文書作成補助ツールornbの開発

関西学院大学理工学部
情報科学科 西谷研究室
27015429
河野大登

2019年3月

概 要

Emacs 上で動作する文書作成の環境として Org-mode が存在する．Org-mode はプレーンテキストであるため，エディタ上での操作に優れ，文章構成を記述する Mark up が覚えやすいという利点がある．

しかし，文書作成の環境として，文書を公開する際，文章中に書かれたリンクを公開してはいけない場合や，リンク切れを起こしている場合があるという課題が存在した．

本研究で開発した ornb ではこれらを解決し，文書作成の補助を行うことを目指す．

本研究で開発したアプリケーションは，文書を公開するにあたり任意の pdf ファイルのリンク表示を変更することを可能とし，リンク切れを起こしている場合はターミナル上でリンク切れを起こしているリンクを赤で表示することで確認できるものとなった．

一方で，リンクの抽出において一つの種類しか抽出できないため，pdf ファイル，jpeg ファイル，png ファイル等複数を抽出することを可能にできるようにするべきであった．また，リンク切れのチェックについては，ファイルシステムを見直し，リンク切れの原因であるファイルの保存場所の違いを解決するために，擬似的なセミラティス構造を持つファイルシステムを開発することで解決すると考えられるため，リンクの表示変更，リンク切れのチェックのいずれも継続的な改良が求められる．

目 次

表 目 次

図 目 次

第1章　序論

1.1　背景

Python 言語で頻繁に利用される対話型プログラミング環境に Jupyter Notebook がある．この Notebook は，その基本コンセプトとして，イギリス出身の理論物理学者である Stephen Wolfram が，使い慣れた文書管理メタファーを利用でき，ライブ計算，任意の動的インターフェース，完全なタイプセット入力，画像入力，自動コード注釈付け，完全な高レベルプログラム的インターフェースなどをサポートするものとした [1]．これらを元に開発されたのが Mathematica や，Maple のような主に数式処理を web ブラウザー上で行うソフトウェアである．Mathematica や Maple は,Web ブラウザー上でコードを書き，実行し，実行結果を表示する．Jupyter notebook も同様に Web ブラウザー上でコードを書き，実行を行うことができる．また，プログラムのみならず Markdown 形式で文書を作成することができるため，レポートや論文を作成することを可能にした非常に便利な GUI ベースの扱いやすいオープンソースソフトウェアである．

しかし，Jupyter Notebook には下記のような欠点が存在する.

1. プレーンテキストではない

   まず初めに，プレーンテキストではないため，タイトル，目次，文章，plot やグラフなどの抽出に手間がかかる．また，Emacs などのエディタで Jupyter Notebook を使用する場合，プログラミング言語も表示されてしまうため，シンプルな文章ではなく複雑になってしまう．

2. 文章記述が Markdown

   次に，文書作成を行う場合，Markdown 形式であることだ．Markdown 形式は HTML や LaTeX を生成することができ便利であるが，覚えるのに時間がかかってしまう．また，HTML に変換する際，変更後の確認をその場ですることができないため，変換後の出力の確認をするために HTML を開き直す必要がある [2].

1.2　目的

西谷研究室では文書作成を行う環境として Org-mode を使用し，日々の研究の過程やメモに加え，教授は授業のテキスト作成も行なっている．Org-mode は，Emacs 上で動作する文書作成の環境で，Markdown 形式とは異なり，極めて覚えやすく，容易に文書を作成できるという利点がある．

さらに，プレーンテキストであるため，エディタでの使用に優れ，目次やタイトル，文章などの情報の抽出を容易に行うことできる．それに加え，Jupyter Notebook と同様にコードの実行，リンク付け, 図や表の表示，ライブ計算，HTML や LaTeX への変換等の機能も兼ね備えている．Org-mode と Jupyter Notebook を比較したものが，下の表 1.1である．

表 1.1: Org-mode と Jupyter Notebook の比較.

	Org-mode	Jupyter Notebook
プレーンテキスト	◯	x
コードの実行	◯	◯
HTML・LaTeX への変換	◯	◯
リンク・図・表	◯	◯
覚えやすさ	◎	△

このように，文書作成においてプレーンテキストであること，覚えやすいことの二つの点で Org-mode の方が優れていることがわかる．

しかし，文書作成の環境として課題が存在した．それは，文書を公開する際に，文章中に書かれたリンクを公開してはいけない場合がある．例えば，書籍の pdf ファイルのリンクは Web ブラウザ上で公開されていない限り，公開してはいけない．

もう一つは，リンク切れを起こしている場合が存在したことである．多数のリンクが存在する場合，自らリンク切れに気づくことは難しい．

そこで本研究では，Org-mode で作成された，文書中に存在する pdf のリンクを公開できない場合にのみ指定したタグや，URL 等に変換し，リンク切れが存在する場合はそれを知らせる機能を提供する CUI アプリケーション ornb を開発する．

第2章　開発環境と手法

2.1　テスト駆動開発 (TDD)

テスト駆動開発のゴールは「動作するきれいなコード」である．テスト駆動開発は，まず初めに「動作するもの」つまり「テスト」を作成し，そのあとに「きれいな」に取り組む手法でありアジャイルソフトウェア開発に欠かせない開発手法である [3]．テスト駆動開発において以下のような順序で進める．

1. テストを書く
2. コンパイラを通す
3. テストを実行し，エラーを確認する
4. テストが正しい動作であることを確認する
5. 重複を排除する

本研究では，初めに，テストとしてメソッドやクラスを定義せず，ただ正しい動作をするプログラムの作成を行なった．これが「動作するもの」の作成である．テストでは別の org ファイルをテスト用に準備し少ないデータで確認を繰り返すことで作業の効率化を図った．

正しい動作を行い，エラーがない事を確認できたとき，初めてメソッドとして定義した．これを各動作毎に繰り返すことで，メソッドとしてその動作での目的は何なのか明確になる．このとき，たくさんのメソッドが存在しているため，次はこれをクラスとして定義した．複数のメソッドを一つのクラスにすることで，多くの重複を排除することができ,「きれいな」プログラムに近づいた．最後に，クラスの継承を行うことで，重複の排除を行うことができ，シンプルで美しい「動作するきれいなコード」の作成を行うことができた．

2.2　Org-mode

Org-mode は，Emacs 上で動作するアウトライナーでありプレーンテキストの文書作成環境である．ノートの保存，TODO リストの管理，スケジュールや時間の管理，また発表原稿やスライドの作成など様々な用途に対応している．また，コードの実行はもちろん，リンク付け, テーブル表記の入力，図や表の表示，ライブ計算，HTML や LaTeX への変換等の機能も兼ね備えている [4]．

以下が Org-mode 上での一般的なリンクのフォーマットである [5]．

[[リンク][項目名]]　または　[[リンク]]

こうすることで，図 2.1のようにリンクとして表示される．

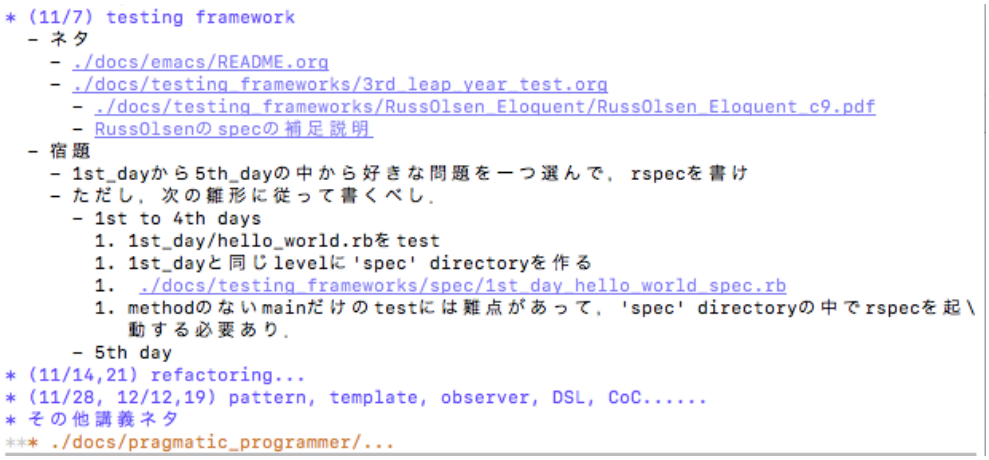

図 2.1: Org ファイルのリンク表記.

マウスのクリックか，リンク上にカーソルがある時に，

Control-c Control-o

のコマンドを入力することでリンクを開くことができる．
また，HTML への変換は，

Control-c Control-e h

のコマンドで変換される．
LaTeX への変換は，

Control-c Control-e l l

のコマンドで変換される．
Org-mode は Markdown 形式とは違い，覚え易いことが特徴であり，文書を非常にシンプルに作成することができるため，HTML や LaTeX に変換したときのミスが少ないという利点がある．
また，各見出しや，項目毎に内容を表示するか非表示にするのかを tab キーで変更することができる．図 2.2は

Org ファイルの見出しとその他講義ネタの./docs/pragmatic_programmer/

の内容のみを表示した場合である．

図 2.2: Org ファイルの見出しとその他講義ネタの./docs/pragmatic_programmer/の内容のみを表示.

また，図 2.3は, 見出しのみ表示し，内容は表示しない場合である.

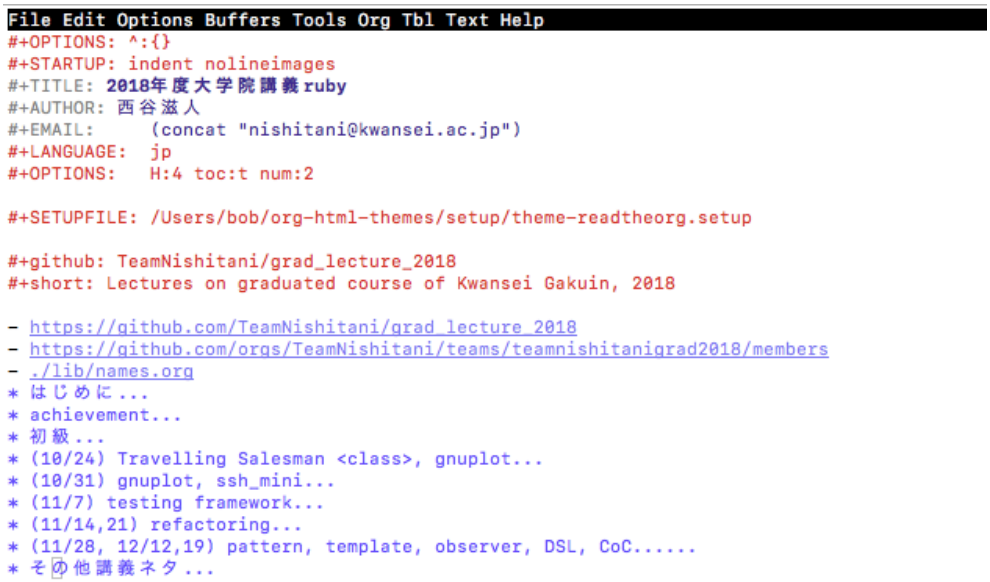

図 2.3: Org ファイルの見出しのみを表示.

このように，編集箇所のみを表示することができるため，規模の大きな文書でも文章構成を確認しながらの作成を可能にし，章や項目毎の内容の確認を容易に行うことができる利点がある．

2.3 Emacs

本研究は，shell ツールの開発であるためエディタが必須である．Emacs は，一般的に使用されているエディタの一つであり，様々な拡張機能を持った高機能エディタである．西谷研究室で推奨エディタとして全員が使用しており，Org-mode も使用できることから Emacs をエディタとして用いる．

2.4 RubyGems

RubyGems は，Ruby 言語用のパッケージ管理ツールであり，通称 gem と呼ばれている．ライブラリの作成や公開，インストールを助けるなど，無料で誰でも使用することが可能であり，直接ウェブサイトを閲覧することや，gem コマンドを使用してそれらを探すことができる [6] .

2.5 GitHub

GitHub は，Git の仕組みを採用したプログラムのソースコードや画像データ等を保存や共有することが可能な開発プラットフォームである．本研究では，コードやシステムのフィードバックをもらうために GitHub を用いて開発を進めた．

第3章　ornbの実装

3.1　ornbの使用方法

まず初めに図3.1に従って，全体の流れを解説する

1. 用意されたorgファイルから，pdfのリンクを抽出しcsvファイルにそのデータとタグを保存する．
2. csvファイルを開き保存されたリンクの中で，変更する必要があるリンクのタグを，表示したい名前に変更する．
3. 変更されたデータをcsvファイルに上書きする．
4. orgファイルとcsvファイルを比較し，リンクに変更が加えらている場合はその変更後を表示するようにし，新しいorgファイルに書き出す．
5. 新しく作成されたorgファイルを自動でhtmlファイルに変換する．

という流れである．

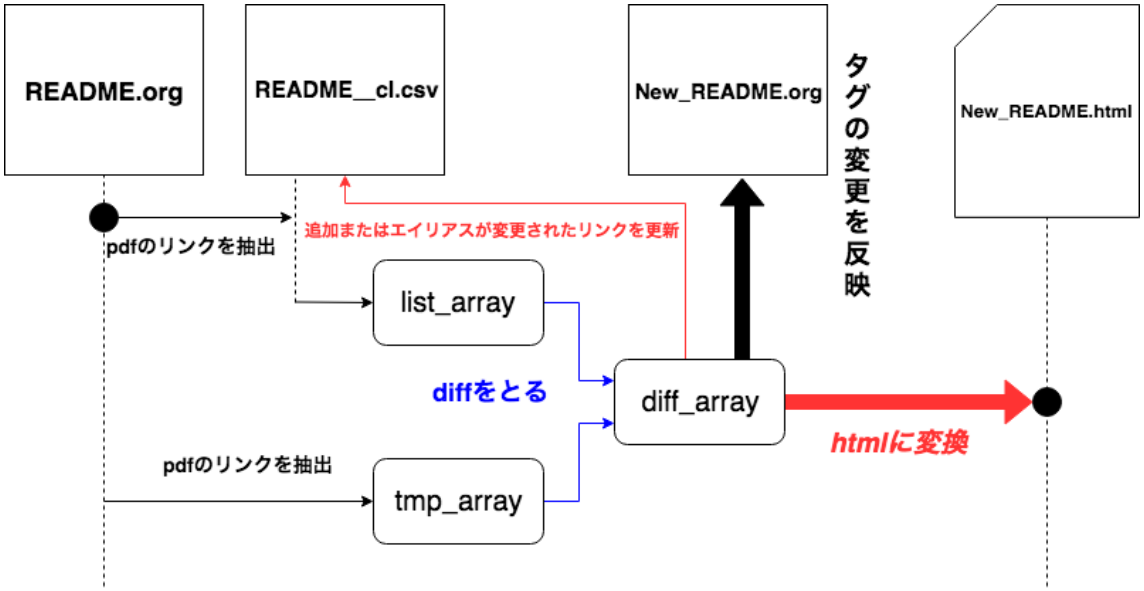

図 3.1: 全体の流れ.

3.2 mk_csv

mk_csv では，抽出した pdf のリンクを citation list として csv ファイルを作成し，後に編集を行うことができるようにする．

まず，org ファイルから pdf のリンクを抽出し, 図 3.2のように csv ファイルを作成する．org ファイルの名前が

```
README.org
```

の場合，

```
README_cl.csv
```

という csv ファイルが作成される．

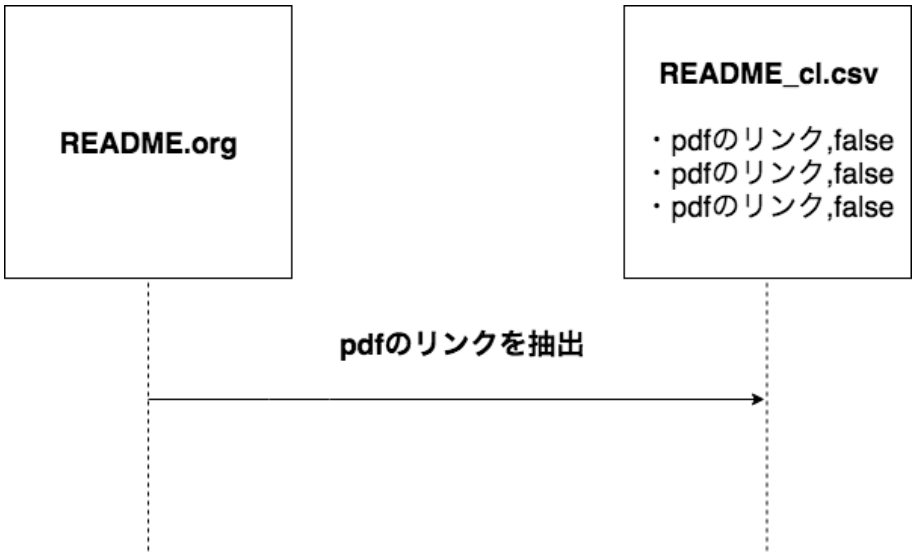

図 3.2: csv ファイルを作成する流れ.

csv ファイルには，以下のように

```
pdf のリンク，false
```

の形で一行ずつデータとして保存される.

```
./docs/emacs/emacs_key_bind3.pdf,false
./docs/emacs/FolderConfig.pdf,false
                    (中略)
./docs/happy_ruby/TanoshiiRuby_v5_c1.pdf,false
./docs/happy_ruby/TanoshiiRuby_v5_c2-3.pdf,false
./docs/happy_ruby/RussOlsen_EloquentRuby_c5.pdf,false
./docs/happy_ruby/RussOlsen_EloquentRuby_c1.pdf,false
```

この時，デフォルトで false をタグとして設定しており，タグが何も変更されない，つまり false であるとき, 新しく作成された org ファイルでの表示は pdf のリンクをそのまま表示する．

次に，csv ファイルに保存されているデータを一行ずつ list_array という配列に格納する．

また，既に csv ファイルが存在する場合は，csv ファイルを作成せずに

```
pdf のリンク,false
```

のデータを一行ずつ tmp_array という配列に格納する．つまり tmp_array には, 前回抽出された list_array のデータに加え，org ファイルが編集され新しく追記された pdf のリンクや変更された pdf のリンクが格納される．

3.3 check_link

check_link では，csv ファイルの状態を常に最新のものとして維持するために，以下のような変更されたタグのデータと，

```
./docs/happy_ruby/RussOlsen_EloquentRuby_c1.pdf,new_link
```

README.org に新しく追記された pdf のリンク./docs/happy_ruby/add_link.pdf,false を diff_array という配列に格納することを目的とする．

org ファイルから直接格納された tmp_array と csv ファイルから格納された list_array の diff を取り add_link という配列に格納する．つまり，

```
新しく追加された pdf のリンク,false
```

のみ add_link に格納される．

図 3.3のように，add_link のデータ

```
新しく追加された pdf のリンク,false
新しく追加された pdf のリンク,false
新しく追加された pdf のリンク,false
```

に加え，csv ファイルで変更されたデータ

```
pdf のリンク,new_link1
pdf のリンク,new_link2
pdf のリンク,new_link3
```

を diff_array に格納する.

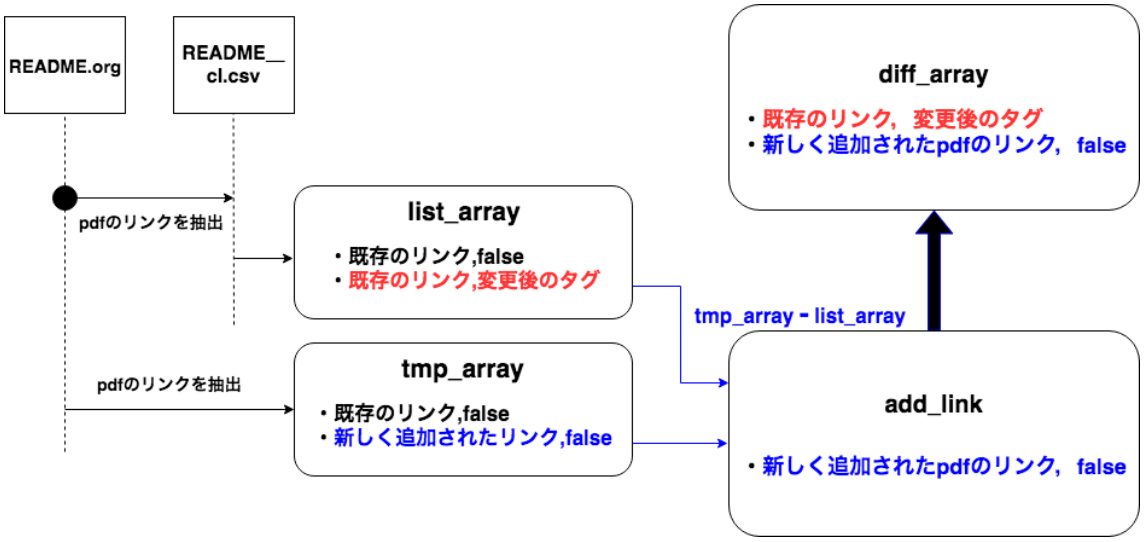

図 3.3: add_link と diff_array 作成までの流れ.

3.4 mk_revised_csv

mk_revised_csv では，citation list として作成された csv ファイルを，常に新しい状態として保つため，add_link のデータを csv ファイルに反映させることを目的とする．こうすることで，次回の実行時，再び同じリンクが add_link や diff_array に格納されることを防ぐ．また，リンク切れのチェックを行う際，最新の citation list とディレクトリ情報を比較するため，citation list の更新が必要である．

csv ファイルは現在，変更が加えられていない．ここでは，図 3.4のように，現在存在する csv ファイルのデータと add_link に格納されたデータを比較し，add_link に格納されたデータを csv ファイルに上書きするようにし，README_cl.csv は常に最新のデータを保持した状態を維持する．こうすることで，次回 org ファイルから pdf のリンクを抽出し，csv ファイルと比較する際，org ファイルに変更がなければデータが一致するようになる．

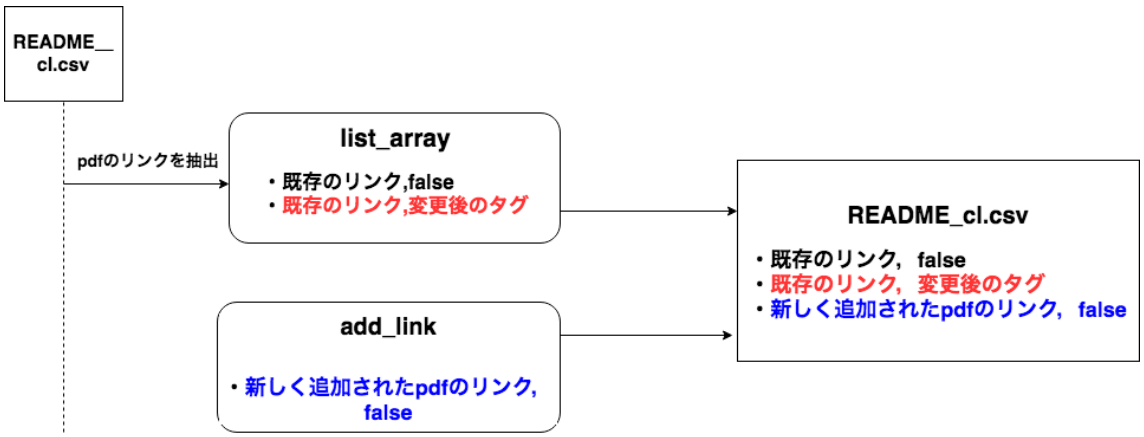

図 3.4: mk_revised_csv の仕組み.

3.5 check_broken_link

check_broken_link では，文書作成の環境としての課題であるリンク切れのチェックを解決するため，図 3.5のようにリンク切れのチェックを行う．

まず org ファイルが存在するディレクトリの情報を取得し，dir に格納する．次に，csv ファイルを呼び出し各データの「pdf のリンク」のみを抽出し，pdf ファイルのファイル名が dir に格納されているのかをチェックする．

もし格納されていなければリンク切れを起こしていることをターミナル上で図 3.6のように赤で表示するようにする．

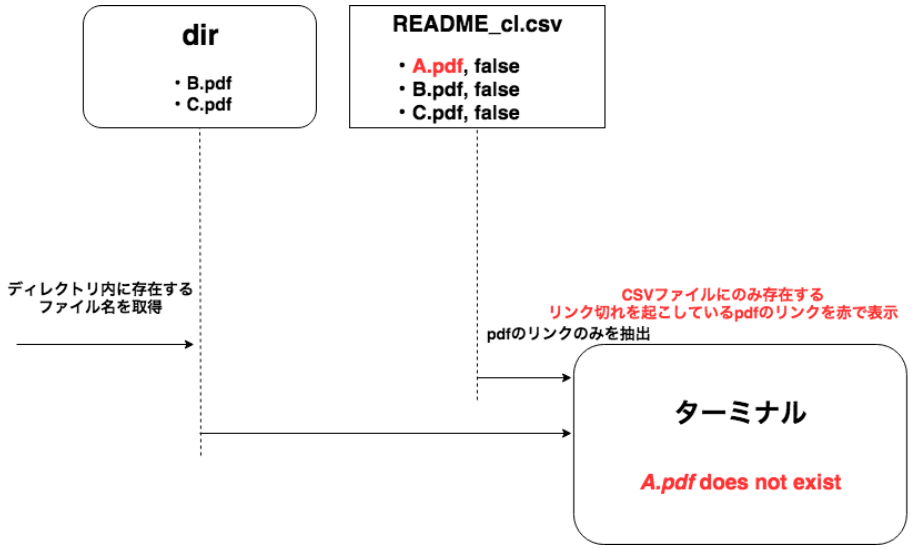

図 3.5: check_broken_link の仕組み．

```
c2_2_orthogonality.pdf does not exist
c3_14_plain_text.pdf does not exist
c3_15_shell.pdf does not exist
c3_16_editor.pdf does not exist
c3_17_backup.pdf does not exist
RussOlsen_EloquentRuby_c10_class.pdf does not exist
c5_29_view.pdf does not exist
RussOlsen_c2_2_Class.pdf does not exist
RussOlsen_Eloquent_c9.pdf does not exist
HeadFirst_SoftwareDevelop_Appenndix1.pdf does not exist
RefactoringRuby_CodeSmell.pdf does not exist
Refactoring2-1.pdf does not exist
Refactoring2-2.pdf does not exist
Refactoring2-3.pdf does not exist
DesignPatternMatsu-Cover-4.pdf does not exist
DesignPatternGangOfFour.pdf does not exist
```

図 3.6: リンク切れを知らせる例．

3.6 tree

Qiita で紹介されている tree [8] を参考に作成した．

README.org のような多数のリンクが存在する場合，どのファイルがどの層に存在するのかを瞬時に把握するためにディレクトリ構造をターミナル上で表示する．

また，デフォルトで README.org を赤で表示するようにしている．

3.7 rev_line

rev_line では，README.org を一行ずつ line という配列に格納する．このとき README.org と diff_array のデータを比較しタグが変更されていれば，pdf のリンクは格納せず，タグのみを line に格納することを目的とする．こうすることで，新しい org ファイルに書き出す際，pdf のリンクではなく，タグが表示されるようになる．

org ファイルを一行ずつ読み込み,line に格納する．このとき，org ファイルに存在する pdf のリンクと diff_array の pdf のリンクを照合する．リンクが diff_array に存在し，org ファイルには存在しない場合新しく追加されたリンクであるため，そのリンクを line に格納する．

また，図 3.7のように，diff_array に格納されたデータの中で，タグが「false」でない場合，pdf のリンクをそのまま表示せず，タグに書かれた名前で表示するよう，タグのみを line に格納する．

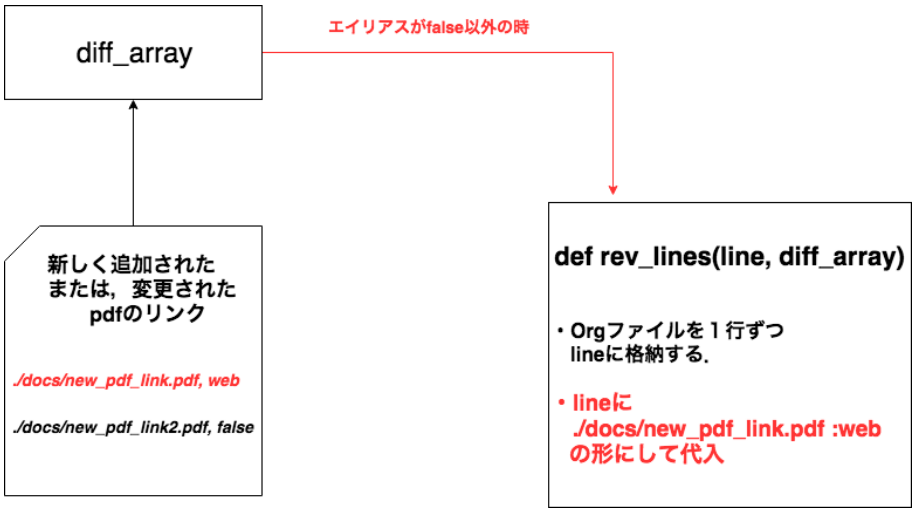

図 3.7: diff_array のデータを新しい org ファイルに反映させる．

3.8 mk_html

org ファイルは，文書作成において非常に便利であるが，ブラウザー上で文書を見る際は，HTML の方が見やすい．そのため，mk_html では org ファイルを html ファイルに変換し，html のテーマを変更することでより見やすい文書とすることを目的とする．

まず初めに，mk_htmlを実行するにあたりRubyGemsよりQiitaで好評のorg_html_themes [7] をインストールしておく．

新しく作成されたorgファイルにhead_code.txtを書き加える．書き加えられた行の図3.8の部分を自分が保存しているorg_html_themesのパスに書き換える．

その後保存し，orgファイルを閉じるとorgファイルとは別にhtmlファイルが作成される．

```
#+SETUPFILE: /Users/hiroto/org-html-themes/setup/theme-readtheorg.setup
```

図 3.8: 書き換える行の例.

第4章　考察

4.1　リンク切れ

大規模な文書作成においてOrg-modeを採用すると，見出しや，子見出し毎に内容を非表示にすることができるため，編集したい見出し毎に，文章の確認を行うことができ，スムーズに文書作成を行うことができる．

本研究では，176行からなるorgファイルをサンプルとして，データの抽出を行なった．その結果，35個のpdfのリンクを自動で抽出した．このように，orgファイルをメモやレポートの原稿として利用すると，日々内容が増え，手作業でのリンク維持に困難が生じる．そのため，orgファイルの規模が大きくなればなるほど，ornbによってリンクリストの表示だけでも提供される意義が深まる．

現在，文書作成はshell上で行うことが多いためEmacs上で文書作成を行う上で，リンクの変換とリンク切れのチェックをterminal上に表示することによって，ユーザが扱い易く，見てわかる便利な機能を提供している．

実際に授業で使用された，orgファイルからpdfのリンクを抽出すると，下記のようにcsvファイルとして保存される．

```
./docs/emacs/emacs_key_bind3.pdf,false
./docs/emacs/FolderConfig.pdf,false
                    (中略)
./docs/happy_ruby/TanoshiiRuby_v5_c1.pdf,false
./docs/happy_ruby/TanoshiiRuby_v5_c2-3.pdf,false
./docs/happy_ruby/RussOlsen_EloquentRuby_c5.pdf,false
./docs/happy_ruby/RussOlsen_EloquentRuby_c1.pdf,false
```

また，以下のように, csvファイルの一番下にあるデータのタグをnew_linkに変更することで，

```
./docs/emacs/emacs_key_bind3.pdf,false
./docs/emacs/FolderConfig.pdf,false
                    (中略)
./docs/happy_ruby/TanoshiiRuby_v5_c1.pdf,false
./docs/happy_ruby/TanoshiiRuby_v5_c2-3.pdf,false
./docs/happy_ruby/RussOlsen_EloquentRuby_c5.pdf,false
./docs/happy_ruby/RussOlsen_EloquentRuby_c1.pdf,new_link
```

図 4.1に記載されている一番下のリンクが図 4.2のように"new_link"という表示に変わっているのが確認できた．また，最終行以外では，タグがデフォルトの"false"である場合は，変更されていないのがわかる．

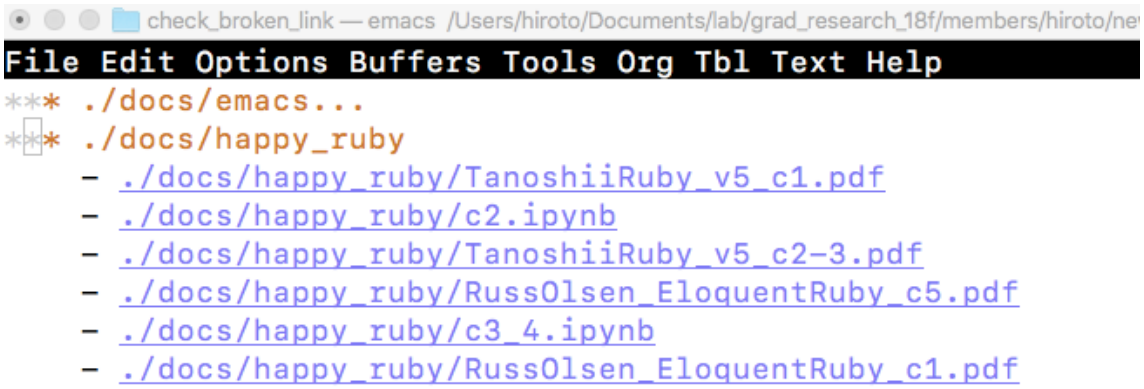

図 4.1: リンク名変更前の org ファイル.

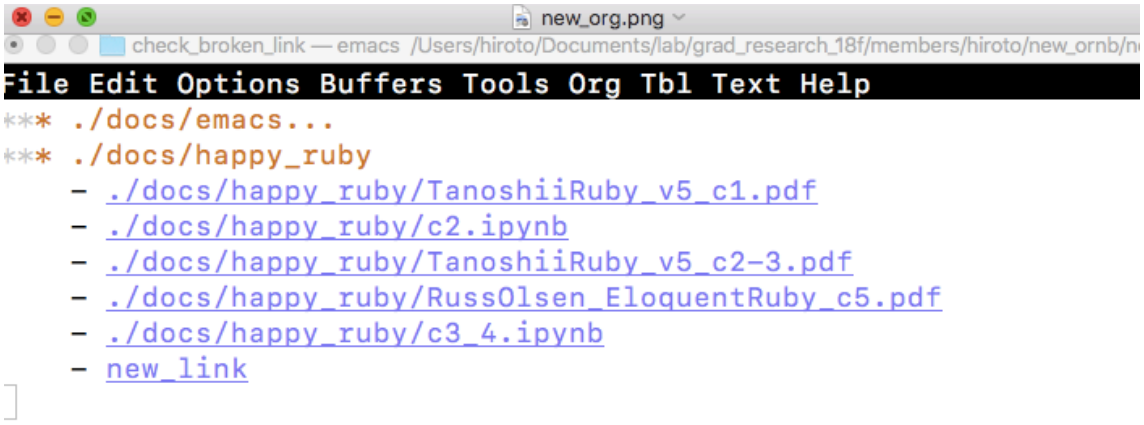

図 4.2: 最下部のリンク名を"new_link"に変更後の org ファイル.

4.2 tree 表示

tree 機能のディレクトリ表示を図 4.3に示した．

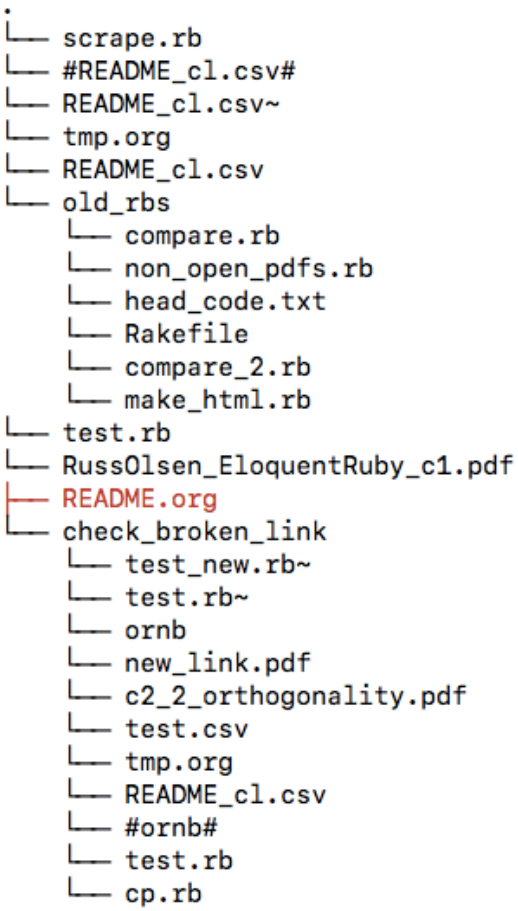

図 4.3: tree でのディレクトリ表示.

tree を採用することで，ディレクトリ構造を terminal 上で確認することが可能になったため，リンク切れチェックの際, 視認性が上がった.

また，tree の表示において，colorize ライブラリを適用することで，ディレクトリ毎に色を付け加えたり，ファイルの種別によって色を変えるなどが可能となった.

4.3 今後の課題

一方で，機能として足りない, 気づいた点が存在した.

一つは，タグの種類を増やし更なる場合分けを行うことで，URL を表示する場合，別名で表示する場合など表示の選択肢を増やすことができれば，さらに利便性が上がる.

さらに，現在は pdf のリンクのみであるため，jpeg ファイルや png ファイルなどの，複数のファイルを一度に抽出し，各種類毎に citation list を作成することでより効率が上がる.

また，リンク切れを防ぐためにファイルシステムを考えなおす必要があると感じた．リンク切れを知らせるのみではなく，リンク切れを起こらないファイルシステムを開発することで，より快適に文書作成を行うことができるためである.

また，開発が遅くなったため研究室のメンバーや教授からの実際に使用したフィードバックは得られていない．そのため，これからフィードバックをしてもらい，機能を追加していく必要がある．

また，Org-mode を使用する研究室のメンバーや教授から，卒業論文の作成時 pdf への変換を行うことや，文書の構造において，見出しなどの階層構造にもう少し手を加えれたいという声を聞いたため，ornb にこれらの機能も追加し，より広範な文書作成の手助けが可能となるように改善する必要性を感じた．

第5章　総括

本研究では，より効率よく，理解度の高い文書作成を助ける補助ツールの開発を進めた．いかに開発内容を簡潔に示す．

1. pdf のリンクを公開してはいけないとき，参照の付け替えを行うことができる．
2. pdf のリンク切れの有無のチェックを行うことができる．
3. ディレクトリ構造を tree 構造でターミナル上に表示することができる．
4. html への自動変換を行うことができる．

今後の展望としては，更新された org ファイルを pdf ファイルとして変換することを可能にし，今後の研究室メンバー等卒業論文を書く学生のスムーズな pdf 変換を助ける機能を追加することでより文書作成の幅が広がると考える．

また，リンク切れの原因の 1 つである，ファイルの保存場所の違いを擬似的なセミラティス構造を持つファイルシステムを開発することで直交性を保ち，リンク切れを防ぐことを可能にしていく．

謝辞

本研究を行うにあたり，終始多大なる御指導，御鞭撻をいただいた西谷滋人教授に対し，深く御礼申し上げます．また，本研究の進行に伴い，様々な助力，知識の供給を頂きました西谷研究室の同輩，先輩方に心から感謝の意を示します．本当にありがとうございました．

参考文献

[1] ノートブックの基本 - Wolfram Language Documentation, https://reference.wolfram.com/language/guide/NotebookBasics.html.ja (accessed on 11 Feb 2019).

[2] Markdown のメリット・デメリット・注意点・感想 - 本やらなんやらの感想置き場, https://spaceplace.hatenablog.jp/entry/markdown-pros-and-cons (accessed on 11 Feb 2019).

[3] Kent Beck (著), 和田 卓人 (翻訳), テスト駆動開発, (オーム社, 2017/10/14).

[4] Org mode for Emacs: あなたの生活をプレーンテキストで, https://orgmode.org/ja/ (accessed on 10 Feb 2019).

[5] Org Mode マニュアル - orgmode.jp, http://orgmode.jp/doc-ja/org-ja.html (accessed on 20 Feb 2019).

[6] ライブラリ - Ruby, https://www.ruby-lang.org/ja/libraries/ (accessed on 10 Feb 2019).

[7] GitHub - fniessen/org-html-themes: How to export Org mode files into awesome HTML in 2 minutes, https://github.com/fniessen/org-html-themes (accessed on 28 Sep 2019).

[8] Ruby で Linux コマンドの再実装 (tree 編) - Qiita, https://qiita.com/agatan/items/4c50554ae22aa4181cc1 (accessed on 25 Jan 2019)