goes here>
#+end_src
```

Here's an example without the `#+begin_src` and `#+end_src` lines.





13.3 Playing with graphviz

Graphviz is another great tool for creating graphics in your documents.

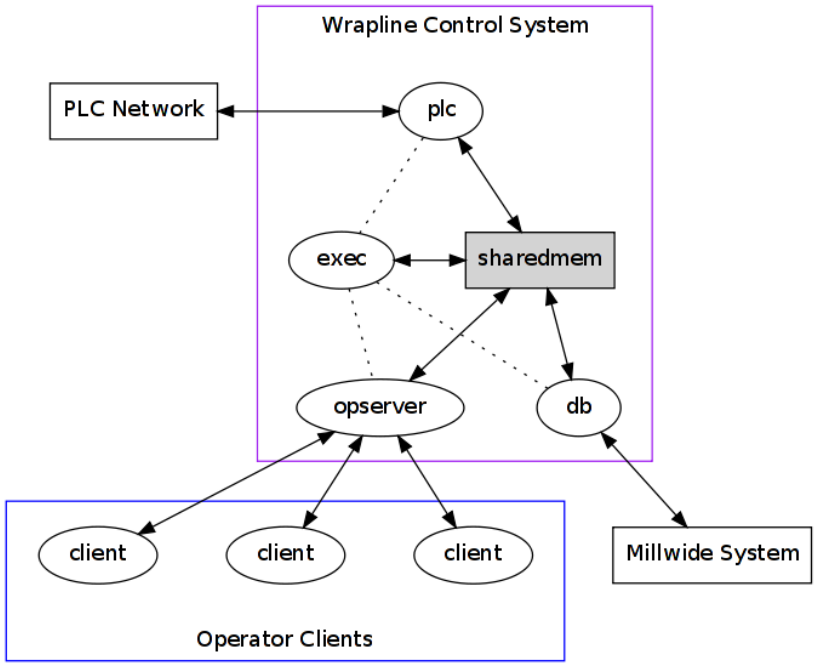
The source for a graphviz graphic looks like this in org-mode:

```
#+begin_src dot :file some_filename.png :cmdline -Kdot -Tpng
  <context of graphviz source goes here>
#+end_src
```

```
digraph G {
  size="8,6"
  ratio=expand
  edge [dir=both]
  plcnet [shape=box, label="PLC Network"]
  subgraph cluster_wrapline {
    label="Wrapline Control System"
    color=purple
    subgraph {
    rank=same
    exec
    sharedmem [style=filled, fillcolor=lightgrey, shape=box]
    }
    edge[style=dotted, dir=none]
    exec -> opserver
    exec -> db
    plc -> exec
    edge [style=line, dir=both]
    exec -> sharedmem
    sharedmem -> db
    plc -> sharedmem
```

```
    sharedmem -> opserver
  }
  plcnet -> plc [constraint=false]
  millwide [shape=box, label="Millwide System"]
  db -> millwide

  subgraph cluster_opclients {
    color=blue
    label="Operator Clients"
    rankdir=LR
    labelloc=b
    node[label=client]
    opserver -> client1
    opserver -> client2
    opserver -> client3
  }
}
```



The `-Kdot` is optional (defaults to `dot`) but you can substitute other graphviz types instead here (ie. `twopi`, `neato`, `circo`, etc).

13.4 Playing with PlantUML

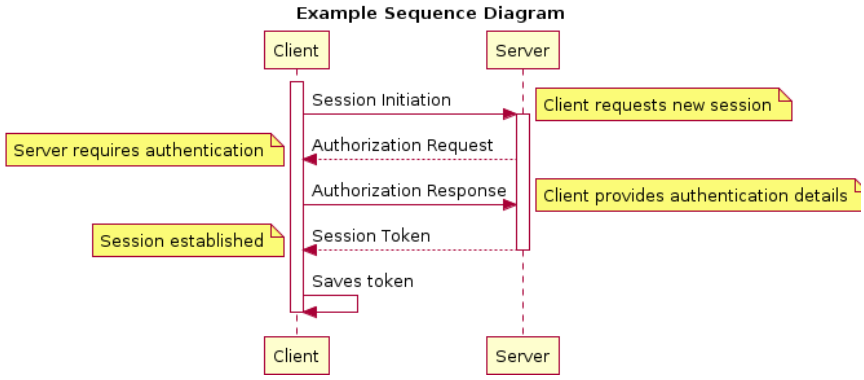
I have just started using PlantUML which is built on top of Graphviz. I'm still experimenting with this but so far I like it a lot. The todo state change diagrams in this document are created with PlantUML.

The source for a PlantUML graphic looks like this in org-mode:

```
#+begin_src plantuml :file somefile.png
  <context of PlantUML source goes here>
#+end_src
```

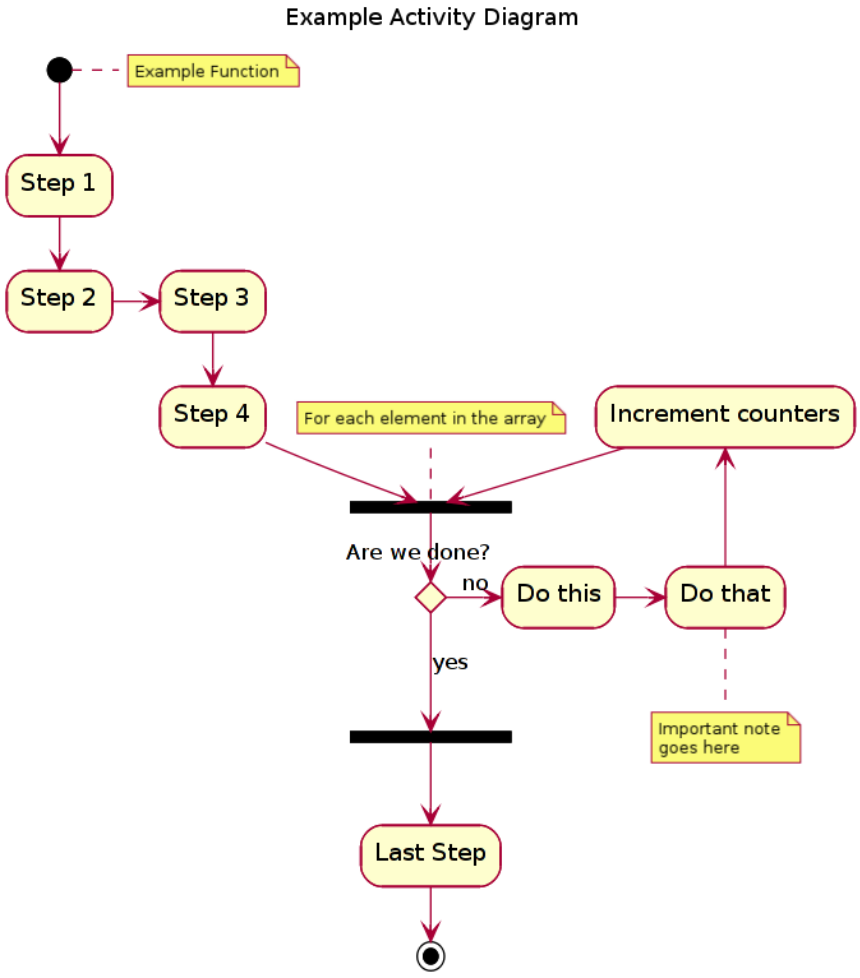
13.4.1 Sequence Diagram

```
title Example Sequence Diagram
activate Client
Client -> Server: Session Initiation
note right: Client requests new session
activate Server
Client <-- Server: Authorization Request
note left: Server requires authentication
Client -> Server: Authorization Response
note right: Client provides authentication details
Server --> Client: Session Token
note left: Session established
deactivate Server
Client -> Client: Saves token
deactivate Client
```



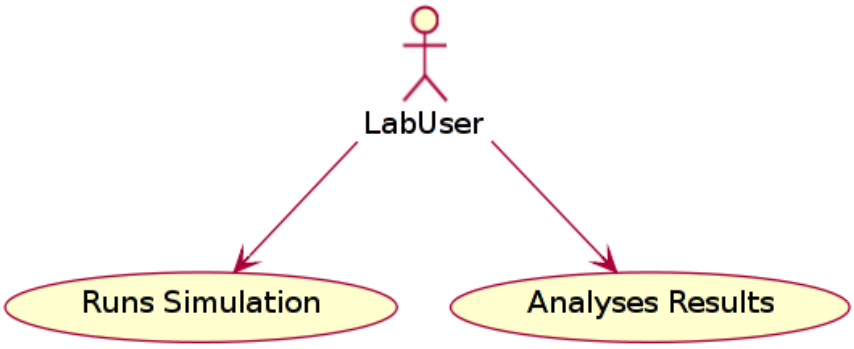
13.4.2 Activity Diagram

```
title Example Activity Diagram
note right: Example Function
(*)--> "Step 1"
--> "Step 2"
-> "Step 3"
--> "Step 4"
--> ==== STARTLOOP ====
note top: For each element in the array
--> if "Are we done?" then
  -> [no] "Do this"
  -> "Do that"
  note bottom: Important note\ngoes here
  -up-> "Increment counters"
  --> ==== STARTLOOP ====
else
  --> [yes] ==== ENDLOOP ====
endif
--> "Last Step"
--> (*)
```



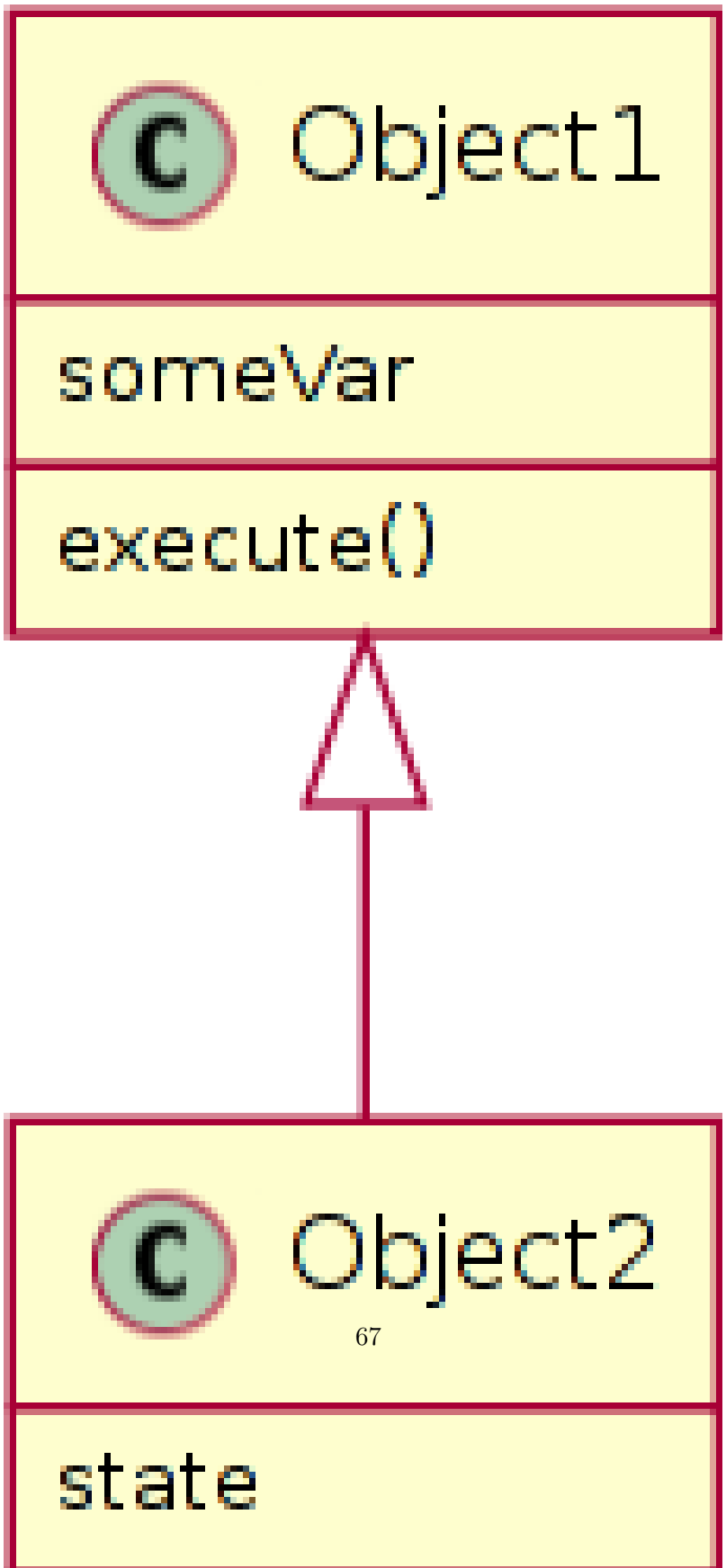
13.4.3 Usecase Diagram

LabUser –-> (Runs Simulation)
LabUser –-> (Analyses Results)

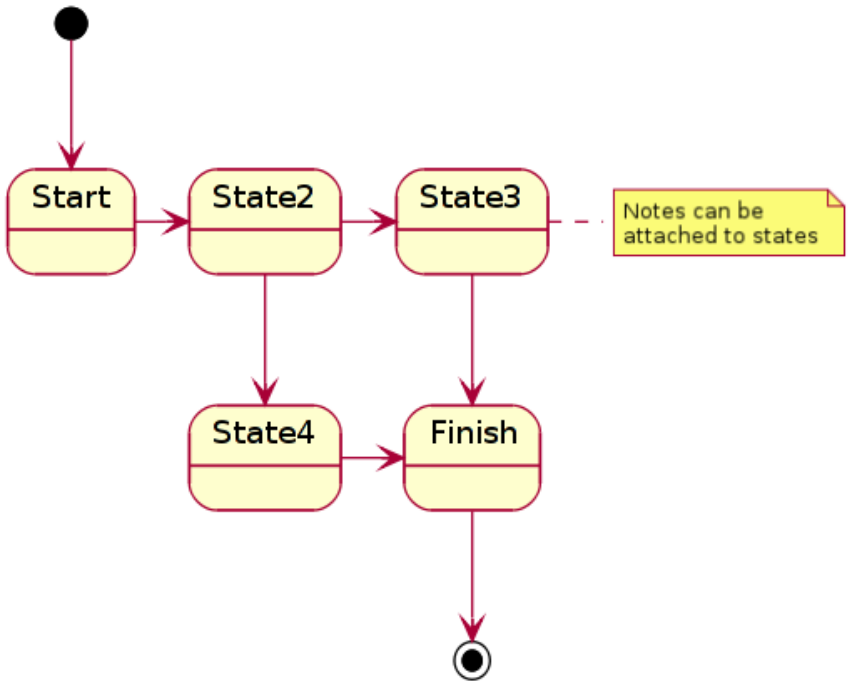


13.4.4 Object Diagram

```
Object1 <|-- Object2
Object1: someVar
Object1: execute()
Object2: getState()
Object2: setState()
Object2: state
```


Object1
someVar
execute()
Object2
state

```
[*] --> Start
Start -> State2
State2 -> State3
note right of State3: Notes can be\nattached to states
State2 --> State4
State4 -> Finish
State3 --> Finish
Finish --> [*]
```



13.4.6 Publishing Single Files

Org-mode exports the current file to one of the standard formats by invoking an export function. The standard key binding for this is `C-c C-e` followed by the key for the type of export you want.

This works great for single files or parts of files – if you narrow the buffer to only part of the org-mode file then you only get the narrowed detail in the export.

13.5 Publishing Projects

I mainly use publishing for publishing multiple files or projects. I don't want to remember where the created export file needs to move to and org-mode

projects are a great solution to this.

The http://doc.norang.ca website (and a bunch of other files that are not publicly available) are all created by editing org-mode files and publishing the project the file is contained in. This is great for people like me who want to figure out the details once and forget about it. I love stuff that Just Works(tm).

I have 5 main projects I use org-mode publishing for currently:

- norang (website)
- doc.norang.ca (website, published documents)
- doc.norang.ca/private (website, non-published documents)
- www.norang.ca/tmp (temporary publishing site for testing org-mode stuff)
- org files (which are selectively included by other websites)

Here's my publishing setup:

```
; experimenting with docbook exports - not finished
(setq org-export-docbook-xsl-fo-proc-command "fop %s %s")
(setq org-export-docbook-xslt-proc-command "xsltproc --output %s /usr/share
;
; Inline images in HTML instead of producting links to the image
(setq org-export-html-inline-images t)
; Do not use sub or superscripts - I currently don't need this functionalit
(setq org-export-with-sub-superscripts nil)
; Use org.css from the norang website for export document stylesheets
(setq org-export-html-style-extra "<link rel=\"stylesheet\" href=\"http://d
(setq org-export-html-style-include-default nil)
; Do not generate internal css formatting for HTML exports
(setq org-export-htmlize-output-type (quote css))
; Export with LaTeX fragments
(setq org-export-with-LaTeX-fragments t)

; List of projects
; norang       - http://www.norang.ca/
; doc          - http://doc.norang.ca/
; org-mode-doc - http://doc.norang.ca/org-mode.html and associated files
; org          - miscellaneous todo lists for publishing
```

```
(setq org-publish-project-alist
      ;
      ; http://www.norang.ca/  (norang website)
      ; norang-org are the org-files that generate the content
      ; norang-extra are images and css files that need to be included
      ; norang is the top-level project that gets published
      (quote (("norang-org"
               :base-directory "~/git/www.norang.ca"
               :publishing-directory "/ssh:www-data@www:~/www.norang.ca/htd
               :recursive t
               :table-of-contents nil
               :base-extension "org"
               :publishing-function org-publish-org-to-html
               :style-include-default nil
               :section-numbers nil
               :table-of-contents nil
               :style "<link␣rel=\"stylesheet\"␣href=\"norang.css\"␣type=\"
               :author-info nil
               :creator-info nil)
              ("norang-extra"
               :base-directory "~/git/www.norang.ca/"
               :publishing-directory "/ssh:www-data@www:~/www.norang.ca/htd
               :base-extension "css\\|pdf\\|png\\|jpg\\|gif"
               :publishing-function org-publish-attachment
               :recursive t
               :author nil)
              ("norang"
               :components ("norang-org" "norang-extra"))
              ;
              ; http://doc.norang.ca/  (norang website)
              ; doc-org are the org-files that generate the content
              ; doc-extra are images and css files that need to be included
              ; doc is the top-level project that gets published
              ("doc-org"
               :base-directory "~/git/doc.norang.ca/"
               :publishing-directory "/ssh:www-data@www:~/doc.norang.ca/htd
               :recursive nil
               :section-numbers nil
               :table-of-contents nil
               :base-extension "org"
```

```
 :publishing-function (org-publish-org-to-html org-publish-or
 :style-include-default nil
 :style "<link␣rel=\"stylesheet\"␣href=\"/org.css\"␣type=\"te
 :author-info nil
 :creator-info nil)
("doc-extra"
 :base-directory "~/git/doc.norang.ca/"
 :publishing-directory "/ssh:www-data@www:~/doc.norang.ca/htd
 :base-extension "css\\|pdf\\|png\\|jpg\\|gif"
 :publishing-function org-publish-attachment
 :recursive nil
 :author nil)
("doc"
 :components ("doc-org" "doc-extra"))
("doc-private-org"
 :base-directory "~/git/doc.norang.ca/private"
 :publishing-directory "/ssh:www-data@www:~/doc.norang.ca/htd
 :recursive nil
 :section-numbers nil
 :table-of-contents nil
 :base-extension "org"
 :publishing-function (org-publish-org-to-html org-publish-or
 :style-include-default nil
 :style "<link␣rel=\"stylesheet\"␣href=\"/org.css\"␣type=\"te
 :auto-sitemap t
 :sitemap-filename "index.html"
 :sitemap-title "Norang␣Private␣Documents"
 :sitemap-style "tree"
 :author-info nil
 :creator-info nil)
("doc-private-extra"
 :base-directory "~/git/doc.norang.ca/private"
 :publishing-directory "/ssh:www-data@www:~/doc.norang.ca/htd
 :base-extension "css\\|pdf\\|png\\|jpg\\|gif"
 :publishing-function org-publish-attachment
 :recursive nil
 :author nil)
("doc-private"
 :components ("doc-private-org" "doc-private-extra"))
;
```

```
; Miscellaneous pages for other websites
; org are the org-files that generate the content
("org-org"
 :base-directory "~/git/org/"
 :publishing-directory "/ssh:www-data@www:~/org"
 :recursive t
 :section-numbers nil
 :table-of-contents nil
 :base-extension "org"
 :publishing-function org-publish-org-to-html
 :style-include-default nil
 :style "<link␣rel=\"stylesheet\"␣href=\"/org.css\"␣type=\"te
 :author-info nil
 :creator-info nil)
;
; http://doc.norang.ca/  (norang website)
; org-mode-doc-org this document
; org-mode-doc-extra are images and css files that need to be
; org-mode-doc is the top-level project that gets published
; This uses the same target directory as the 'doc' project
("org-mode-doc-org"
 :base-directory "~/git/org-mode-doc/"
 :publishing-directory "/ssh:www-data@www:~/doc.norang.ca/htd
 :recursive t
 :section-numbers nil
 :table-of-contents nil
 :base-extension "org"
 :publishing-function (org-publish-org-to-html org-publish-or
 :plain-source t
 :htmlized-source t
 :style-include-default nil
 :style "<link␣rel=\"stylesheet\"␣href=\"/org.css\"␣type=\"te
 :author-info nil
 :creator-info nil)
("org-mode-doc-extra"
 :base-directory "~/git/org-mode-doc/"
 :publishing-directory "/ssh:www-data@www:~/doc.norang.ca/htd
 :base-extension "css\\|pdf\\|png\\|jpg\\|gif"
 :publishing-function org-publish-attachment
 :recursive t
```

```
                :author nil)
               ("org-mode-doc"
                :components ("org-mode-doc-org" "org-mode-doc-extra"))
               ;
               ; http://doc.norang.ca/  (norang website)
               ; org-mode-doc-org this document
               ; org-mode-doc-extra are images and css files that need to be
               ; org-mode-doc is the top-level project that gets published
               ; This uses the same target directory as the 'doc' project
               ("tmp-org"
                :base-directory "/tmp/publish/"
                :publishing-directory "/ssh:www-data@www:~/www.norang.ca/htd
                :recursive t
                :section-numbers nil
                :table-of-contents nil
                :base-extension "org"
                :publishing-function (org-publish-org-to-html org-publish-or
                :plain-source t
                :htmlized-source t
                :style-include-default t
                :auto-sitemap t
                :sitemap-filename "index.html"
                :sitemap-title "Test Publishing Area"
                :sitemap-style "tree"
                :author-info nil
                :creator-info nil)
               ("tmp-extra"
                :base-directory "/tmp/publish/"
                :publishing-directory "/ssh:www-data@www:~/www.norang.ca/htd
                :base-extension "png"
                :publishing-function org-publish-attachment
                :recursive t
                :author nil)
               ("tmp"
                :components ("tmp-org" "tmp-extra")))))

; I'm lazy and don't want to remember the name of the project to publish wh
; a file that is part of a project.  So this function saves the file, and p
; the project that includes this file
;
```

```
; It's bound to C-S-F12 so I just edit and hit C-S-F12 when I'm done and m
(defun bh/save-then-publish ()
  (interactive)
  (save-buffer)
  (org-save-all-org-buffers)
  (org-publish-current-project))

(global-set-key (kbd "C-s-<f12>") 'bh/save-then-publish)
```

The main projects are `norang`, `doc`, `doc-private`, `org-mode-doc`, and `tmp`. These projects publish directly to the webserver directory on a remote web server that serves the site. Publishing one of these projects exports all modified pages, generates images, and copies the resulting files to the webserver so that they are immediately available for viewing.

The http://doc.norang.ca/ site contains subdirectories with client and private documentation that are restricted by using Apache Basic authentication. I don't create links to these sites from the publicly viewable pages. http://doc.norang.ca/someclient/ would show the index for any org files under `~/git/doc.norang.ca/someclient/` if that is set up as a viewable website. I use most of the information myself but give access to clients if they are interested in the information/notes that I keep about their systems.

This works great for me - I know where my notes are and I can access them from anywhere on the internet. I'm also free to share notes with other people by simply giving them the link to the appropriate site.

All I need to remember to do is edit the appropriate org file and publish it with C-S-F12 – not exactly hard :)

Recently I added a temporary publishing site for testing exports and validation. This is the `tmp` site which takes files from `/tmp/publish` and exports those files to a website publishing directory. This makes it easy to try new throw-away things on a live server.

13.6 Miscellaneous Export Settings

This is a collection of export and publishing related settings that I use.

13.6.1 Fontify Latex listings for source blocks

For export to latex I use the following setting to get fontified listings from source blocks:

```
(setq org-export-latex-listings t)
```

13.6.2 Export HTML without XML header

I use the following setting to remove the xml header line for HTML exports. This xml line was confusing Open Office when opening the HTML to convert to ODT.

```
(setq org-export-html-xml-declaration (quote (("html" . "")
                                              ("was-html" . "<?xml version=
                                              ("php" . "<?php echo \"<?xml
```

13.6.3 Allow binding variables on export without confirmation

The following setting allows #+BIND: variables to be set on export without confirmation. In rare situations where I want to override some org-mode variable for export this allows exporting the document without a prompt.

```
(setq org-export-allow-BIND t)
```

14 Reminders

I use appt for reminders. It's simple and unobtrusive – putting pending appointments in the status bar and beeping as 12, 9, 6, 3, and 0 minutes before the appointment is due.

Everytime the agenda is displayed (and that's lots for me) the appointment list is erased and rebuilt from the current agenda details for today. This means everytime I reschedule something, add or remove tasks that are time related the appointment list is automatically updated the next time I look at the agenda.

14.1 Reminder Setup

```
; Erase all reminders and rebuilt reminders for today from the agenda
(defun bh/org-agenda-to-appt ()
  (interactive)
  (setq appt-time-msg-list nil)
  (org-agenda-to-appt))

; Rebuild the reminders everytime the agenda is displayed
(add-hook 'org-finalize-agenda-hook 'bh/org-agenda-to-appt 'append)
```

```
; This is at the end of my .emacs - so appointments are set up when Emacs s
(bh/org-agenda-to-appt)

; Activate appointments so we get notifications
(appt-activate t)

; If we leave Emacs running overnight - reset the appointments one minute a
(run-at-time "24:01" nil 'bh/org-agenda-to-appt)
```

15 Productivity Tools

This section is a miscellaneous collection of Emacs customizations that I use with org-mode so that it Works-For-Me(tm).

15.1 Yasnippet

Yasnippet is cool but I don't use this anymore. I've replaced yasnippet with a combination of `abbrev-mode` and `skeletons` which are available by default in Emacs.

The following description applies to yasnippet version 0.5.10. The setup requirements may have changed with newer versions.

You type the snippet name and `TAB` and yasnippet expands the name with the contents of the snippet text - substituting snippet variables as appropriate.

Yasnippet comes with lots of snippets for programming languages. I used a few babel related snippets with `org-mode`.

I downloaded and installed the unbundled version of yasnippet so that I can edit the predefined snippets. I unpacked the yasnippet software in my `~/.emacs.d/plugins` directory, renamed `yasnippet0.5.10` to `yasnippet` and added the following setup in my `.emacs`:

```
(add-to-list 'load-path (expand-file-name "~/.emacs.d/plugins"))

(require 'yasnippet)
(yas/initialize)
(yas/load-directory "~/.emacs.d/plugins/yasnippet/snippets")

;; Make TAB the yas trigger key in the org-mode-hook and enable flyspell m
(add-hook 'org-mode-hook
          (lambda ()
            ;; yasnippet
```

```
(make-variable-buffer-local 'yas/trigger-key)
(org-set-local 'yas/trigger-key [tab])
(define-key yas/keymap [tab] 'yas/next-field-group)
;; flyspell mode for spell checking everywhere
(flyspell-mode 1)
;; auto-fill mode on
(auto-fill-mode 1)))
```

I used snippets for the following:

- `begin` for generic `#+begin_` blocks
- `dot` for graphviz
- `uml` for PlantUML graphics
- `sh` for bash shell scripts
- `elisp` for emacs lisp code
- initials of a person converts to their full name I used this while taking meeting notes

Here is the definition for the `begin` snippet:
org-mode Yasnippet: ~/.emacs.d/plugins/yasnippet/snippets/text-mode/org-mode/begin

```
#name : #+begin_...#+end_
# --
#+begin_$1 $2
$0
#+end_$1
```

I used this to create `#+begin_*` blocks like

- `#+begin_example`
- `#+begin_src`
- etc.

Simply type `begin` and then `TAB` it replaces the `begin` text with the snippet contents. Then type `src TAB emacs-lisp TAB` and your snippet block is done. I've shortened this specific sequence to just `elisp TAB` since I use it fairly often. There is also the build-in org-mode `<s` in column 1 to

expand to `#+begin_src ... #+end_src` and `<e` for `#+begin_example ... #+end_example`.

Hit `C-c SingeQuote(')` and insert whatever emacs-lisp code you need. While in this block you're in a mode that knows how to format and colourize emacs lisp code as you enter it which is really nice. `C-c SingleQuote(')` exits back to org-mode. This recognizes any emacs editing mode so all you have to do is enter the appropriate mode name for the block.

```
  dot
```

```
#dot : #+begin_src dot ... #+end_src
# --
#+begin_src dot :file $1 :cmdline -Kdot -Tpng
$0
#+end_src
```

```
  uml
```

```
#uml : #+begin_src plantuml ... #+end_src
# --
#+begin_src plantuml :file $1
$0
#+end_src
```

```
  sh
```

```
#sh: #+begin_src sh ... #+end_src
# --
#+begin_src sh :results output
$0
#+end_src
```

```
  elisp
```

```
#elisp : #+begin_src emacs-lisp ...#+end_src emacs-lisp
# --
#+begin_src emacs-lisp
$0
#+end_src
```

This is a great time saver.

15.2 Abbrev-mode and Skeletons

Description coming soon. My basic setup came from the Worg FAQ about shortcuts.

15.3 Limit your view to what you are working on

There is more than one way to do this. Use what works for you.

15.3.1 Narrowing to a subtree with `bh/org-todo`

`f5` and `S-f5` are bound the functions for narrowing and widening the emacs buffer as follows:

```
(global-set-key (kbd "<f5>") 'bh/org-todo)

(defun bh/org-todo ()
  (interactive)
  (widen)
  (org-narrow-to-subtree)
  (org-show-todo-tree nil))

(global-set-key (kbd "<S-f5>") 'bh/widen)

(defun bh/widen ()
  (interactive)
  (widen)
  (org-reveal))
```

This makes it easy to hide all of the other details in your org-file temporarily by limiting your view to this task subtree. Tasks are folded and hilighted so that only tasks which are incomplete are shown.

I hit `f5` a lot. This basically does a `org-narrow-to-subtree` and `C-c C-v` combination leaving the buffer in a narrowed state. I use `S-f5` to widen back to the normal view.

I also have the following setting to force showing the next headline.

```
(setq org-show-entry-below (quote ((default))))
```

This prevents too many headlines from being folded together when I'm working with collapsed trees.

15.3.2 Limiting the agenda to a subtree

`C-c C-x <` turns on the agenda restriction lock for the current subtree. This keeps your agenda focused on only this subtree. Alarms and notifications are still active outside the agenda restriction. `C-c C-x >` turns off the agenda restriction lock returning your agenda view back to normal.

15.3.3 Limiting the agenda to a file

You can limit the agenda view to a single file in multiple ways.

You can use the agenda restriction lock `C-c C-x <` on the any line before the first heading to set the agenda restriction lock to this file only. This is equivalent using a prefix argumment (`C-u C-c C-x <`) anywhere in the file. This lock stays in effect until you remove it with `C-c C-x >`.

Another way is to invoke the agenda with `F12 < a` while visiting an org-mode file. This limits the agenda view to just this file. I occasionally use this to view a file not in my `org-agenda-files` in the agenda.

15.4 Tuning the Agenda Views

Various customizations affect how the agenda views show task details. This section shows each of the customizations I use in my workflow.

15.4.1 Highlight the current agenda line

The following code in my `.emacs` file keeps the current agenda line highlighted. This makes it obvious what task will be affected by commands issued in the agenda. No more acting on the wrong task by mistake!

The clock modeline time is also shown with a reverse background.

```
;; Always hilight the current agenda line
(add-hook 'org-agenda-mode-hook '(lambda () (hl-line-mode 1)) 'append)

;; The following custom-set-faces create the highlights
(custom-set-faces
  ;; custom-set-faces was added by Custom.
  ;; If you edit it by hand, you could mess it up, so be careful.
  ;; Your init file should contain only one such instance.
  ;; If there is more than one, they won't work right.
 '(highlight ((t (:background "cyan"))))
 '(hl-line ((t (:inherit highlight :background "darkseagreen2"))))
 '(org-mode-line-clock ((t (:background "grey75" :foreground "red" :box (:l
```

15.4.2 Keep tasks with timestamps visible on the global todo lists

Tasks with dates (`SCHEDULED:`, `DEADLINE:`, or active dates) show up in the agenda when appropriate. The block agenda view (`F12 a`) tries to keep tasks showing up only in one location (either in the calendar or other

todo lists in later sections of the block agenda.) I now rarely use the global todo list search in org-mode (`F12 t`, `F12 m`) and when I do I'm trying to find a specific task quickly. These lists now include everything so I can just search for the item I want and move on.

The block agenda prevents display of tasks with deadlines or scheduled dates in the future so you can safely ignore these until the appropriate time.

```
;; Keep tasks with dates on the global todo lists
(setq org-agenda-todo-ignore-with-date nil)

;; Keep tasks with deadlines on the global todo lists
(setq org-agenda-todo-ignore-deadlines nil)

;; Keep tasks with scheduled dates on the global todo lists
(setq org-agenda-todo-ignore-scheduled nil)

;; Keep tasks with timestamps on the global todo lists
(setq org-agenda-todo-ignore-timestamp nil)

;; Remove completed deadline tasks from the agenda view
(setq org-agenda-skip-deadline-if-done t)

;; Remove completed scheduled tasks from the agenda view
(setq org-agenda-skip-scheduled-if-done t)

;; Remove completed items from search results
(setq org-agenda-skip-timestamp-if-done t)
```

15.4.3 Use the Diary for Holidays and Appointments

I don't use the emacs Diary for anything but I like seeing the holidays on my agenda. This helps with planning for those days when you're not supposed to be working.

```
(setq org-agenda-include-diary nil)
(setq org-agenda-diary-file "~/git/org/diary.org")
```

The diary file keeps `date-tree` entries created by the capture mode 'appointment' template. I use this also for miscellaneous tasks I want to clock during interruptions.

I don't use a `~/diary` file anymore. That is just there as a zero-length file to keep Emacs happy. I use org-mode's diary functions instead.

Inserting entries with `i` in the emacs agenda creates date entries in the `~/git/org/diary.org` file.

I include holidays from the calendar in my `todo.org` file as follows:

```
#+FILETAGS: PERSONAL
* Appointments
  :PROPERTIES:
  :CATEGORY: Appt
  :ARCHIVE:  %s_archive::* Appointments
  :END:
** Holidays
   :PROPERTIES:
   :Category: Holiday
   :END:
%%(org-calendar-holiday)
** Some other Appointment
   ...
```

I use the following setting so any time strings in the heading are shown in the agenda.

```
(setq org-agenda-insert-diary-extract-time t)
```

15.4.4 Searches include archive files

I keep a single archive file for each of my org-mode project files. This allows me to search the current file and the archive when I need to dig up old information from the archives.

I don't need this often but it sure is handy on the occasions that I do need it.

```
;; Include agenda archive files when searching for things
(setq org-agenda-text-search-extra-files (quote (agenda-archives)))
```

15.4.5 Agenda view tweaks

The following agenda customizations control

- display of repeating tasks
- display of empty dates on the agenda
- task sort order

- start the agenda weekly view with `today`
- display of the grid
- habits at the bottom

I use a custom sorting function so that my daily agenda lists tasks in order of importance. Tasks on the daily agenda are listed in the following order:

1. tasks with times at the top so they are hard to miss
2. entries for today (active timestamp headlines that are not scheduled or deadline tasks)
3. deadlines due today
4. late deadline tasks
5. scheduled items for today
6. pending deadlines (due soon)
7. late scheduled items
8. habits

The lisp for this isn't particularly pretty but it works.
Here are the `.emacs` settings:

```
;; Show all future entries for repeating tasks
(setq org-agenda-repeating-timestamp-show-all t)

;; Show all agenda dates - even if they are empty
(setq org-agenda-show-all-dates t)

;; Sorting order for tasks on the agenda
(setq org-agenda-sorting-strategy
      (quote ((agenda habit-down time-up user-defined-up priority-down effo
               (todo category-up priority-down effort-up)
               (tags category-up priority-down effort-up)
               (search category-up))))

;; Start the weekly agenda today
```

```
(setq org-agenda-start-on-weekday nil)

;; Enable display of the time grid so we can see the marker for the current
(setq org-agenda-time-grid (quote((daily today remove-match)
                                  #("----------------" 0 16
                                    (org-heading t))
                                  (800 1000 1200 1400 1600 1800 2000))))

;; Display tags farther right
(setq org-agenda-tags-column -102)

;;
;; Agenda sorting functions
;;
(setq org-agenda-cmp-user-defined 'bh/agenda-sort)

(defun bh/agenda-sort (a b)
  "Sorting strategy for agenda items.
Late deadlines first, then scheduled, then non-late deadlines"
  (let (result num-a num-b)
    (cond
     ; time specific items are already sorted first by org-agenda-sorting-

     ; non-deadline and non-scheduled items next
     ((bh/agenda-sort-test 'bh/is-not-scheduled-or-deadline a b))

     ; deadlines for today next
     ((bh/agenda-sort-test 'bh/is-due-deadline a b))

     ; late deadlines next
     ((bh/agenda-sort-test-num 'bh/is-late-deadline '< a b))

     ; scheduled items for today next
     ((bh/agenda-sort-test 'bh/is-scheduled-today a b))

     ; pending deadlines last
     ((bh/agenda-sort-test-num 'bh/is-pending-deadline '< a b))

     ; late scheduled items next
     ((bh/agenda-sort-test-num 'bh/is-scheduled-late '> a b))
```

```
      ; finally default to unsorted
      (t (setq result nil)))
     result))

(defmacro bh/agenda-sort-test (fn a b)
  "Test for agenda sort"
  `(cond
    ; if both match leave them unsorted
    ((and (apply ,fn (list ,a))
          (apply ,fn (list ,b)))
     (setq result nil))
    ; if a matches put a first
    ((apply ,fn (list ,a))
     (setq result -1))
    ; otherwise if b matches put b first
    ((apply ,fn (list ,b))
     (setq result 1))
    ; if none match leave them unsorted
    (t nil)))

(defmacro bh/agenda-sort-test-num (fn compfn a b)
  `(cond
    ((apply ,fn (list ,a))
     (setq num-a (string-to-number (match-string 1 ,a)))
     (if (apply ,fn (list ,b))
         (progn
           (setq num-b (string-to-number (match-string 1 ,b)))
           (setq result (if (apply ,compfn (list num-a num-b))
                            -1
                          1)))
       (setq result -1)))
    ((apply ,fn (list ,b))
     (setq result 1))
    (t nil)))

(defun bh/is-not-scheduled-or-deadline (date-str)
  (and (not (bh/is-deadline date-str))
       (not (bh/is-scheduled date-str))))
```

```
(defun bh/is-due-deadline (date-str)
  (string-match "\texttt{Deadline:}" date-str))

(defun bh/is-late-deadline (date-str)
  (string-match "In *\\(-.*\\)d\.:" date-str))

(defun bh/is-pending-deadline (date-str)
  (string-match "In \\([^-]*\\)d\.:" date-str))

(defun bh/is-deadline (date-str)
  (or (bh/is-due-deadline date-str)
      (bh/is-late-deadline date-str)
      (bh/is-pending-deadline date-str)))

(defun bh/is-scheduled (date-str)
  (or (bh/is-scheduled-today date-str)
      (bh/is-scheduled-late date-str)))

(defun bh/is-scheduled-today (date-str)
  (string-match "\texttt{Scheduled:}" date-str))

(defun bh/is-scheduled-late (date-str)
  (string-match "Sched\.\(.*\)x:" date-str))
```

15.5 Checklist handling

Checklists are great for repeated tasks with lots of things that need to be done. For a long time I was manually resetting the check boxes to unchecked when marking the repeated task `DONE` but no more! There's a contributed `org-checklist` that can uncheck the boxes automagically when the task is marked done.

Add the following to your `.emacs`

```
(add-to-list 'load-path (expand-file-name "~/git/org-mode/contrib/lisp"))

(require 'org-checklist)
```

and then to use it in a task you simply set the property `RESET_CHECK_BOXES` to `t` like this

```
** TODO Invoicing and Archive Tasks [0/7]
```

```
\texttt{DEADLINE:} <2009-07-01 Wed +1m -0d>

:PROPERTIES:
:RESET_CHECK_BOXES: t
:END:

- [ ] Do task 1
- [ ] Do task 2
...
- [ ] Do task 7
```

15.6 Backups

`Backups that you have to work hard at don't get gone.`

I lost a bunch of data over 10 years ago due to not having a working backup solution. At the time I said `I'm not going to lose any important data ever again`. So far so good :)

My backups get done religiously. What does this have to do with org-mode? Not much really, other than I don't spend time doing backups – they just happen – which saves me time for other more interesting things.

My backup philosophy is to make it possible to recover your data – not necessarily easy. It doesn't have to be easy/fast to do the recovery because I'll rarely have to recover data from the backups. Saving time for recovery doesn't make sense to me. I want the backup to be fast and painless since I do those all the time.

I set up an automated network backup over 10 years ago that is still serving me well today. All of my systems gets daily backups to a network drive. These are collected monthly and written to an external removable USB disk.

Once a month my task for backups prompts me to move the current collection of montly backups to the USB drive for external storage. Backups take minimal effort currently and I'm really happy about that.

Since then `git` came into my life, so backups of `git` repositories that are on multiple machines is much less critical than it used to be. There is an automatic backup of everything pushed to the remote repository.

15.7 Handling blocked tasks

Blocked tasks are tasks that have subtasks which are not in a done todo state. Blocked tasks show up in a grayed font by default in the agenda.

To enable task blocking set the following variable:

```
(setq org-enforce-todo-dependencies t)
```

This setting prevents tasks from changing to `DONE` if any subtasks are still open. This works pretty well except for repeating tasks. I find I'm regularly adding `TODO` tasks under repeating tasks and not all of the subtasks need to be complete before the next repeat cycle.

You can override the setting temporarily by changing the task with `C-u C-u C-u C-c C-t` but I never remember that. I set a permanent property on the repeated tasks as follows:

```
* TODO New Repeating Task
  \texttt{SCHEDULED:} <2009-06-16 Tue +1w>

  :PROPERTIES:
  :NOBLOCKING: t
  :END:
...
** TODO Subtask
```

This prevents the `New Repeating Task` from being blocked if some of the items under it are not complete.

Occassionally I need to complete tasks in a given order. Org-mode has a property `ORDERED` that enforces this for subtasks.

```
* TODO Some Task
  :PROPERTY:
  :ORDERED: t
  :END:
** TODO Step 1
** TODO Step 2
** TODO Step 3
```

In this case you need to complete `Step 1` before you can complete `Step 2`, etc. and org-mode prevents the state change to a done task until the preceding tasks are complete.

15.8 Org Task structure and presentation

This section describes various org-mode settings I use to control how tasks are displayed while I work on my org mode files.

15.8.1 Controlling display of leading stars on headlines

Org-mode has the ability to show or hide the leading stars on task headlines. It's also possible to have headlines at odd levels only so that the stars and heading task names line up in sublevels.

I don't hide leading stars - I want to see the heading levels explicitly. When I tried the hide leading stars setting I found myself typing ' *' when adding a new heading and then the font lock shows I messed up and created a list instead.

To make org show leading stars use

```
(setq org-hide-leading-stars nil)
```

15.8.2 org-indent mode

I recently started using org-indent mode. I like this setting a lot. It removes the indentation in the org-file but displays it as if it was indented while you are working on the org file buffer.

org-indent mode displays as if org-odd-levels-only is true but it has a really clean look that I prefer over my old setup.

I have org-indent mode on by default at startup with the following setting:

```
(setq org-startup-indented t)
```

15.8.3 Show headings at odd levels only or odd-even levels

I've converted my files between odd-levels-only and odd-even using the functions `org-convert-to-odd-levels` and `org-convert-to-oddeven-levels` functions a number of times. I ended up going back to odd-even levels to reduce the amount of leading whitespace on tasks. I didn't find that lining up the headlines and tasks in odd-levels-only to be all that helpful.

```
(setq org-odd-levels-only nil)
```

15.8.4 Handling blank lines

Blank lines are evil :). They keep getting inserted in between headlines and I don't want to see them in collapsed (contents) views. When I use `TAB` to fold (cycle) tasks I don't want to see any blank lines but the default `org-cycle-separate-lines` setting hides single blank lines and reveals where extra empty lines are created in the document. This gives me an easy way to identify and eradicate these evil blank lines.

The following setting hides single blank lines inside folded contents of a tasks:

```
(setq org-cycle-separator-lines 2)
```

I find extra blank lines in lists and headings a bit of a nuisance. To get a body after a list you need to include a blank line between the list entry and the body – and indent the body appropriately. Most of my lists have no body detail so I like the look of collapsed lists with no blank lines better.

The following setting prevents creating blank lines before list items and headings:

```
(setq org-blank-before-new-entry (quote ((heading)
                                         (plain-list-item))))
```

15.8.5 Adding new tasks quickly without disturbing the current task content

To create new headings in a project file it is really convenient to use `C-RET`, `C-S-RET`, `M-RET`, and `M-S-RET`. This inserts a new headline possibly with a `TODO` keyword. With the following setting

```
(setq org-insert-heading-respect-content nil)
```

org inserts the heading at point for the `M-` versions and respects content for the `C-` versions. The respect content setting is temporarily turned on for the `C-` versions which adds the new heading after the content of the current item. This lets you hit `C-S-RET` in the middle of an entry and the new heading is added after the body of the current entry but still allow you to split an entry in the middle with `M-S-RET`.

15.8.6 Notes at the top

I enter notes for tasks with `C-c C-z` (or just `z` in the agenda). Changing tasks states also sometimes prompt for a note (e.g. moving to `WAITING` prompts for a note and I enter a reason for why it is waiting). These notes are saved at the top of the task so unfolding the task shows the note first.

```
(setq org-reverse-note-order nil)
```

15.8.7 Searching and showing results

Org-mode's searching capabilities are really effective at finding data in your org files. `C-c / /` does a regular expression search on the current file and shows matching results in a collapsed view of the org-file.

I have org-mode show the hierarchy of tasks above the matched entries and also the immediately following sibling task (but not all siblings) with the following settings:

```
(setq org-show-following-heading t)
(setq org-show-hierarchy-above t)
(setq org-show-siblings nil)
```

This keeps the results of the search relatively compact and mitigates accidental errors by cutting too much data from your org file with `C-k`. Cutting folded data (including the ...) can be really dangerous since it cuts text (including following subtrees) which you can't see. For this reason I always show the following headline when displaying search results.

15.8.8 Editing and Special key handling

Org-mode allows special handling of the C-a, C-e, and C-k keys while editing headlines. I also use the setting that pastes (yanks) subtrees and adjusts the levels to match the task I am pasting to. See the docstring (`C-h v org-yank-adjust-subtrees`) for more details on each variable and what it does.

I have `org-special-ctrl-a/e` reversed because most of the time I want to get to the beginning of the headline so the speed commands work and this still allows easy access to the beginning of the heading text when I need that.

```
(setq org-special-ctrl-a/e 'reversed)
(setq org-special-ctrl-k t)
(setq org-yank-adjusted-subtrees t)
```

15.9 Attachments

Attachments are great for getting large amounts of data related to your project out of your org-mode files. Before attachments came along I was including huge blocks of SQL code in my org files to keep track of changes I made to project databases. This bloated my org file sizes badly.

Now I can create the data in a separate file and attach it to my project task so it's easily located again in the future.

I set up org-mode to generate unique attachment IDs with `org-id-method` as follows:

```
(setq org-id-method (quote uuidgen))
```

Say you want to attach a file `x.sql` to your current task. Create the file data in `/tmp/x.sql` and save it.

Attach the file with `C-c C-a a` and enter the filename: `x.sql`. This generates a unique ID for the task and adds the file in the attachment directory.

```
** Attachments                                                    :ATTACH:
   :PROPERTIES:
   :Attachments: x.sql
   :ID:       f1d38e9a-ff70-4cc4-ab50-e8b58b2aaa7b
   :END:
```

The attached file is saved in `data/f1/d38e9a-ff70-4cc4-ab50-e8b58b2aaa7b/`. Where it goes exactly isn't important for me – as long as it is saved and retrievable easily. Org-mode copies the original file `/tmp/x.sql` into the appropriate attachment directory.

Tasks with attachments automatically get an `ATTACH` tag so you can easily find tasks with attachments with a tag search.

To open the attachment for a task use `C-c C-a o`. This prompts for the attachment to open and `TAB` completion works here.

The `ID` changes for every task header when a new `ID` is generated.

It's possible to use named directories for attachments but I haven't needed this functionality yet – it's there if you need it.

I store my org-mode attachments with my org files in a subdirectory `data`. These are automatically added to my `git` repository along with any other org-mode changes I've made.

15.10 Deadlines and Agenda Visibility

Deadlines and due dates are a fact or life. By default I want to see deadlines in the agenda 30 days before the due date.

The following setting accomplishes this:

```
(setq org-deadline-warning-days 30)
```

This gives me plenty of time to deal with the task so that it is completed on or before the due date.

I also use deadlines for repeating tasks. If the task repeats more often than once per month it would be always bugging me on the agenda view. For these types of tasks I set an explicit deadline warning date as follows:

```
** TODO Pay Wages
   \texttt{DEADLINE:} <2009-07-01 Wed +1m -0d>
```

This example repeats monthly and shows up in the agenda on the day it is due (with no prior warning). You can set any number of lead days you want on DEADLINES using -Nd where N is the number of days in advance the task should show up in the agenda. If no value is specified the default `org-deadline-warning-days` is used.

15.11 Exporting Tables to CSV

I generate org-mode tables with details of task specifications and record structures for some of my projects. My clients like to use spreadsheets for this type of detail.

It's easy to share the details of the org-mode table by exporting in HTML but that isn't easy for anyone else to work with if they need to edit data.

To solve this problem I export my table as comma delimited values (CSV) and then send that to the client (or read it into a spreadsheet and email the resulting spreadsheet file).

Org-mode can export tables as TAB or comma delimited formats. I set the default format to CSV with:

```
(setq org-table-export-default-format "orgtbl-to-csv")
```

Exporting to CSV format is the only one I use and this provides the default so I can just hit RETURN when prompted for the format.

To export the following table I put the cursor inside the table and hit `M-x org-table-export` which prompts for a filename and the format which defaults to orgtbl-to-csv from the setting above.

One	Two	Three
1	1	2
3	6	5
fred	kpe	mary
234.5	432.12	324.3

This creates the file with the following data

```
One,Two,Three
1,1,2
3,6,5
fred,kpe,mary
234.5,432.12,324.3
```

15.12 Visiting links

Links to emails, web pages, and other files are sprinkled all over my org files. The following setting control how org-mode handles opening the link.

```
(setq org-link-frame-setup ((vm . vm-visit-folder)
                            (gnus . org-gnus-no-new-news)
                            (file . find-file-other-window)))
```

I like to keep links in the same window so that I don't end up with a ton of frames in my window manager. I normally work in a full-screen window and having links open in the same window just works better for me.

15.13 Logging stuff

Most of my logging is controlled by the global `org-todo-keywords`

My logging settings are set as follows:

```
(setq org-log-done (quote time))
(setq org-log-into-drawer t)
```

My `org-todo-keywords` are set as follows:

```
(setq org-todo-keywords (quote ((sequence "TODO(t)" "NEXT(n)" "|" "DONE(d!/
                                 (sequence "WAITING(w@/!)" "SOMEDAY(s!)" "|"
                                 (sequence "OPEN(O!)" "|" "CLOSED(C!)"))))
```

This adds a log entry whenever a task moves to any of the following states:

- to or out of `DONE` status
- to `WAITING` status (with a note) or out of `WAITING` status
- to `SOMEDAY` status
- to `CANCELLED` status (with a note) or out of `CANCELLED` status
- to `OPEN` status

- to `CLOSED` status

I keep clock times and states in the `LOGBOOK` drawer to keep my tasks uncluttered. If a task is WAITING then the reason for why it is waiting is near the top of the LOGBOOK and unfolding the LOGBOOK drawer provides that information. From the agenda simply hitting `SPC` on the task will reveal the LOGBOOK drawer.

15.14 Limiting time spent on tasks

Org-mode has this great new feature for signalling alarms when the estimated time for a task is reached. I use this to limit the amount of time I spend on a task during the day.

As an example, I've been working on this document for over two months now. I want to get it finished but I can't just work on it solely until it's done because then nothing else gets done. I want to do a little bit every day but limit the total amount of time I spend documenting org-mode to an hour a day.

To this end I have a task

```
** NEXT Document my use of org-mode
   :LOGBOOK:...
   :PROPERTIES:
   :CLOCK_MODELINE_TOTAL: today
   :Effort:   1:00
   :END:
```

The task has an estimated effort of 1 hour and when I clock in the task it gives me a total in the mode-line like this

```
--:**  org-mode.org   91% (2348,73) Git:master  (Org Fly yas Font)-----[0:35/1:00 (Do
```

I've spent 35 minutes of my 1 hour so far today on this document and other help on IRC.

I set up an alarm so the Star Trek door chime goes off when the total estimated time is hit. (Yes I'm a Trekkie :))

```
(setq org-clock-sound "/usr/local/lib/tngchime.wav")
```

When the one hour time limit is hit the alarm sound goes off and a message states that I should be done working on this task. If I switch tasks and try to clock in this task again I get the sound each and every time I clock in the task. This nags me to go work on something else :)

You can use similar setups for repeated tasks. By default the last repeat time is recorded as a property when a repeating task is marked done. For repeating tasks the mode-line clock total counts since the last repeat time by default. This lets you accumulate time over multiple days and counts towards your estimated effort limit.

15.15 Habit Tracking

John Wiegley recently added support for Habit tracking to org-mode.

I have lots of habits (some bad) but I'd still like to improve and build new good habits. This is what habit tracking is for. It shows a graph on the agenda of how well you have been doing on developing your habits.

I have habits like:

- Hand wash the dishes
- 30 minute brisk walk
- Clean the house

etc. and most of these need a push to get done regularly. Logging of the done state needs to be enabled for habit tracking to work.

A habit is just like a regular task except it has a special `PROPERTY` value setting and a special `SCHEDULED` date entry like this:

```
** TODO Update Org Mode Doc
   \texttt{SCHEDULED:} <2009-11-21 Sat .+7d/30d>

   [2009-11-14 Sat 11:45]
   :PROPERTIES:
   :STYLE: habit
   :END:
```

This marks the task as a habit and separates it from the regular task display on the agenda. When you mark a habit done it shows up on your daily agenda the next time based on the first interval in the SCHEDULED entry (`.+7d`)

The special `SCHEDULED` entry states that I want to do this every day but at least every 2 days. If I go 3 days without marking it DONE it shows up RED on the agenda indicating that I have been neglecting this habit.

The world isn't going to end if you neglect your habits. You can hide and display habits quickly using the `K` key on the agenda.

These are my settings for habit tracking.

```
; Enable habit tracking (and a bunch of other modules)
(setq org-modules (quote (org-bbdb
                          org-bibtex
                          org-crypt
                          org-gnus
                          org-id
                          org-info
                          org-jsinfo
                          org-habit
                          org-inlinetask
                          org-irc
                          org-mew
                          org-mhe
                          org-protocol
                          org-rmail
                          org-vm
                          org-wl
                          org-w3m)))

; global STYLE property values for completion
(setq org-global-properties (quote (("STYLE_ALL" . "habit"))))
; position the habit graph on the agenda to the right of the default
(setq org-habit-graph-column 50)
```

During the day I'll turn off the habit display in the agenda with `K`. This is a persistent setting and since I leave my Emacs running for days at a time my habit display doesn't come back. To make sure I look at the habits daily I have the following settings to redisplay the habits in the agenda each day. This turns the habit display on again at 6AM each morning.

```
(run-at-time "06:00" 86400 '(lambda () (setq org-habit-show-habits t)))
```

15.16 Habits only log DONE state changes

I tend to keep habits under a level 1 task `* Habits` with a special logging property that only logs changes to the `DONE` state. This allows me to cancel a habit and not record a timestamp for it since that messes up the habit graph. Cancelling a habit just to get it off my agenda because it's undoable (like get up before 6AM) should not mark the habit as done today. I only cancel habits that repeat every day.

My habit tasks look as follows - and I tend to have one in every org file that can have habits defined

```
* Habits
  :PROPERTIES:
  :LOGGING:  DONE(!)
  :ARCHIVE:  %s_archive::* Habits
  :END:
```

15.17 Auto revert mode

I use git to synchronize my org-mode files between my laptop and my workstation. This normally requires saving all the current changes, pushing to a bare repo, and fetching on the other system. After that I need to revert all of my org-mode files to get the updated information.

I used to use `org-revert-all-org-buffers` but have since discovered `global-auto-revert-mode`. With this setting any files that change on disk where there are no changes in the buffer automatically revert to the on-disk version.

This is perfect for synchronizing my org-mode files between systems.

```
(setq global-auto-revert-mode t)
```

15.18 Handling Encryption

I used to keep my encrypted data like account passwords in a separate GPG encrypted file. Now I keep them in my org-mode files with a special tag instead. Encrypted data is kept in the org-mode file that it is associated with.

`org-crypt` allows you to tag headings with a special tag `crypt` and org-mode can keep data in these headings encrypted when saved to disk. You decrypt the heading temporarily when you need access to the data and org-mode re-encrypts the heading as soon as you save the file.

I use the following setup for encryption:

```
(require 'org-crypt)
; Encrypt all entries before saving
(org-crypt-use-before-save-magic)
(setq org-tags-exclude-from-inheritance (quote ("crypt")))
; GPG key to use for encryption
(setq org-crypt-key "F0B66B40")
```

`M-x org-decrypt-entry` will prompt for the passphrase associated with your encryption key and replace the encrypted data where the point is with the plaintext details for your encrypted entry. As soon as you save the file the data is re-encrypted for your key. Encrypting does not require prompting for the passphrase - that's only for looking at the plain text version of the data.

I tend to have a single level 1 encrypted entry per file (like `* Passwords`). I prevent the `crypt` tag from using inheritance so that I don't have encrypted data inside encrypted data. I found `M-x org-decrypt-entries` prompting for the passphrase to decrypt data over and over again (once per entry to decrypt) too inconvenient.

I leave my entries encrypted unless I have to look up data - I decrypt on demand and then save the file again to re-encrypt the data. This keeps the data in plain text as short as possible.

15.18.1 Auto Save Files

Emacs temporarily saves your buffer in an autosave file while you are editing your org buffer and a sufficient number of changes have accumulated. If you have decrypted subtrees in your buffer these will be written to disk in plain text which possibly leaks sensitive information. To combat this org-mode now asks if you want to disable the autosave functionality in this buffer.

Personally I really like the autosave feature. 99% of the time my encrypted entries are perfectly safe to write to the autosave file since they are still encrypted. I tend to decrypt an entry, read the details for what I need to look up and then immediately save the file again with `C-x C-s` which re-encrypts the entry immediately. This pretty much guarantees that my autosave files never have decrypted data stored in them.

I disable the default org crypt auto-save setting as follows:

```
(setq org-crypt-disable-auto-save nil)
```

15.19 Speed Commands

There's a new and exciting feature called `org-speed-commands` in the org-mode.

Speed commands allow access to frequently used commands when on the beginning of a headline - similar to one-key agenda commands. Speed commands are user configurable and org-mode provides a good set of default commands.

I have the following speed commands set up in addition to the defaults. I don't use priorities so I override the default settings for the 1, 2, and 3 keys. I also disable cycling with 'c' and add 'q' as a quick way to get back to the agenda.

```
(setq org-use-speed-commands t)
(setq org-speed-commands-user (quote (("1" . delete-other-windows)
                                      ("2" . split-window-vertically)
                                      ("3" . split-window-horizontally)
                                      ("h" . hide-other)
                                      ("k" . org-kill-note-or-show-branches
                                      ("q" . bh/show-org-agenda)
                                      ("r" . org-reveal)
                                      ("s" . org-save-all-org-buffers)
                                      ("z" . org-add-note)
                                      ("c" . self-insert-command)
                                      ("C" . self-insert-command)
                                      ("J" . org-clock-goto))))

(defun bh/show-org-agenda ()
  (interactive)
  (switch-to-buffer "*Org Agenda*")
  (delete-other-windows))
```

The variable `org-speed-commands-default` sets a lot of useful defaults for speed command keys. The default keys I use the most are `I` and `O` for clocking in and out and `t` to change todo state.

`J` jumps to the current or last clocking task.

`c` and `C` are disabled so they self insert. I use `TAB` and `S-TAB` for cycling - I don't need `c` and `C` as well. `TAB` works everywhere while `c` and `C` only works on the headline and sometimes I accidentally cycle when I don't intend to.

15.20 Org Protocol

Org protocol is a great way to create capture notes in org-mode from other applications. I use this to create tasks to review interesting web pages I visit in Firefox.

I have a special capture template set up for org-protocol to use (set up with the `w` key).

My org-mode setup for org-protocol is really simple. It enables org-protocol and creates a single org-protocol capture template as described in

`Capture Templates`.

```
(require 'org-protocol)
```

The bulk of the setup is in the Firefox application so that C-M-r on a page in Firefox will trigger the org-protocol capture template with details of the page I'm currently viewing in firefox.

I set up org-protocol in firefox as described in Keybindings for Firefox.

15.21 Require a final newline when saving files

The following setting was mainly for editing yasnippets where I want to be able to expand a snippet but stay on the same line. I used this mainly for replacing short strings or initials with full names for people during meeting notes. I now use `abbrev-mode-` for this and no longer need this setting.

```
(setq require-final-newline nil)
```

When I save a file in Emacs I want a final newline - this fits better with the source code projects I work on. This is the setting I use now:

```
(setq require-final-newline t)
```

15.22 Insert inactive timestamps and exclude from export

I insert inactive timestamps when working on org-mode files.

For remember tasks the timestamp is in the remember template but for regular structure editing I want the timestamp automatically added when I create the headline.

I have a function that is run by an org-mode hook to automatically insert the inactive timestamp whenever a headline is created.

```
(defun bh/insert-inactive-timestamp ()
  (interactive)
  (org-insert-time-stamp nil t t nil nil nil))

(defun bh/insert-heading-inactive-timestamp ()
  (save-excursion
    (org-return)
    (org-cycle)
    (bh/insert-inactive-timestamp)))

(add-hook 'org-insert-heading-hook 'bh/insert-heading-inactive-timestamp 'a
```

Everytime I create a heading with `M-RET` or `M-S-RET` the hook invokes the function and it inserts an inactive timestamp like this

```
** <point here>
   [2009-11-22 Sun 18:45]
```

This keeps an automatic record of when tasks are created which I find very useful.

I also have a short cut key defined to invoke this function on demand so that I can insert the inactive timestamp anywhere on demand.

```
(global-set-key (kbd "<f9> t") 'bh/insert-inactive-timestamp)
```

To prevent the timestamps from being exported in documents I use the following setting

```
(setq org-export-with-timestamps nil)
```

15.23 Return follows links

The following setting make `RET` insert a new line instead of opening links. This setting is a love-hate relationship for me. When it first came out I immediately turned it off because I wanted to insert new lines in front of my links and `RET` would open the link instead which at the time I found extremely annoying. I've used it for a while with it set but ultimately turned it off again.

```
(setq org-return-follows-link nil)
```

15.24 Highlight clock when running overtime

The current clocking task is displayed on the modeline. If this has an estimated time and we run over the limit I make this stand out on the modeline by changing the background to red as follows

```
(custom-set-faces
  ;; custom-set-faces was added by Custom.
  ;; If you edit it by hand, you could mess it up, so be careful.
  ;; Your init file should contain only one such instance.
  ;; If there is more than one, they won't work right.
 '(org-mode-line-clock ((t (:background "grey75" :foreground "red" :box (:l
```

15.25 Meeting Notes

I take meeting notes with org-mode. I record meeting conversations in point-form using org-mode lists. If action items are decided on in the meeting I'll denote them with a bullet and a TODO: or DONE: flag.

A meeting is a task and it is complete when the meeting is over. The body of the task records all of the interesting meeting details. If TODO items are created in the meeting I make separate TODO tasks from those.

I use the function `bh/prepare-meeting-notes` to prepare the meeting notes for emailing to the participants (in a fixed-width font like "Courier New"). As soon as the meeting is over the notes are basically ready for distribution – there's not need to waste lots of time rewriting the minutes before they go out. I haven't bothered with fancy HTML output – the content is more important than the style.

```
** TODO Sample Meeting
   - Attendees
     - [ ] Joe
     - [X] Larry
     - [X] Mary
     - [X] Fred
   - Joe is on vacation this week
   - Status Updates
     + Larry
       - did this
       - and that
       - TODO: Needs to follow up on this
     + Mary
       - got a promotion for her recent efforts
     + Fred
       - completed all his tasks 2 days early
       - needs more work
       - DONE: everything

** TODO Sample Meeting
   - Attendees
     - [ ] Joe
     - [X] Larry
     - [X] Mary
     - [X] Fred
   - Joe is on vacation this week
```

```
   - Status Updates
     + Larry
       - did this
       - and that
>>>>>>>> TODO: Needs to follow up on this
     + Mary
       - got a promotion for her recent efforts
     + Fred
       - completed all his tasks 2 days early
       - needs more work
>>>>>>>> DONE: everything
```

Here is the formatting function. Just highlight the region for the notes and it turns tabs into spaces, and highlights todo items. The resulting notes are in the kill buffer ready to paste to another application.

```
(defun bh/prepare-meeting-notes ()
  "Prepare meeting notes for email
   Take selected region and convert tabs to spaces, mark TODOs with leading
  (interactive)
  (let (prefix)
    (save-excursion
      (save-restriction
        (narrow-to-region (region-beginning) (region-end))
        (untabify (point-min) (point-max))
        (goto-char (point-min))
        (while (re-search-forward "^\\( *-\\) \\(TODO\\|DONE\\): " (point-
          (replace-match (concat (make-string (length (match-string 1)) ?>)
        (goto-char (point-min))
        (kill-ring-save (point-min) (point-max))))))
```

15.26 Highlights persist after changes

I'm finding I use org-occur `C-c / /` a lot when trying to find details in my org-files. The following setting keeps the highlighted results of the search even after modifying the text. This allows me to edit the file without having to reissue the org-occur command to find the other matches in my file.

```
(setq org-remove-highlights-with-change nil)
```

15.27 Getting up to date org-mode info documentation

I use the org-mode info documentation from the git repository so I set up emacs to find the info files from git before the regular (out of date) system versions.

```
(add-to-list 'Info-default-directory-list "~/git/org-mode/doc")
```

15.28 Prefer future dates or not?

By default org-mode prefers dates in the future. This means that if today's date is May 2 and you enter a date for April 30th (2 days ago) org-mode will jump to April 30th of next year. I used to find this annoying when I wanted to look at what happened last Friday since I have to specify the year. Now I've trained my fingers to go back relatively in the agenda with `b` so this isn't really an issue for me anymore.

To make org-mode prefer the current year when entering dates I set the following variable:

```
(setq org-read-date-prefer-future nil)
```

I now have this variable set to `t`.

15.29 Automatically change list bullets

I take point-form notes during meetings. Having the same list bullet for every list level makes it hard to read the details when lists are indented more than 3 levels.

Org-mode has a way to automatically change the list bullets when you change list levels.

Current List Bullet	Next indented list bullet
+	-
-	-
1.	-
1)	-

```
(setq org-list-demote-modify-bullet (quote (("+" . "-")
                                            ("*" . "-")
                                            ("1." . "-")
                                            ("1)" . "-"))))
```

15.30 Remove indentation on agenda tags view

I don't like the indented view for sublevels on a tags match in the agenda but I want to see all matching tasks (including sublevels) when I do a agenda tag search (`F12 m`).

To make all of the matched headings for a tag show at the same level in the agenda set the following variable:

```
(setq org-tags-match-list-sublevels t)
```

15.31 Fontify source blocks natively

I use babel for including source blocks in my documents with

```
#+begin_src LANG
  ...
#+end_src
```

where LANG specifies the language to use (ditaa, dot, sh, emacs-lisp, etc) This displays the language contents fontified in both the org-mode source buffer and the exported document.

See this Git Repository synchronization in this document for an example..

15.32 Agenda persistent filters

This is a great feature! Persistent agenda filters means if you limit a search with `/ TAB SomeTag` the agenda remembers this filter until you change it.

Enable persistent filters with the following variable

```
(setq org-agenda-persistent-filter t)
```

The current filter is displayed in the modeline as `{+SomeTag}` so you can easily see what filter currently applies to your agenda view.

I use this with `FILETAGS` to limit the displayed results to a single client or context.

15.33 Add tags for flagged entries

Everyone so often something will come along that is really important and you know you want to be able to find it back fast sometime in the future.

For these types of notes and tasks I add a special `:FLAGGED:` tag. This tag gets a special fast-key `?` which matches the search key in the agenda

for flagged items. See Tags for the setup of `org-tag-alist` for the `FLAGGED` entry.

Finding flagged entries is then simple - just `F12 ?` and you get them all.

15.34 Prevent horizontal window splitting

Emacs 23 wants to split the window both horizontally and vertically based on screen usage. With today's widescreen monitors this often means we split the window horizontally instead of vertically.

I found this change obtrusive and turn it off with the following setting.

```
(setq split-width-threshold 9999)
```

15.35 Mail links open compose-mail

The following setting makes org-mode open `mailto:` links using compose-mail.

```
(setq org-link-mailto-program (quote (compose-mail "%a" "%s")))
```

15.36 Composing mail from org mode subtrees

It's possible to create mail from an org-mode subtree. I use `C-c M-o` to start an email message with the details filled in from the current subtree. I use this for repeating reminder tasks where I need to send an email to someone else. The email contents are already contained in the org-mode subtree and all I need to do is `C-c M-o` and any minor edits before sending it off.

15.37 Use smex for M-x ido-completion

I discovered smex for IDO-completion for M-x commands after reading a post of the org-mode mailing list. I actually use M-x a lot now because IDO completion is so easy.

Here's the smex setup I use

```
(add-to-list 'load-path (expand-file-name "~/.emacs.d"))
(require 'smex)
(smex-initialize)

(global-set-key (kbd "M-x") 'smex)
(global-set-key (kbd "M-X") 'smex-major-mode-commands)
```

15.38 Use Emacs bookmarks for fast navigation

I've started using emacs bookmarks to save a location and return to it easily. Normally I want to get back to my currently clocking task and that's easy - just hit `F11`. When I'm working down a long checklist I find it convenient to set a bookmark on the next item to check, then go away and work on it, and return to the checkbox to mark it done.

I use Emacs bookmarks for this setup as follows:

```
;; Bookmark handling
;;
(global-set-key (kbd "<C-f6>") '(lambda () (interactive) (bookmark-set "SAV
(global-set-key (kbd "<f6>") '(lambda () (interactive) (bookmark-jump "SAVE
```

When I want to save the current location I just hit `C-f6` and then I can return to it with `f6` anytime. I overwrite the same bookmark each time I set a new position.

15.39 Using org-mime to email

I'm experimenting with sending mime mail from org. I've added `C-c M=o` key bindings in the `org-mode-hook` to generate mail from an org-mode subtree.

```
(require 'org-mime)
```

15.40 Remove multiple state change log details from the agenda

I skip multiple timestamps for the same entry in the agenda view with the following setting.

```
(setq org-agenda-skip-additional-timestamps-same-entry t)
```

This removes the clutter of extra state change log details when multiple timestamps exist in a single entry.

15.41 Drop old style references in tables

I drop the old A3/B4 style references from tables when editing with the following setting.

```
(setq org-table-use-standard-references (quote from))
```

15.42 Use system settings for file-application selection

To get consistent applications for opening tasks I set the `org-file-apps` variable as follows:

```
(setq org-file-apps (quote ((auto-mode . emacs)
                            ("\\.mm\\'" . system)
                            ("\\.x?html?\\'" . system)
                            ("\\.pdf\\'" . system))))
```

This uses the entries defined in my system `mailcap` settings when opening file extensions. This gives me consistent behaviour when opening an link to some HTML file with `C-c C-o` or when previewing an export.

15.43 Use the current window for the agenda

```
; Overwrite the current window with the agenda
(setq org-agenda-window-setup 'current-window)
```

15.44 Delete IDs when cloning

```
(setq org-clone-delete-id t)
```

15.45 Propagate STARTED to parent tasks

When a task is marked `STARTED` (either manually or by clocking it in) the `STARTED` state propagates up the tree to any parent tasks of this task that are `TODO` or `NEXT`. As soon as I work on the first `NEXT` task in a tree the project is also marked `STARTED`. This helps me keep track of things that are in progress.

Here's the setup I use to propagate `STARTED` to parent tasks:

```
;; Mark parent tasks as started
(defvar bh/mark-parent-tasks-started nil)

(defun bh/mark-parent-tasks-started ()
  "Visit each parent task and change TODO states to STARTED"
  (unless bh/mark-parent-tasks-started
    (when (equal state "STARTED")
      (let ((bh/mark-parent-tasks-started t))
        (save-excursion
          (while (org-up-heading-safe)
            (when (member (nth 2 (org-heading-components)) (list "TODO" "NE
```

```
                (org-todo "STARTED")))))))))

(add-hook 'org-after-todo-state-change-hook 'bh/mark-parent-tasks-started '
```

16 Things I Don't Use

This is a partial list of things I know about but do not use. `org-mode` is huge with tons of features. There are features out there that I don't know about yet or haven't explored so this list is not going to be complete.

16.1 Task Priorities

I use the agenda to figure out what to do work on next. I don't use priorities at all. I've played with them in the past and always go back to using no priorities.

I disable the priority setting keys in org-mode using

```
(setq org-enable-priority-commands nil)
```

16.2 Archive Sibling

This was a cute idea but I find archiving entire complete subtrees better. I don't mind having a bunch of tasks marked `DONE` (but not archived)

16.3 Cycling plain lists

Org mode can fold (cycle) plain lists. I don't use this feature.

```
(setq org-cycle-include-plain-lists nil)
```

Turning this on makes my `F5` sparse trees way too big. I just want to see the headlines.

16.4 Strike-through emphasis

Strike-through emphasis is just unreadable and tends to only show up when pasting data from other files into org-mode. This just removes the strike-through completely which I find a lot nicer.

```
(setq org-emphasis-alist (quote (("*" bold "<b>" "</b>")
                                 ("/" italic "<i>" "</i>")
                                 ("_" underline "<span style=\"text-decorat
```

```
("=" org-code "<code>" "</code>" verbatim)
("~" org-verbatim "<code>" "</code>" verba
```

16.5 Subscripts and Superscripts

I don't currently write documents that need subscripts and superscript support. I disable handling of `_` and `^` for subscript and superscripts with

```
(setq org-use-sub-superscripts nil)
```

17 Using Git for Automatic History, Backups, and Synchronization

Editing folded regions of your org-mode file can be hazardous to your data. My method for dealing with this is to put my org files in a `Git` source repository.

My setup saves all of my org-files every hour and creates a commit with my changes automatically. This lets me go back in time and view the state of my org files for any given hour over the lifetime of the document. I've used this once or twice to recover data I accidentally removed while editing folded regions.

17.1 Automatic Hourly Commits

My Emacs setup saves all org buffers at 1 minute before the hour using the following code in my `.emacs`

```
(run-at-time "00:59" 3600 'org-save-all-org-buffers)
```

A `cron` job runs at the top of the hour to commit any changes just saved by the call to `org-save-all-org-buffers` above. I use a script to create the commits so that I can run it on demand to easily commit all modified work when moving from one machine to another.

`crontab` details:

```
0 * * * * ~/bin/org-git-sync.sh >/dev/null
```

17.1.1 ~/bin/org-git-sync.sh

Here is the shell script I use to create a `git` commit for each of my org-repositories. This loops through multiple repositories and commits any modified files. I have the following org-mode repositories:

- org

  for all of my organization project files and todo lists

- doc-norang.ca

  for any changes to documents under http://doc.norang.ca/

- www.norang.ca

  for any changes to my other website http://www.norang.ca/

This script does not create empty commits - `git` only creates a commit if something was modified.

```
#!/bin/sh
# Add org file changes to the repository
REPOS="org doc.norang.ca www.norang.ca"

for REPO in $REPOS
do
    echo "Repository: $REPO"
    cd ~/git/$REPO
    # Remove deleted files
    git ls-files --deleted -z | xargs -0 git rm >/dev/null 2>&1
    # Add new files
    git add . >/dev/null 2>&1
    git commit -m "$(date)"
done
```

I use the following `.gitignore` file in my org-mode `git` repositories to keep export generated files out of my `git` repositories. If I include a graphic from some other source than ditaa or graphviz then I'll add it to the repository manually. By default all PNG graphic files are ignored (since I assume they are produced by ditaa during export)

```
core
core.*
*.html
*~
.#*
\#*\#
*.txt
*.tex
```

```
*.aux
*.dvi
*.log
*.out
*.ics
*.pdf
*.xml
*.org-source
*.png
*.toc
```

17.2 Git - Edit files with confidence

I use `git` in all of my directories where editing a file should be tracked.

This means I can edit files with confidence. I'm free to change stuff and break things because it won't matter. It's easy to go back to a previous working version or to see exactly what changed since the last commit. This is great when editing configuration files (such as apache webserver, bind9 DNS configurations, etc.)

I find this extremely useful where your edits might break things and having `git` tracking the changes means if you break it you can just go back to the previous working version easily. This is also true for package upgrades for software where the upgrade modifies the configuration files.

I have every version of my edits in a local `git` repository.

17.3 Git Repository synchronization

I acquired a Eee PC 1000 HE which now serves as my main road-warrior laptop replacing my 6 year old Toshiba Tecra S1.

I have a server on my LAN that hosts bare git repositories for all of my projects. The problem I was facing is I have to leave in 5 minutes and want to make sure I have up-to-date copies of everything I work on when I take it on the road (without Internet access).

To solve this I use a server with bare git repositories on it. This includes my org-mode repositories as well as any other git repositories I'm interested in.

Just before I leave I run the `git-sync` script on my workstation to update the bare git repositories and then I run it again on my Eee PC to update all my local repositories on the laptop. For any repositories that give errors due to non-fast-forward merges I manually merge as required and rerun `git-sync`

until it reports no errors. This normally takes a minute to two to do. Then I grab my Eee PC and leave. When I'm on the road I have full up-to-date history of all my git repositories.

The `git-sync` script replaces my previous scripts with an all-in-one tool that basically does this:

- for each repository on the current system
  - fetch objects from the remote
  - for each branch that tracks a remote branch
    - Check if the ref can be moved
      - fast-forwards if behind the remote repository and is fast-forwardable
      - Does nothing if ref is up to date
      - Pushes ref to remote repository if ref is ahead of remote repository and fast-forwardable
      - Fails if ref and remote have diverged

This automatically advances changes on my 35+ git repositories with minimal manual intervention. The only time I need to manually do something in a repository is when I make changes on my Eee PC and my workstation at the same time - so that a merge is required.

Here is the `git-sync` script

```
#!/bin/sh
#

# Local bare repository name
syncrepo=norang
reporoot=~/git

# Display repository name only once
log_repo() {
  [ "x$lastrepo" == "x$repo" ] || {
    printf "\nREPO: ${repo}\n"
    lastrepo="$repo"
  }
}

# Log a message for a repository
```

```
log_msg() {
  log_repo
  printf "␣␣$1\n"
}

# fast-forward reference $1 to $syncrepo/$1
fast_forward_ref() {
  log_msg "fast-forwarding␣ref␣$1"
  current_ref=$(cat .git/HEAD)
  if [ "x$current_ref" = "xref:␣refs/heads/$1" ]
  then
    # Check for dirty index
    files=$(git diff-index --name-only HEAD --)
    git merge refs/remotes/$syncrepo/$1
  else
    git branch -f $1 refs/remotes/$syncrepo/$1
  fi
}

# Push reference $1 to $syncrepo
push_ref() {
  log_msg "Pushing␣ref␣$1"
  if ! git push --tags $syncrepo $1
  then
    exit 1
  fi
}

# Check if a ref can be moved
#   - fast-forwards if behind the sync repo and is fast-forwardable
#   - Does nothing if ref is up to date
#   - Pushes ref to $syncrepo if ref is ahead of syncrepo and fastforwardal
#   - Fails if ref and $syncrop/ref have diverged
check_ref() {
  revlist1=$(git rev-list refs/remotes/$syncrepo/$1..$1)
  revlist2=$(git rev-list $1..refs/remotes/$syncrepo/$1)
  if [ "x$revlist1" = "x" -a "x$revlist2" = "x" ]
  then
    # Ref $1 is up to date.
    :
```

```
  elif [ "x$revlist1" = "x" ]
  then
    # Ref $1 is behind $syncrepo/$1 and can be fast-forwarded.
    fast_forward_ref $1 || exit 1
  elif [ "x$revlist2" = "x" ]
  then
    # Ref $1 is ahead of $syncrepo/$1 and can be pushed.
    push_ref $1 || exit 1
  else
    log_msg "Ref $1 and $syncrepo/$1 have diverged."
    exit 1
  fi
}

# Check all local refs with matching refs in the $syncrepo
check_refs () {
  git for-each-ref refs/heads/* | while read sha1 commit ref
  do
    ref=${ref/refs\/heads\//}
    git for-each-ref refs/remotes/$syncrepo/$ref | while read sha2 commit r
    do
      if [ "x$sha2" != "x" -a "x$sha2" != "x" ]
      then
        check_ref $ref || exit 1
      fi
    done
  done
}

# For all repositories under $reporoot
#   Check all refs matching $syncrepo and fast-forward, or push as necessar
#   to synchronize the ref with $syncrepo
#   Bail out if ref is not fastforwardable so user can fix and rerun
time {
  retval=0
  if find $reporoot -type d -name '*.git' | {
      while read repo
      do
        repo=${repo/\/.git/}
        cd ${repo}
```

```
        upd=$(git remote update $syncrepo 2>&1 || retval=1)
        [ "x$upd" = "xFetching $syncrepo" ] || {
          log_repo
          printf "$upd\n"
        }
        check_refs || retval=1
      done
      exit $retval
    }
  then
    printf "\nAll done.\n"
  else
    printf "\nFix and redo.\n"
  fi
}

exit $retval
```

18 Change History - What's new

This is version

```
v1.0.0-117-g03ea
```

of this document is created using the publishing features of `org-mode`.

The source for this document can be found as colorized HTML and plain text org file.

I try to update this document about once a month.

The change history for this document can be found at git://git.norang.ca/org-mode-doc.git.