卒業論文

ユーザメモと wiki を連携するシステムの開発

関西学院大学　理工学部　情報科学科

3550 江本 沙紀

2017 年 3 月

指導教員　　西谷　滋人　教授

目次

1　開発の背景

近年，ナレッジマネジメントが企業経営の重要な要素と言われ，導入する企業が増えている．ナレッジマネジメントとは，個人の持つ知識や情報を組織全体で共有し，有効に活用することで業績を上げようという経営手法である．日本語では，「知識管理」などと訳され，「KM」と略されることもある [1].

ナレッジマネジメントでは，グループ開発において共有する知識は暗黙知と形式知に分けられる [2]．暗黙知は主に口伝によって一対一でつたえられたり，あるいは体で覚えるというのが一般的である．しかし，定着するまでの間は一般的にメモという形で個人的な知識として扱われるのが普通である．一方で図書館や web などでは文書や hyper text として誰もが読める形で保管，提供される知識は形式知と呼ばれる．

形式知と暗黙知

形式知	暗黙知
言葉や文字で表現できる 記録として残されている	言葉や文字で表現できない 無意識や筋肉記憶の中に存在
多くの人に伝えるときに効果的	人と人のコミュニケーションによってのみ伝達可能
(例)料理本，教科書	(例)包丁の使い方，火加減

図 1　暗黙知と形式知

暗黙知の形式知化はいくつも行われており，google 検索したときによく見る Qiita.com などもそれらをまとめるサイトを提供している．西谷研究室では，各所属学生の暗黙知を形式化するために，my_help という gem を開発し，自分のためのメモを残して活用している．本研究では，西谷研究室内でのナレッジマネジメントを推進するため，メモソフト my_help から wiki clone の hiki へ自動変換するシステムの開発と，my_help をよりよい

ソフトにするために FrontPage の設計をする．

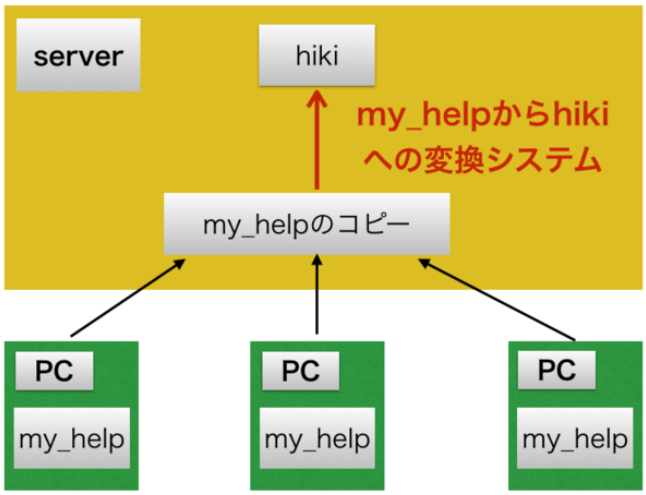

図 2

本論文の構成は次の通りである．対象とするソフトはあまり一般に普及しているものではないため，最初にソフトの特徴と簡単な振る舞いを紹介する．次に３章では作成したソフトの振る舞いとコードの中身を詳述する．４章では今後これらの変換ソフトを使って web を作成するためにプロトタイプを作成し，そこで必要とされる機能について検討を行っている．最終章では結論をまとめた．

2　関連するソフトの振る舞い

本章では，対象とするソフトはあまり一般に普及しているものではないため，最初にソフト (my_help および hiki) の特徴と簡単な振る舞いを紹介する．

2.1　my_help

my_help は西谷研究室で使用しているユーザ独自のメモを作成する gem である．

gem は正式名称を RubyGems といい，Ruby 用のライブラリを使う時に必要となるソフトウェアのこと [3]．パッケージ管理ツール gem があることで，Ruby 用ライブラリのインストール，アンインストール，バージョン管理などを簡単に行うことができる．プログラミング言語 Ruby のファイルに付属されていて，無料で利用することができる．

gem の利点は次の通りである。

- 標準化された構造があるので，初めてみた人でも分かるようになっている．
- gem があることで，簡単に Ruby 用ライブラリをインストールでき，初心者でもアプリ機能を装備できる．
- 誰でも作成，配布が可能である．

my_help が提供するコマンドとその振る舞いを help 表示から示す．

```
Usage: my_help [options]
    -v, --version                    show program Version.
    -l, --list                       list specific helps
    -e, --edit NAME                  edit NAME help(eg test_help)
    -i, --init NAME                  initialize NAME help(eg test_help).
    -m, --make                       make executables for all helps.
    -c, --clean                      clean up exe dir.
        --install_local              install local after edit helps
        --delete NAME                delete NAME help
        --hiki                       my_help2hiki
```

my_help では help の要素をこの help 表示と同じ構造，項目と対応する記述で表示される．それぞれの項目はまとまりをつくり，それを-l でリスト表示される．–version およ

び–list は gem 標準に準拠するために用意されている．

my_help を研究室内で利用する利点は次の通りである．

- 研究室内でのメモの書き方が統一できる．
- どこにメモをしたか忘れることがない．
- 普段研究の為に使うターミナルから離れること無くメモを残すことができるので，書きたいときにすぐに書くことができる．

2.2 hiki

Ruby で書かれた高機能・高速 Wiki クローン [4]．CGI(Commond Gateway Interface) を利用して，Web サーバと連動して動く [5]．西谷研究室では，hiki の形式を利用してサイトを作り，研究室内での情報共有や gem の使い方などを掲載して閲覧できるようにしている．また，卒業論文の作成にも hiki の形式で作成している．

3　方法

本研究で開発した my_help から hiki への自動変換ソフト my_help2hiki は my_help の gem に組み込んでいる．本章では，my_help2hiki のコマンドとコマンドによる振る舞いを記述する．

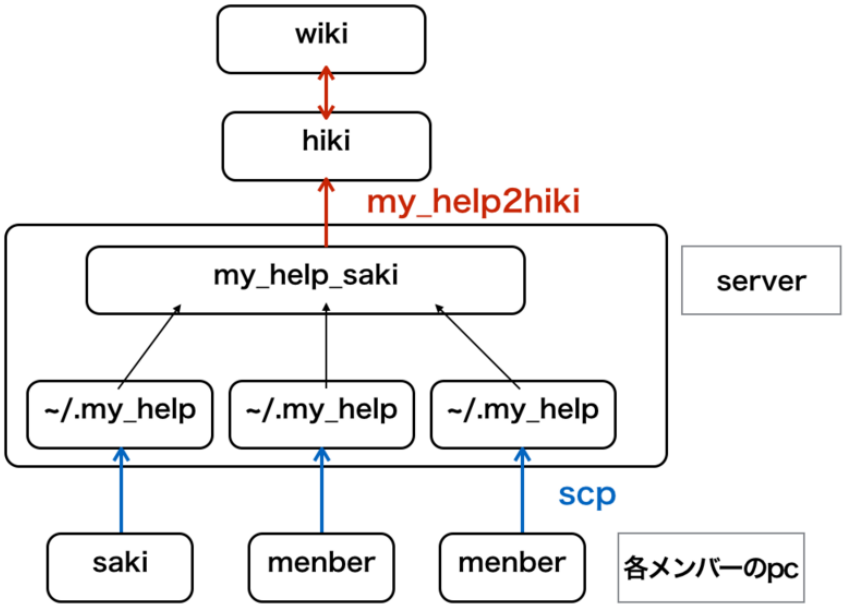

図 3　my_help2hiki

my　my_help2hiki は図のようになっている，各学生が my_help で作ったメモは.my_help のディレクトリに保存される．これを scp コマンドを利用して server の my_help_saki のディレクトリにコピーする．コピーした my_help の内容を my_help2hiki を利用して hiki に変換し，wiki で表示する．

3.1　my_help/lib/specific_help.rb

specific_help.rb は以下のようになっている

```
/Users/saki/my_help/lib% cat specific_help.rb
# -*- coding: utf-8 -*-
```

```
require "optparse"
require "yaml"
require "my_help/version"
require 'fileutils'
require "coderay"

module SpecificHelp
  class Command

    def self.run(file,argv=[])
      new(file, argv).execute
    end

    def initialize(file,argv=[])
      @source_file = file
      @help_cont = YAML.load(File.read(file))
      @help_cont[:head].each{|line| print line.chomp+"\n" } if @help_cont[:hea
      @help_cont[:license].each{|line| print "#{line.chomp}\n" } if @help_cont
      @argv = argv
    end

    def execute
      if @argv.size==0
        if @source_file.include?('todo')
          @argv << '--all'
        else
          @argv << '--help'
        end
      end
      command_parser = OptionParser.new do |opt|
        opt.on('-v', '--version','show program Version.') { |v|
          opt.version = MyHelp::VERSION
          puts opt.ver
```

```
    }
    @help_cont.each_pair{|key,val|
      next if key==:head or key==:license
      opts = val[:opts]
      opt.on(opts[:short],opts[:long],opts[:desc]) {disp_help(key)}
    }
    opt.on('--edit','edit help contents'){edit_help}
    opt.on('--to_hiki','convert to hikidoc format'){to_hiki}
    opt.on('--all','display all helps'){all_help}
    opt.on('--store [item]','store [item] in backfile'){|item| store(item)
    opt.on('--push', 'push my_todo on remote host'){push}
    opt.on('--remove [item]','remove [item] and store in backfile'){|item
    opt.on('--add [item]','add new [item]'){|item| add(item) }
    opt.on('--backup_list [val]','show last [val] backup list'){|val| back
  end
  begin
    command_parser.parse!(@argv)
  rescue=> eval
    p eval
  end
  exit
end

def backup_list(val)
  val = val || 10
  print "\n ...showing last #{val} stored items in backup.\n"
  backup=mk_backup_file(@source_file)
  File.open(backup,'r'){|file|
    backup_cont = YAML.load(File.read(backup))
    backup_size = backup_cont.size
    backup_size = backup_size>val.to_i ? val.to_i : backup_size
    backup_keys = backup_cont.keys
    backup_keys.reverse.each_with_index{|item,i|
```

```
      break if i>=backup_size
      line = item.to_s.split('_')
      printf("%10s : %8s%8s\n",line[0],line[1],line[2])
    }
  }
end

def mk_backup_file(file)
  path = File.dirname(file)
  base = File.basename(file)
  backup= File.join(path,"."+base)
  FileUtils.touch(backup) unless File.exists?(backup)
  return backup
end

def store(item)
  if item==nil
    print "spcify --store [item].\n"
    exit
  else
    print "Trying to store #{item}\n"
  end
  backup=mk_backup_file(@source_file)
  unless store_item = @help_cont[item.to_sym] then
    print "No #{item} in this help.  The items are following...\n"
    keys = @help_cont.keys
    keys.each{|key|
      p key
    }
    exit
  end
  p store_name = item+"_"+Time.now.strftime("%Y%m%d_%H%M%S")
  backup_cont=YAML.load(File.read(backup)) || {}
```

```
  backup_cont[store_name.to_sym]=store_item
  File.open(backup,'w'){|file| file.print(YAML.dump(backup_cont))}
end


def push
  p "push my_todo"
  data_dir = File.join(ENV['HOME'],'.my_help')
  FileUtils.cd(data_dir)
  system "pwd"
  system "rm -rf ~/.my_help/*.yml~"
  system "scp -r ~/.my_help saki@nishitani0:~"
  system "ssh saki@nishitani0 ls ~/.my_help"


end


def add(item='new_item')
  print "Trying to add #{item}\n"
  new_item={:opts=>{:short=>'-'+item[0], :long=>'--'+item, :desc=>item},
      :title=>item, :cont=> [item]}
  @help_cont[item.to_sym]=new_item
  File.open(@source_file,'w'){|file| file.print YAML.dump(@help_cont)}

def remove(item)
  print "Trying to remove #{item}\n"
  store(item)
  @help_cont.delete(item.to_sym)
  File.open(@source_file,'w'){|file| file.print YAML.dump(@help_cont)}
end

def edit_help
```

```
  system("emacs #{@source_file}")
end

def to_hiki
  @help_cont.each_pair{|key,val|
    if key==:head or key==:license
      hiki_disp(val)
    else
      hiki_help(key)
    end
  }
end

def all_help
  @help_cont.each_pair{|key,val|
    if key==:head or key==:license
      val[0]+=":"
      disp(val)
    else
      disp_help(key)
    end
  }
end

def hiki_help(key_word)
  items =@help_cont[key_word]
  puts "\n!!"+items[:title]+"\n"
  hiki_disp(items[:cont])
end

def hiki_disp(lines)
  lines.each{|line| puts "*#{line}"}  if lines != nil
end
```

```
    def disp_help(key_word)
      print_separater
      items =@help_cont[key_word]
      puts CodeRay.scan("-#{items[:title]}:", :Taskpaper).term
      disp(items[:cont])
      print_separater
    end

    def disp(lines)
      lines.each{|line| puts CodeRay.scan("+#{line}", :Taskpaper).term}
    end

    def print_separater
      print "---\n"
    end

  end
end
```

3.2 my_help/lib/my_help.rb

my_help.rb は以下のようになっている.

```
/Users/saki/my_help/lib% cat my_help.rb
# -*- coding: utf-8 -*-
require "optparse"
require "yaml"
require "fileutils"
require "my_help/version"
require "systemu"

module MyHelp
```

```
class Command

  def self.run(argv=[])
    new(argv).execute
  end

  def initialize(argv=[])
    @argv = argv
    @default_help_dir = File.expand_path("../../lib/daddygongon", __FILE__)
    @local_help_dir = File.join(ENV['HOME'],'.my_help')
    set_help_dir_if_not_exists
  end

  def set_help_dir_if_not_exists
    return if File::exists?(@local_help_dir)
    FileUtils.mkdir_p(@local_help_dir, :verbose=>true)
    Dir.entries(@default_help_dir).each{|file|
      next if file=='template_help.yml'
      file_path=File.join(@local_help_dir,file)
      next if File::exists?(file_path)
      FileUtils.cp((File.join(@default_help_dir,file)),@local_help_dir,:verb
    }
  end

  def execute
    @argv << '--help' if @argv.size==0
    command_parser = OptionParser.new do |opt|
      opt.on('-v', '--version','show program Version.') { |v|
        opt.version = MyHelp::VERSION
        puts opt.ver
      }
      opt.on('-l', '--list', 'list specific helps'){list_helps}
      opt.on('-e NAME', '--edit NAME', 'edit NAME help(eg test_help)'){|file
```

```
    opt.on('-i NAME', '--init NAME', 'initialize NAME help(eg test_help).
    opt.on('-m', '--make', 'make executables for all helps.'){make_help}
    opt.on('-c', '--clean', 'clean up exe dir.'){clean_exe}
    opt.on('--install_local','install local after edit helps'){install_loc
    opt.on('--delete NAME','delete NAME help'){|file| delete_help(file)}
    opt.on('--hiki','my_help2hiki'){hiki}
  end
  begin
    command_parser.parse!(@argv)
  rescue=> eval
    p eval
  end
  exit
end

def delete_help(file)
  del_files=[]
  del_files << File.join(@local_help_dir,file)
   exe_dir=File.join(File.expand_path('../..',@default_help_dir),'exe')
   del_files << File.join(exe_dir,file)
  p del_files << File.join(exe_dir,short_name(file))
  print "Are you sure to delete these files?[yes]"
  if gets.chomp=='yes' then
    del_files.each{|file| FileUtils.rm(file,:verbose=>true)}
  end
end

USER_INST_DIR="USER INSTALLATION DIRECTORY:"
INST_DIR="INSTALLATION DIRECTORY:"
def install_local
  Dir.chdir(File.expand_path('../..',@default_help_dir))
  p pwd_dir = Dir.pwd
  status, stdout, stderr = systemu "gem env|grep '#{USER_INST_DIR}'"
```

```
  if stdout==""
    status, stdout, stderr = systemu "gem env|grep '#{INST_DIR}'"
  end
  p system_inst_dir = stdout.split(': ')[1].chomp
  if pwd_dir == system_inst_dir
    puts "Download my_help from github, and using bundle for edit helps\n"
    puts "Read README in detail.\n"
    exit
  end
  system "git add -A"
  system "git commit -m 'update exe dirs'"
  system "Rake install:local"
end

def short_name(file)
  file_name=file.split('_')
  return file_name[0][0]+"_"+file_name[1][0]
end

def make_help
  local_help_entries.each{|file|
    exe_cont="#!/usr/bin/env ruby\nrequire 'specific_help'\n"
    exe_cont << "help_file = File.join(ENV['HOME'],'.my_help','#{file}')\n
    exe_cont << "SpecificHelp::Command.run(help_file, ARGV)\n"
    file = File.basename(file,'.yml')
    [file, short_name(file)].each{|name|
      p target=File.join('exe',name)
      File.open(target,'w'){|file| file.print exe_cont}
      FileUtils.chmod('a+x', target, :verbose => true)
    }
  }
  install_local
end
```

```
def clean_exe
  local_help_entries.each{|file|
    next if ['emacs_help','e_h','my_help','my_todo'].include?(file)
    file = File.basename(file,'.yml')
    [file, short_name(file)].each{|name|
      p target=File.join('exe',name)
      FileUtils::Verbose.rm(target)
    }
  }
end

def init_help(file)
  p target_help=File.join(@local_help_dir,file+'.yml')
  if File::exists?(target_help)
    puts "File exists. rm it first to initialize it."
    exit
  end
  p template = File.join(@default_help_dir,'template_help.yml')
  FileUtils::Verbose.cp(template,target_help)
end

def edit_help(file)
  p target_help=File.join(@local_help_dir,file)
  system "emacs #{target_help}.yml"
end

def local_help_entries
  entries= []
  Dir.entries(@local_help_dir).each{|file|
    next unless file.include?('_')
    next if file[0]=='#' or file[-1]=='~' or file[0]=='.'
    entries << file
```

```
      }
      return entries
    end

    def list_helps
      print "Specific help file:\n"
      local_help_entries.each{|file|
        file_path=File.join(@local_help_dir,file)
        file = File.basename(file,'.yml')
        begin
          help = YAML.load(File.read(file_path))
        rescue=> eval
          p eval
          print "\n YAML load error in #{file}."
          print "  See the line shown above and revise by\n"
          print "  emacs #{file_path}\n"
          exit
        end
        print "  #{file}\t:#{help[:head][0]}\n"
      }
    end

    def hiki
      system "ls ~/.my_help2"
      system "emacs_help --to_hiki > ~/Sites/hiki-1.0/data/text/emacs_help_sak
      system "my_todo --to_hiki > ~/Sites/hiki-1.0/data/text/my_todo_saki"
      system "ssh_help --to_hiki > ~/Sites/hiki-1.0/data/text/ssh_help_saki"
      system "open -a safari 'http://localhost/~saki/hiki-1.0/?FrontPage'"
    end

  end
end
```

3.3 使用法，コマンド

- TARGET –push：作成したメモ (TARGET) をサーバに送る
- my_help –hiki：作成したメモを hiki 形式に変換し，wiki で表示できるようにする．

3.4 コマンドの振る舞い

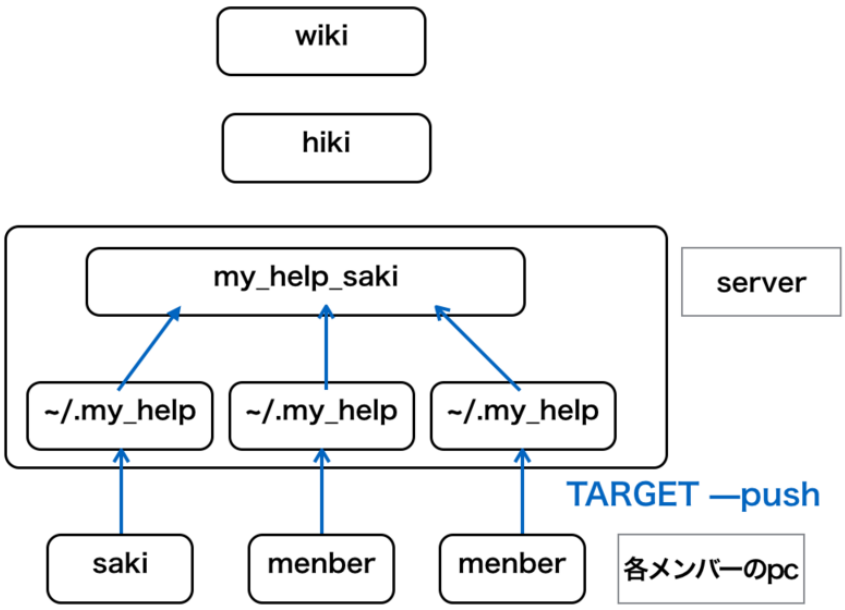

図 4　TARGET –push

■TARGET –push

```
def push
  p "push my_todo"
  data_dir = File.join(ENV['HOME'],'.my_help')
  FileUtils.cd(data_dir)
  system "pwd"
  system "rm -rf ~/.my_help/*.yml~"
  system "scp -r ~/.my_help saki@nishitani0:~"
```

```
      system "ssh saki@nishitani0 ls ~/.my_help"
```

- 3,4 行目

  my_help では，作成したメモが.my_help のディレクトリに自動的に追加されるので，ディレクトリを.my_help に移動する．

- 6 行目

  .my_help にメモが追加されるとき，yaml 形式のファイルで保存される．メモを更新すると，一つ前に保存したファイルは*.yml というファイル名でバックアップとして残される．**rm -rf** で不必要なファイルは削除し，サーバにコピーするときのデータ量を減らしている．

- 7 行目

  scp -r /[directory 名] [server 名] server に ssh 接続を行い, directory を server にコピーする．-r はディレクトリ全体をコピーすることを示している．西谷研究室で利用している nishitani0 というサーバにコピーしている．

- 8 行目

  nishitani0 に ssh 接続し.my_help の中身を書き出して，コピーができているかコマンドを実行した時に確認が行えるようにしている．

■my_help –hiki

```
def hiki
      p 'my_help2hiki'
      system "emacs_help --to_hiki > ~/Sites/hiki-1.0/data/text/emacs_help_sa
      system "my_todo --to_hiki > ~/Sites/hiki-1.0/data/text/my_todo_saki"
      system "ssh_help --to_hiki > ~/Sites/hiki-1.0/data/text/ssh_help_saki"
      system "open -a safari 'http://localhost/~saki/hiki-1.0/?FrontPage'"
    end
```

- 2-4 行目

  my_help には，**TARGET –to_hiki** というコマンドがあり，これによって yaml

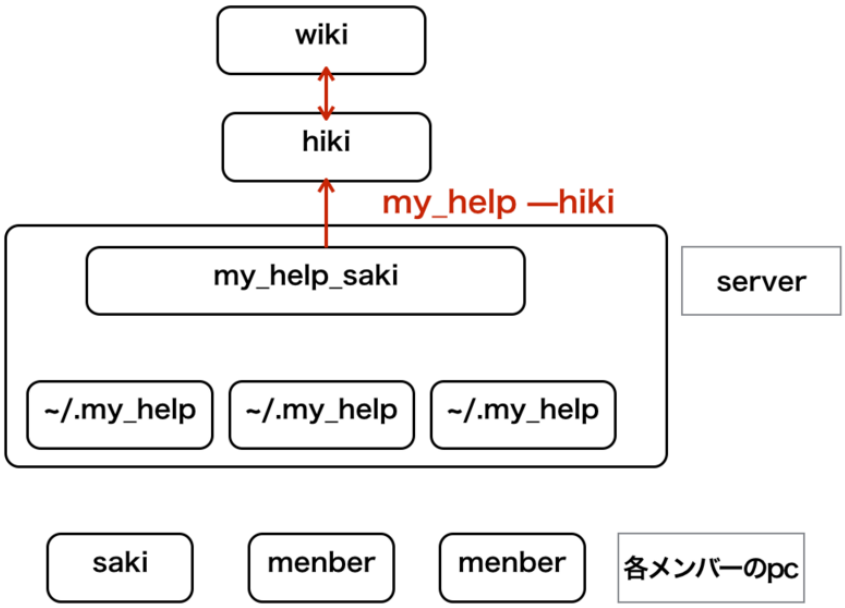

図 5　my_help2hiki

形式で保存されているメモを hiki 形式で書き出すことができる．この –hiki のコマンドを使って hiki 形式にしたものを，wiki で表示することのできるフォルダである **/Sites/hiki-1.0/data/text/**に入れることで，wiki での表示を可能にしている．emacs_help，my_todo，ssh_help は全て私の my_help に入っているメモ．

- 5 行目

  wiki のページ図 6 に示した FrontPage を表示するコマンド．これによりメモが更新されているのを即時確認することができる．FrontPage は以下のようになっている．

```
!saki’s help
*[[ssh_help_saki]]
*[[my_todo_saki]]
*[[emacs_help_saki]]
```

  先頭に**!**をつけることで 1 行目の saki’s help を見出しにし，2 4 行目は*****によって箇条書き，角括弧でリンクになっている．

図 6　コマンドを実行したときに開く FrontPage

4　FrontPage の設計

my_help をよりよくするための設計を示す．各学生の FrontPage，各 help のページに分けて，実装すると便利になる理由と共に記述している．

図 7　FrontPage

■1. メモの内容による分類

研究室内の所属学生の利用するハードウェア，ソフトウェアは同じものが多い．例えば，西谷研究室ではハードウェアは全員が mac を使い，プログラム作成はターミナル，hiki 文書の作成には mi というソフトを使用している．それぞれのハードウェア，ソフトウェアに関する help を分類分けしておくことで，調べたいことに関しての help を探しやすくすることができる．

■2. 研究室内のメンバーの help へのリンク

同研究室の他学生の FrontPage へのリンクを作る．同回のメンバーが書いたメモを見たい，先輩のメモを見たい，他メンバーの研究を知りたい．そのときの目的に

応じて閲覧する help を選ぶことができる．

■3. 閲覧回数が上位の help へのリンク

閲覧回数の多い，研究室内の学生が見た回数の多いものへのリンクを作る．研究室に入ってばかりで分からないことが多いときにこの help を見れば，研究室のことが理解できる．知らなくても支障はないが，知っておくと便利な豆知識を得ることが期待される．

■4. 更新の新しい help へのリンク

更新が新しいものを表示しておくことで，メンバーが得た最新の知識を得やすくなる．また，他メンバーがどのような研究を進めているか，どのようなことを調べてたのかを知ることができ，自分の研究の進め方の参考にすることができる．

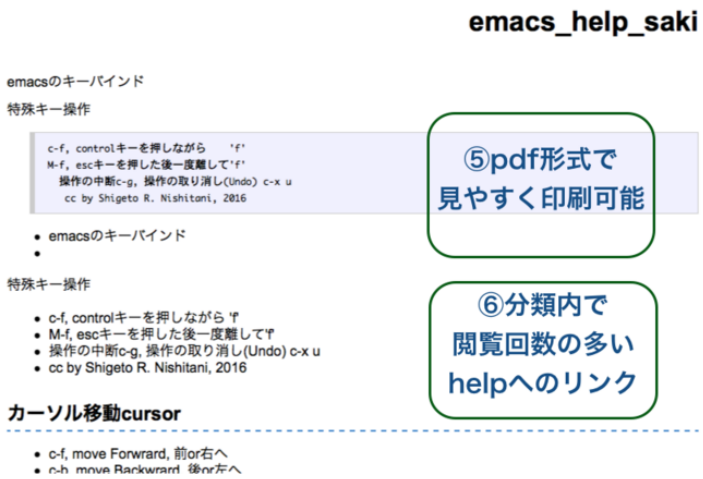

図 8　例 emacs_help

■5.pdf 形式で見やすく印刷可能

紙媒体で持ち歩くことを可能にするために，pdf 形式に変換することで見やすく

した help を印刷することができる．emacs_help を例にして考える．この help はターミナルで emacs を使うときのキーバインドを表している．ターミナルを利用していると，この help を開きながら操作することは手間がかかる．そこで見やすくした help を紙に印刷することでこの手間を省くことができる．西谷研究室では hiki から latex への変換ソフト，hiki2latex があるので，作成可能だと考えられる．

■6. 分類内で閲覧回数の多い help へのリンク

分類内で閲覧回数の多い help は他メンバーの多くが得た知識なので，知っておくべき知識であるといえる．自分がその分類内で分からないことを解決する手がかりになることが期待される

5 結果

```
/Users/saki% my_help --list
Specific help file:
  emacs_help :emacs のキーバインド
  memo_help :ヘルプのサンプル雛形
  my_todo :my_todo
  ssh_help :ssh の help
```

これは私の my_help の中身を書き出している．下が emacs_help の中身である．

```
/Users/saki% emacs_help --all
emacs のキーバインド

特殊キー操作
  c-f, control キーを押しながら     'f'
  M-f, esc キーを押した後一度離して'f'
    操作の中断 c-g，操作の取り消し (Undo) c-x u
     cc by Shigeto R. Nishitani, 2016
+emacs のキーバインド:
+
特殊キー操作
+  c-f, control キーを押しながら     'f'
+  M-f, esc キーを押した後一度離して'f'
+    操作の中断 c-g，操作の取り消し (Undo) c-x u
+     cc by Shigeto R. Nishitani, 2016:
---
-カーソル移動 cursor:
+c-f, move Forwrard,    前 or 右へ
+c-b, move Backwrard,   後 or 左へ
+c-a, go Ahead of line, 行頭へ
+c-e, go End of line,   行末へ
+c-n, move Next line,   次行へ
```

+c-p, move Previous line, **前行へ**

-**ページ移動** page:

+c-v, move Vertical, **次のページへ**

+M-v, move reversive Vertical, **前のページへ**

+c-l, centerise Line, **現在行を中心に**

+M-<, move Top of file, **ファイルの先頭へ**

+M->, move Bottom of file, **ファイルの最後尾へ**

-**ファイル操作** file:

+c-x c-f, Find file, **ファイルを開く**

+c-x c-s, Save file, **ファイルを保存**

+c-x c-w, Write file NAME, **ファイルを別名で書き込む**

-**編集操作** edit:

+c-d, Delete char, **一字削除**

+c-k, Kill line, **一行抹消，カット**

+c-y, Yank, **ペースト**

+c-w, Kill region, **領域抹消，カット**

+ **領域選択は，先頭** or **最後尾で** c-space **した後，最後尾** or **先頭へカーソル移動**

+c-s, forward incremental Search WORD, **前へ** WORD **を検索**

+c-r, Reverse incremental search WORD, **後へ** WORD **を検索**

+M-x query-replace WORD1 <ret> WORD2**：対話的置換**（y or n **で可否選択**）

-**ウィンドウ操作** window:

+c-x 2, 2 windows, **二つに分割**

+c-x 1, 1 windows, **一つに戻す**

+c-x 3, 3rd window sep, **縦線分割**

+c-x o, Other windows, **次の画面へ移動**

-**バッファー操作** buffer(**すでに** open **して** emacs **にバッファーされた** file):

+c-x b, show Buffer, **バッファのリスト**

+c-x c-b, next Buffer, **次のバッファへ移動**

-**終了操作** quit:

+c-x c-c, Quit emacs, **ファイルを保存して終了**

+c-z, suspend emacs, **一時停止，**fg **で復活**

このように各学生のメモを何種類も作成できるが，自分のパソコンでしか見ることができなかった．本研究による my_help2hiki を使うことで，図 6 のように my_help を wiki で表示可能になり，各学生のメモを研究室内で共有することができるようになる．さらに 4 章で述べたような設計ができれば，研究室内のナレッジマネジメントは今より推進させると考えている．

6　今後の課題

西谷研究室には内部サイトがあり，研究室内で使うシステムのマニュアルなどが公開されている．hiki 形式への変換ができれば wiki で表示することはできるようになるが，my_help2hiki のコマンドで wiki のページを作成しても，現段階では研究室内全員が見ることはできない．内部サイトへの自動表示を可能にするようなコマンドを作る必要があると考えられる．今後，作成した my_help の設計に基づき実装を進めてほしい．

参考文献

[1]「e-Words ナレッジマネジメント」,http://e-words.jp/w/ナレッジマネジメント.html,2017/1/27 アクセス.

[2] ニック・ミルトン,「プロジェクト・ナレッジ・マネジメント」(生産性出版，東京都渋谷区渋谷，2009 年発行)，pp.4-5.

[3]「Ruby on Rails 初心者必見！パッケージ管理ツール『gem』を徹底解説」,https://blog.codecamp.jp/rails-gem,2017/1/27 アクセス.

[4]「Hiki -Front Page-」,http://hikiwiki.org/ja/,2017/1/27 アクセス.

[5]「Hiki -Wikipedia」,https://ja.wikipedia.org/wiki/Hiki,2017/1/27 アクセス.