vi vim

　vim は vi improved の略とされ、vim の改良版である。現在では、vi で開くと、vim が開く。
　今ではたいていの人はパソコンはウインドウズのような GUI から始める。画面のメニューバーやアイコンを見ながらマウスをクリックすることでパソコンに 何かの操作をさせる。メニューバーには文字でコピーとか貼り付けとか書いてあるから、ここを選択すればコピーできるのだななどと誰にでもわかる。アイコン もその操作をイメージする絵になっている。
今、文書の最後の行全体をコピーして、それを次の行に貼り付ける操作をするとする。ウインドウズでは次のような操作になるだろう。
マウスでスクロールして、一番最後の行を出す。
↓
マウスでドラッグして、最後の行全体を選択する。
↓
マウスでメニューバーのコピーをクリックする。
↓
画面をクリックする。
↓
Enter キーを押す。
↓
マウスでメニューバーの貼り付けをクリックする。
　少しパソコンに慣れてくると、いちいちメニューバーからコピーや貼り付けを選択するより、Ctrl キーを押しながら C キーを押してコピーし、Ctrl キーを押しながら V キーを押すほうが速いと気づく。また文書の一番最後に移動するのは、スクロールするよりも、Ctrl キーを押しながら End キーを押す ほうが速い。このキー操作ですｋると次のようになる。
Ctrl キーを押しながら End キーを押す。
↓
マウスでドラッグして、最後の行全体を選択する。
↓
Ctrl キーを押しながら C キーを押す。
↓
画面をクリックする。
↓
Enter キーを押す。
↓
Ctrl キーを押しながら V キーを押す。
ウインドウズのテキストエディターではこれが最速だろうか。これを vi で操作すると、次のようになる。
G キーを押す。
↓

y キーを 2 回押す。
↓
p キーを押す。
つまり Gyya と入力すればすんでしまう。これは速い。
　vi の操作はこのように速いため、またすべての操作がキーボードでできるから GUI が使えない環境でも使えるから、今でも vi の愛好者は絶えない。今の vi は Shift キーと Ctrl キーを押しながら V キーを押すことで他の文書の文を貼り付けることもでき、ウインドウズのテキストエディターのようにも使 える。ウインドウズのテキストエディターには、メニューバーにたくさんの項目が並んでいる。これをすべてキーボードに割り当てようというのだから、かなり の数になることが想像できる。これを覚えなければならないことになる。ウインドウズのアプリケーションソフトのワードなどは、バージョンが変わるとメ ニューバーの配置がかなり変わってしまうことが多い。今まで覚えていたのと違うからとまどったり、使い方がわからなかったりする。ようやく使い方に慣れた かと思うと、また新しいバージョンが出る。このことが繰り返される。しかしキーボードに割り当てたコマンドはそう簡単に変えれない。これはキーボードの配置を変えるようなものである。キーボードの配置を変えるようなことをすれば、世界中からブーイングが起こる。今までのブラインドタッチができなくなるから である。キーボードに割り当てているコマンドはたくさんあるのだけれど、一度覚えてしまえば、少なくとも私たちが生きている間は使えることになる。以下に 気づいたことを書く。
vi を開くには、端末に vi と入力すればよい。Shiken.txt というファイルを開くには
vi Shiken.txt
と入力すればよい。vi の後のスペースは一つでなくてもよい。指定したファイル名がない時は新しいファイルをつくることになる。
複数のファイルを開く時は、
vi ファイル名 1 ファイル名 2 ファイル名 3
のようにスペースをあけて書いていく。

- A　append の略。入力モードになり、キーボードで打った文字がカーソルのある行の最後に入力されていく。
- a　append の略。入力モードになり、キーボードで打った文字がコマンドモードの時カーソルがあった文字の右側に入るようになる。add だから付け加える感じになるのである。
- ar　argument(引数)の略。複数のファイルを開いている時、開いているファイル名を下に表示する。現在編集しているファイルは[]で囲んである。
- B　back の略。カーソルをスペースを区切りとして、前の単語の先頭へ移す。カーソルのある単語から 3 番めの単語にカーソルを移すには、3B とやればよい。
  $sth = $dsh->prepare($sql);
  で最後にカーソルがあるとすると、B を打っていくと、; $ = $
  と移動する。
- b　back の略。カーソルを前の単語の先頭へ移す。記号も単語とみなされる。カーソルのある単語から 3 番めの単語にカーソルを移すには、3b とやればよい。
  $sth = $dsh->prepare($sql);

で最後にカーソルがあるとすると、b を打っていくと、;)s(p-d$=s$
と移動する。

- C　c は change の略。カーソルのある行が削除され、新しい行を入力できるようになる。cc と同じだが、Shift キーを使うからこちらが少し遅い。S とも同じである。
- c　c は change の略。c を入力しただけでは、何も変わらないが、その後 Enter キーを押すと、カーソルのある行が削除され、新しい行を入力できる ようになる。cc と同じである。c3、あるいは 3c と打って、Enter キーを押すと、カーソルのある行を含んで 4 行が削除され、カーソルがあった所から 4 行目の所に入力できるようにカーソルが移動する。
- caw　caw は change a word の略。カーソルのある単語全体が削除され、新しい文字を入力できるようになる。cw は、カーソルのある単語全体が削除されず、単語の中でカーソルのある文字より右が削除される。cW と入力しても cw と同じである。
- cb　change back の略。一つの単語のカーソルのある文字から、左の文字が削除され、新しい文字を入力できるようになる。
- cc　c は change の略。カーソルのある行が削除され、新しい行を入力できるようになる。C と同じだが、Shift キーを使わないからこちらが少し速い。S とも同じである。
- Ctrl-B　b は backwards の略。現在画面に表示されている上の部分が表示される。
- Ctrl-D　d は downward の略。3 を入力し、Ctrl キーを押した状態で d キーを打つと、画面が 3 行分下に行く。数字を入力しないと、画面が半分下に行く。
- Ctrl-E　e は extra lines の略。1 行下にスクロールする。よって現在カーソルのある所は上に行く。これを使うのは現在カーソルのある所の下を見たいのだが、カーソル自体 は動かしたくない場合である。下にさらに行を出したいので、extra lines の E を使っていると考えるべきである。
- Ctrl-F　f は forward の略。現在画面に表示されている下の部分が表示される。
- Ctrl-G　g は get の略。開いているファイル名とその行数が表示され、ファイルを変更して保存していないなら、[変更あり]と表示される。
- Ctrl-L　L は clear の略。下に表示されるシステムメッセージが消える。
- Ctrl-N　入力モードでいくらか単語を入力した後これを押すことで、単語の候補が出る。Ctrl-N をさらに押すと、下の候補が選択されていく。Ctrl-P と の違いは、Ctrl-N は一番よく使う候補が一番上に表示され、その次の候補がその下に表示されていくが、Ctrl-P は一番よく使う候補が一番下に表示 され、その次の候補がその上に表示されていく。入力ミスを防ぐ効果がある。
- Ctrl-P　入力モードでいくらか単語を入力した後これを押すことで、単語の候補が出る。Ctrl-P をさらに押すと、上の候補が選択されていく。Ctrl-N と の違いは、Ctrl-P は一番よく使う候補が一番下に表示され、その次の候補がその上に表示されていくが、Ctrl-N は一番よく使う候補が一番上に表示 され、その次の候補がその下に表示されていく。入力ミスを防ぐ効果がある。
- Ctrl-R　r は redo の略。u でもとに戻したが、もとに戻しすぎた時、戻しすぎを戻す。u の反対の働きをする。
- Ctrl-U　u は upwards の略。3 を入力し、Ctrl キーを押した状態で u キーを打つと、画面

が 3 行分上に行く。数字を入力しないと、画面が半分上に行く。

- Ctrl-V　V は visual の略。ビジュアルブロックモードになる。Ctrl-V を押した時にカーソルがあった点とカーソルを右に動かした所の点を結んだ線を対角線とする長方形の中にあるものを選択する。
- Ctrl-W h　W は window の略 二つのウインドウの左のウインドウにフォーカスを移す。
- Ctrl-W j　W は window の略 二つのウインドウの下のウインドウにフォーカスを移す。
- Ctrl-W k　W は window の略 二つのウインドウの上のウインドウにフォーカスを移す。
- Ctrl-W l　W は window の略 二つのウインドウの右のウインドウにフォーカスを移す。
- Ctrl-W q　W は window の略 q は quit の略。現在カーソルのあるウインドを閉じる。
- Ctrl-W r　W は window の略 左右、上下のウインドウを入れ替える。
- Ctrl-W v　w は window の略。v は vertical の略。Ctrl-W を押した後に、v を打って現在開いている文書を縦に２つに分けて表示することができ る。これは Ctrl-W Ctrl-V と V キーを打つときに Ctrl キーを押したましても同じことになる。今のパソコンは横長で上下の画面が少ないから、下にあるものを見ようとして、下に カーソルを移動すると、上が見れなくなる。上と下の両方を見ながら作業したい時この機能は便利である。一方の画面で変更して保存すると、その保存はもう一 つの画面にも反映される。下にあるものを上にコピーしたい時は、下を表示させた画面でコピーして、上を表示させた画面で貼り付ければよいのである。一つの 画面から他の画面に移りたい時は、Ctrl-W を押した後に w を打つ。
- Ctrl-W w　w は window の略。分割してある画面で、一つの画面から他の画面に移りたい時に使う。Ctrl-W を押した後に w を打つ。
- Ctrl-Y　画面を 1 行上にスクロールする。よって現在カーソルのある所は下に行く。実際に使う場合大事なのは画面がどう動くかでなく現在カーソルのある所が上に 行くか下に行くかである。Y は下に向いている矢印のイメージである。Ctrl-Y はカーソルのある所を下に進める時に使うのだと考えるべきである。
- Ctrl-[　コマンド画面にもどる。Esc キーを打つのと同じである。Esc キーはしばしば半角/全角キーを間違えて打つことがあり、コマンドモードにもどるに は、こちらを使った方が無難である。プログラムは一から書く散文でなく、問題なく動くとわかっている部品を組み合わせ、その一部を変更してつくるものであ る。一から書くとミスが出やすい。だからプログラムを書く基本は部品の読み込みとコピーと一部の変更である。これはみなコマンドモードでできるものであ る。それで vi は基本的にコマンドモードで使うものである。入力モードはやむを得ない場合に使うものである。あまり好ましくない入力モードを終えたらすぐ に Ctrl-[でコマンドモードに戻るという癖をつけておくべきである。
- cw　change word の略。cw を入力すると、一つの単語のカーソルのある文字から、右の文字が削除され、新しい文字を入力できるようになる。
- c$　c は change の略。c$を入力すると、一行の中で、カーソルのある文字から、右の文字がすべて削除され、新しい文字を入力できるようになる。
- D　delete の略。カーソル位置からカーソルのある行の最後まですべて削除する。
- d　delete の略。d を打って次に移動を表すキーを打つことで、移動した所にあるものを

削除する。dw なら、w はカーソルを次の単語の先頭へ移動する から、今カーソルがある文字から次の単語までが削除される。gg で最初の行の先頭に移動し、d を打ち、G を打つなら、今表示されているものはすべて削除さ れる。

- daw　delete a word の略。今カーソルがある単語が削除される。diw との違いはカーソルがスペースの所にある時、daw はスペースとその右の単語を削除するが、diw はスペースのみを削除する。
- dd　delete の略。カーソルのある行をすべて削除する。d$としても同じことができるが、キー操作は明らかに dd のほうが速い。カーソルのある行から 4 行削除したいなら、4dd とやればよい。
- dG　d は delete の略。カーソルのある行から最後まですべて削除する。
- dgg　d は delete の略。カーソルのある行から最初まですべて削除する。
- dggdG　d は delete の略。すべて削除する。
- diw　delete innera word の略。今カーソルがある単語が削除される。daw との違いはカーソルがスペースの所にある時、daw はスペースとその右の単語を削除するが、diw はスペースのみを削除する。
- dl　delete の略。カーソルがある 1 文字を削除する。カーソルがある文字を含んで左に 3 文字削除したいなら 3dl とする。x と同じである。x が 1 文字の入力ですむから少し速くなる。
- dw　delete word の略。今カーソルがある文字から次の単語までを削除する。
- d0　d は delete の略。0 でカーソルが、その行の最初まで移動するから、カーソルのある文字からその行の最初まで削除でる。
- d$　d は delete の略。$でカーソルが、その行の最後まで移動するから、カーソルのある文字からその行の最後まで削除でる。
- E　end の略。カーソルをスペースを区切りとして、次の単語の最後へ移す。カーソルのある単語から 3 番めの単語にカーソルを移すには、3E とやればよい。
  $sth = $dsh->prepare($sql);
  で先頭にカーソルがあるとすると、E を打っていくと、h = ;
  と移動する。
- e　end の略。カーソルを次の単語の最後へ移す。記号も単語とみなされる。カーソルのある単語から 3 番めの単語にカーソルを移すには、3e とやればよい。
  $sth = $dsh->prepare($sql);
  で先頭にカーソルがあるとすると、e を打っていくと、$ h = $ h > e $ l ;
  と移動する。
- ea　e は end の略。a は append の略。カーソルがある単語の最後から入力する入力モードになる。
- Esc　コマンドモードに切り替える。Ctrl-[を打つのと同じである。Esc キーを打とうとして、半角/全角キーを間違えて打つことがよくあり、コマンド モードにもどるには、Ctrl-[を使った方が無難である。ビジュアルモードからコマンドモードにもどる時にも使う。ビジュアルモードの状態で v を打って もコマンドモードにもどるからこちらが簡単で

ある。

- F　find の略。カーソルがある行のカーソルの左にある F の次に入力する文字にカーソルが移る。例えば Fs と入力すると、カーソルがある行のカーソルの 左にある最初の s にカーソルが移る。その次の s にカーソルを移したければ;を入力する。前の s にもどりたければ,を入力する。行が長い時のカーソル移動に 便利である。
- f　find の略。カーソルがある行のカーソルの右にある f の次に入力する文字にカーソルが移る。例えば fs と入力すると、カーソルがある行のカーソルの 右にある最初の s にカーソルが移る。その次の s にカーソルを移したければ;を入力する。前の s にもどりたければ,を入力する。行が長い時のカーソル移動に 便利である。
- G　Go の略。カーソルを最終行の行頭へ移動する。G の前に数字をつけて、例えば 10G とすると、10 行目の行頭に移動する。最初の行に移動したければ、1G とすればよいのである。
- ga　get ascii の略。カーソルがある行のアスキーコードが 10 進法、16 進法、8 進法で表示される。
- gE　go end の略。カーソルをスペースを区切りとして、前の単語の最後へ移す。カーソルのある単語から 3 番めの単語にカーソルを移すには、3gE とやればよい。
  $sth = $dsh->prepare($sql);
  で最後にカーソルがあるとすると、gE を打っていくと、; = h
  と移動する。
- ge　go end の略。カーソルを次の単語の最後へ移す。記号も単語とみなされる。カーソルのある単語から 3 番めの単語にカーソルを移すには、3ge とやればよい。
  $sth = $dsh->prepare($sql);
  で最後にカーソルがあるとすると、ge を打っていくと、; l $ e > h $ = h $
  と移動する。
- gg　go の略。カーソルを最初の行の行頭に移動する。1G と同じだがこちらが入力が容易。

- ggdG　すべて削除する。
- gU　g は get の略 U は uppercase の略。この後に移動のコマンドを入力すると、移動した所の小文字が大文字に変わる。例えば、現在カーソルが行頭にあるとすると、gU$と入力することで、カーソルのある行の小文字がすべて大文字に変わる。
- gUU　g は get の略 U は uppercase の略。現在カーソルのある行の小文字がすべて大文字に変わる。
- gu　g は get の略 u は uppercase の略、uppercase は大文字の意味だが、u を小文字にすることで、小文字という感じにしている。この後に 移動のコマンドを入力すると、移動した所の大文字が小文字に変わる。例えば、現在カーソルが行頭にあるとすると、gu$と入力することで、カーソルのある 行の大文字がすべて小文字に変わる。
- guu　g は get の略 u は uppercase の略。現在カーソルのある行の大文字がすべて小文字に変わる。uppercase は大文字の意味だが、u を小文字にすることで、小文字という感じにしている。

- g~　g は get の略。この後に移動のコマンドを入力すると、移動した所の大文字が小文字に変わり、小文字が大文字に変わる。例えば、現在カーソルが行頭に あるとすると、g~$と入力することで、カーソルのある行の大文字がすべて小文字に変わり、小文字がすべて大文字に変わる。
- H　home の略。カーソルを画面の最初の行の行頭に移動する。カーソルを左に移動する時は h を使う。上に最大限移動する時は、その大文字の H を使うのだと考えてもよい。
- h　カーソルを左へ 1 文字移動する。キーボードのホームポジションは右の人差し指を j キーに置く。h キーを打つには右の人差し指を左に動かすから、カーソ ルが左に行くという感覚と合う。5 文字左へ移動したければ 5h とすればよい。ビジュアルモードで行の一部を選択する時は、h の前に数字をつけてカーソルを 移動させる方法が特に有用である。
- I　insert の略。入力モードになり、キーボードで打った文字がカーソルのある行の最初に入力されていく。
- i　insert の略。入力モードになり、キーボードで打った文字がコマンドモードの時カーソルがあった文字の左側に入るようになる。insert だから割り込む感じになるのである。
- J　join の略。行と行をくっつけて一つの行にする。カーソルより下 4 行を 1 行にしたいなら、4J とする。
- j　カーソルを下へ 1 行移動する。3 行下へ移動するには、3j とやればよい。
- k　カーソルを上へ 1 行移動する。5 行上へ移動するには、5k とやればよい。
- L　last の略。カーソルを画面の最後の行の行頭に移動する。カーソルを左に移動する時は l を使う。下に最大限移動する時は、その大文字の L を使うのだと考えてもよい。
- l　カーソルを右へ 1 文字移動する。キーボードのホームポジションは右の薬指を l キーに置く。l キーは h,j,k キーの右にあり、カーソルが右に行くとい う感覚と合う。3 文字右へ移動したければ 3l とすればよい。日本語入力がされている時、1 行が長くなる。l を繰り返し打って右に行くと、カーソルの移動に 時間がかかる。その場合にこの l の前に数字を入力して、移動するのは有用である。その他の方法としては w を使うと少し速くなる。f を使うのも速いが、これ は半角/全角キーを打って日本語を一語入力する必要がある。半角文字が混じる日本語で半角文字にカーソルを移動する場合は f を使うのが一番速い。ビジュア ルモードで行の一部を選択する時は、l の前に数字をつけてカーソルを移動させる方法が特に有用である。
- M　middle の略。カーソルを画面の中央の行の行頭に移動する。
- m　mark の略。現在カーソルがある行をマークする。例えば現在
  @file=<FILE>;
  にカーソルがあるとする。mf と入力すると、これに f のマークがつく。カーソルを他の行に動かして、'f と入力すると、カーソルが@file=< FILE>;の先頭の@に移る。このマークはこのファイルに保存され、一度ファイルを閉じて、後にまた編集するために開いた時も'f と入力してカー ソルをを@file=<FILE>;の先頭に移すことができる。何にどんなマークをつけたのか忘れてしまうことがある。その時は:marks でマークの一覧が表示される。マークが画面に表示しきれない時は、Enter キーで次のマークが表示され、一番下のマークまで表示されると Enter キー でもとの画面にもどる。

- N　next の略。/や?で検索をした時に、前の検索結果に進むのに使う。
- n　next の略。/や?で検索をした時に、次の検索結果に進むのに使う。
- O　open の略。カーソルのある行の上に新しい行を追加して、入力モードになる。
- o　open の略。カーソルのある行の下に新しい行を追加して、入力モードになる。
- P　put あるいは paste の略。d で削除したり、y でコピーしたものを貼り付ける。一行を削除したり、コピーした時はカーソルのある行の上に貼り付ける。行の一部を削除したり、コピーしたものはカーソルのある文字の左に貼り付ける。
- p　put あるいは paste の略。d で削除したり、y でコピーしたものを貼り付ける。一行を削除したり、コピーした時はカーソルのある行の下に貼り付ける。行の一部を削除したり、コピーしたものはカーソルのある文字の右に貼り付ける。
- q　入力したものを保存しておく機能である。次のような手順になる。まずカーソルを空白行に置く。空白行がなければ o で空白行をつくる。次に q を入力し て、その後にアルファベット 1 字あるいは数字 1 文字を入力する。i あるいは a で入力モードにする。記録したいものを打ち込む。入力が終われば Ctrl- [でコマンドモードにもどる。q を打つ。これで入力したものが、q の後に入力した文字や数字に保存された。次にこれを使いたければ、コマンドモードで@の 後に、その記録した文字または数字を入力するとよい。具体的言うと次のようになる。
  まず空白行の最初にカーソルを移す。
  qc
  と入力する。
  i
  と入力して入力モードにする。
  これは試験です。
  と入力する。
  Ctrl-[
  でコマンドモードにもどす。
  q
  を入力する。 これで c レジスタに「これは試験です。」が保存された。
  これを使うには、挿入したい所にカーソルを移し、
  @c
  と入力する。 「これは試験です。」
  が挿入される。
  @c と入力すれば、何回でもこの文が挿入される。
  入力モードでの入力はキーボードを使わずに、Ctrl-C で他の文書からコピーしてきて、
  Shift-Ctrl-V で貼り付けることもできることもできる。
  何をレジスタに保存したが忘れてしまった時は
  :reg
  で保存したものが表示される。
  保存したものは、一度ファイルを閉じて、パソコンの電源を切っても保存されていて、次

にパソコンを立ち上げ、そのファイルを編集する時にも使える。vi で別のファイルを開いても使える。

@文字(あるいは数字)は、カーソルのある文字の左に挿入される。だから行末で挿入しようとすると、最後の文字の左に挿入され、行の最後に挿入前の文字が 1 字残ることになる。これは行の一部に使うのでなく、行単位のものを記録しておいて、使うほうがよさそうである。

これを使うと便利なのは、q: で以前に実行したコマンドを表示したときである。q: を実行すると、一番最後の行にカーソルが行くが、ここに新しいコマンドを入力できる。長いコマンドは入力が手間なので、レジスタに保存しておく。例えば置 換は:%s///cg と入力するものが多い。s にこれをレジスタとして記憶する。すると、@s と入力することで、%s///cg が出る。なお q: の後、入力する時には、: はいらないから、レジスタには: のない形で保存する。

- q:　q は quote の略か。以前に実行したコマンドを表示できる。これを閉じるには、:q と入力する。表示されたコマンドで再度実行したいコマンドにカー ソルを移し、Enter キーを打つことで、そのコマンドを実行できる。そのコマンドの編集もできる。この機能が役に立つのは、コマンドを入力してそれにエ ラーが出た時である。例えば taro を hanako に置き換えようとして、

  :%s/taro/hanako/gc

  と入力しようとしたが、打ち間違えて

  :%s/taro/hanako/gx

  と入力してしまった。画面には、「余分な文字が後ろにあります」というエラーが出る。q:の機能がないと、この誤ったコマンドを表示させることができな い。q:を使うと、この誤ったコマンドが画面に出るから、誤りが確認でき、単に x を c に変えるだけで、Enter キーを打って実行できる。コマンドをもう 一度最初から書くのに比べると速くなる。
- q/　q は quote の略か。以前に実行した検索履歴を表示できる。これを閉じるには、:q と入力する。検索したい項目にカーソルを移し、Ctrl-C で Commad line にその検索項目を送ることができる。Command Line には自動的に/が入力されているから、Enter キーを打つことでその項目を検索できる。
- q?　q は quote の略か。以前に実行した検索履歴を表示できる。これを閉じるには、:q と入力する。検索したい項目にカーソルを移し、Ctrl-C で Commad line にその検索項目を送ることができる。Command Line には自動的に?が入力されているから、Enter キーを打つことでその項目を検索できる。
- R　replace の略。カーソルのある所の文字が、R の次に打つ文字に変わり、カーソルが自動的に次の文字に移る。文字を入力するとカーソルのある所の文字がその文字に変わる。これを繰り返す。置換モードから抜けるには、Esc キーを押す。
- r　replace の略。カーソルのある所の文字が、r の次に打つ文字に変わる。s との違いは s は入力モードに変わるが、r はコマンドモードのまま変えることである。少しの文字の変更ならコマンドモードのまましたほうが簡単である。
- reg　replace の略。レジスタに保存したものを表示する。これを閉じるには、Esc キーを

2 回押す。

- S　substitute の略。カーソルのある所の 1 行の文字が消去され、そこに新しい文字を入力できる入力モードになる。cc C と同じである。5S とすればカーソルのある行を含んで下に 5 行が消去され新しい文字を入力できる入力モードになる。
- s　substitute の略。カーソルのある所の文字が消去され、そこに新しい文字を入力できる入力モードになる。3s とすればカーソルのある文字を含 んで右に 3 文字が消去され新しい文字を入力できる入力モードになる。r との違いは r はコマンドモードのままで文字を変えるが、s は入力モードになる。
- u　undo の略。変更をもとにもどす。
- V　visual の略である。ビジュアルモードになる。v との違いは V は行単位で選択することである。h l キーは使えず、j k キーで選択する。
- v　visual の略である。ビジュアルモードになり、hjkl のキーを打つことでカーソルを移動させ、マウスでドラッグするように画面の一部を選択できる。y を打つこと で選択したものがコピーでき、p を打つことで貼り付けることができる。d を打つことで削除できる。英語は単語と単語の間にスペースがあるから、y 数字 W で 行の一部をコピーできる。しかし日本語の場合は単語の間にスペースがなく、この方法を用いることができない。v を用いると行の一部を選択することができ、 行の一部をコピーできる。日本語で書かれた所を処理するのに便利な機能である。もとのコマンドモードにもどすには、再度 v を打つ。
- vaB　visual a block の略。カーソルを{または}に移動して vaB を入力すると{}とともに{}の中にあるものも選択する。Perl や PHP では、{}で囲むことが多い。{}の中身を選択してくれるのは、ありがたい機能である。
- vab　visual a block の略。カーソルを(または)に移動して vab を入力すると()とともに()の中にあるものも選択する。
- vaw　visual a word の略。vaw を入力した時にカーソルがある単語を選択する。
- viB　visual inner block の略。カーソルを{または}に移動して viB を入力すると{}の中にあるものを選択する。Perl や PHP では、{}で囲むことが多い。{}の中身を選択してくれるのは、ありがたい機能である。
- vib　visual inner block の略。カーソルを(または)に移動して vib を入力すると()の中にあるものを選択する。
- W　word の略。カーソルをスペースを区切りとして、次の単語の先頭へ移す。カーソルのある単語から 3 番めの単語にカーソルを移すには、3W とやればよい。
  $sth = $dsh->prepare($sql);
  で先頭にカーソルがあるとすると、W を打っていくと、$ = $
  と移動する。
- w　word の略。カーソルを次の単語の先頭へ移す。記号も単語とみなされる。カーソルのある単語から 3 番めの単語にカーソルを移すには、3w とやればよい。
  $sth = $dsh->prepare($sql);
  で先頭にカーソルがあるとすると、w を打っていくと、$ s = $ d - p (s)
  と移動する。

- X　カーソルのある左の文字を削除する。削除することを英語で cross out と言い、cross だから x のイメージになる。ウインドを閉じる時も x をクリックするから、x が何かをなくす時に使うのはすでになじみがある。3X とすると、カーソルのある文字を含まずに左に 3 文字削除する。
- x　カーソルのある文字を削除する。削除することを英語で cross out と言い、cross だから x のイメージになる。ウインドを閉じる時も x をクリックするから、x が何かをなくす時に使うのはすでになじみがある。 DEL キーを打っても同じことだが、DEL キーをブラインドタッチで打つのは難しいが、x ならブラインドタッチで打つことができる。3x とすると、カーソ ルのある文字を含んで右に 3 文字削除する。dl でも同じことができる。x が 1 文字の入力ですむから少し速くなる。ただし、d$でカーソルのある文字からそ の行の最後まで削除でき、d0 でカーソルのある文字からその行の最初まで削除できるが、x$ x0 ではできない。
- xp　x はカーソルのある文字を削除する。その文字はバッファに保存される。削除したからカーソルは次の文字に移る。ここで p を打つとカーソルの右に削除し た文字が貼り付けられる。結果的にカーソルのある文字とその右の文字を入れ替えたことになる。3xp とすると、カーソルのある文字を含んで右に 3 文字削除 し、4 文字目の文字の右にこの 3 つの文字を貼り付ける。
- Y　yank の略。カーソルのある 1 行をコピーし、バッファに保存する。yy と同じである。Y は大文字だから Shift キーを使わなければならない。それで yy のほうが少し速い。
- y　yank の略。y の後に移動を示す文字を入れることで、その移動した所にある文字がコピーされる。例えば、gg でカーソルを最初の行の先頭に移動した 後、y を打ち、G を打つなら、全文がコピーされる。移動キーで適当な所に移動し、p キーを打つなら全文を貼り付けることができる。y15l とすれば、カー ソルのある所から右に 15 文字コピーできるのだが、15 字がどこまでになるかは、数えなければならないことになる。これは面倒である。v でビジュアルモー ドにすれば、15l とすれば、どこまで選択したかがちょうどマウスでドラッグした時のように表示される。その後 y を打てばコピーできる。
- yW　yank word の略。y でコピーし、W でカーソルが次の単語まで移動するから、カーソルのある単語を一つコピーできる。W はスペースを区切りとして、一つの単語と考えてカーソルが移動する。例えば、
  taro, hanako, yoko, takeshi, hiroko
  とあり、y3W なら、taro, hanako, yoko がコピーされる。これは 3yW としてもよい。
  y3w なら、,も 1 文字と考えるから、taro, hanako がコピーされる。
- yw　yank word の略。y でコピーし、w でカーソルが次の単語まで移動するから、カーソルのある単語を一つコピーできる。w は記号も一つの単語と考えてカーソルが移動する。例えば、
  taro, hanako, yoko, takeshi, hiroko
  とあり、y3w なら、,も 1 文字と考えるから、taro, hanako がコピーされる。これは 3yw としてもよい。y3W なら、taro, hanako, yoko がコピーされる。
- yy　yank の略。カーソルのある 1 行をコピーし、バッファに保存する。カーソルのある

行を含んで下に 3 行をコピーしバッファに保存したいなら、3yy とする。yy3 とすると、カーソルがある行を 3 回コピーすることになる。
当然ながら、別の行を yy すれば、保存してあるバッファは書き換えられてしまう。いろんな所の行をバッファに保存し、何回も使いたいことがある。その場合 は変数にバッファを保存する方法がある。 "a3yy で今カーソルのある行を含んで下に 3 行が変数 a に保存される。次にカーソルを別の行に移動し、"b2yy でカーソルのある行を含んで下に 2 行が変数 b に保存される。さらにカーソルを動かし、別の行に進み、"c4yy でカーソルのある行を含んで下に 4 行が変数 b に保存される。次に貼り付けたい所にカーソルを動かし、"ap で変数 a に保存したものが、貼り付けら れ、"bp で変数 b に保存したものが貼り付けられ、"cp で変数 c に保存したものが貼り付けられる。

- zb　b は bottom の略。カーソルのある行を画面の一番下にもっていく。
- zd　d は delete の略。折りたたんだ所にカーソルを移し、zd を入力することで、折りたたんだ文章がもとにもどる。
- zf　f は fold の略。選択した範囲を折りたたんで画面から消すことができる。長い文書を編集する時に、下の文章を参照にしながら編集したい時に、中の文 章を所を折りたためば、下の文章が画面に出るから、下の文章を見ながら編集できる。折りたたんだものをもとにもどすには、折りたたんだ所にカーソルを移 し、zd を入力する。
- zt　t は top の略。カーソルのある行を画面の一番上にもっていく。
- ZZ　変更を保存して vi を終了する。：x と同じである。Z は大文字だから Shift キーを使わなければならず、キー操作は ：x が少し速くなる。
- zz　カーソルのある行を画面の中央にもっていく。z.でも同じである。
- z.　カーソルのある行を画面の中央にもっていく。zz でも同じである。
- 0　これは数字の 0 である。今カーソルがある行の最初の文字にカーソルを移動する。0 は数字の始めだから最初というイメージがある。^との違いは、行の最初に空白がある時、0 は空白に移動するが、^は最初の文字に移動する。
- %　カーソルが()の上にある時、対応する()へ移動する。
- $　今カーソルがある行の最後の文字にカーソルを移動する。$は正規表現で文字列の終わりを示すから行の最後というのは連想できる。
- ^　今カーソルがある行の最初の文字にカーソルを移動する。^は正規表現で文字列の始まりを示すから行の最初というのは連想できる。0 との違いは、行の最初に空白がある時、0 は空白に移動するが、^は最初の文字に移動する。
- /　/の後ろに文字列を入力して Enter キーを押すことで、その文字列を検索できる。カーソルのある位置から下に検索する。次の検索結果に進むのは n を打つ。前の検索結果にもどるのは N を打つ。
  他のファイルを Ctrl-C でコピーしたものは、Shif-Ctrl-V でここに貼り付けることができる。
  検索は正規表現も使うことができる。今 taro を検索するとすると、
  /^taro　行の最初に taro があるのを検索する。
  /taro$　行の最後に taro があるのを検索する。

/の後に何も記入せずに Enter キーを打つと前回検索したものを検索する。前回は?で検索したものであっても、/の後に Enter キーを打って、前回?で検索したものを検索する。

- ?　?の後ろに文字列を入力して Enter キーを押すことで、その文字列を検索できる。カーソルのある位置から上に検索する。次の検索結果に進むのは n を打つ。前の検索結果にもどるのは N を打つ。
  検索は正規表現も使うことができる。今 taro を検索するとすると、
  ?^taro　行の最初に taro があるのを検索する。
  ?taro$　行の最後に taro があるのを検索する。
  ?の後に何も記入せずに Enter キーを打つと前回検索したものを検索する。前回は/で検索したものであっても、?の後に Enter キーを打って、前回/で検索したものを検索する。
- .　直前の変更操作を繰り返す。例えば dd で一行を消去して、次の一行も削除する時は、もう一度 dd とすればいいのだが、.でも同じことをする。こちらのほうがキーを 1 回打つだけだから少し速くなる。
- (　カーソルが空白行にない時は、上の空白行にカーソルが移動し、カーソルが空白行にある時は、空白行でない上の行の先頭にカーソルが移動する。コードを 書く時にまとまったコードの下に空白行を置いておくと、これでコードのまとまりごとにカーソルを移動することができる。
-)　カーソルが空白行にない時は、下の空白行にカーソルが移動し、カーソルが空白行にある時は、空白行でない下の行の先頭にカーソルが移動する。コードを 書く時にまとまったコードの下に空白行を置いておくと、これでコードのまとまりごとにカーソルを移動することができる。
- {　上の空白行にカーソルが移動する。
- }　下の空白行にカーソルが移動する。
- #　カーソルのある単語を検索し単語の背景色を変えて表示する。#を押していくと、最初にカーソルがあった所から、上方向に検索した単語の最初の文字に カーソルを移していく。よく$を打とうとして、間違って#を打ってしまうことがある。背景色表示がされ見にくいなら:noh を使うと背景色の表示がなくなる。
- *　カーソルのある単語を検索し単語の背景色を変えて表示する。*を押していくと、最初にカーソルがあった所から、下方向に検索した単語の最初の文字にカーソルを移していく。

- ~　カーソルのある文字が小文字ならば大文字に変え、大文字ならば小文字に変える。
- >　ビジュアルモードで選択した箇所に右のインデントを一つつける。
- >>　現在カーソルがある行にタブを一つつける。
- <　ビジュアルモードで選択した箇所に左のインデントを一つつける。
- <<　現在カーソルがある行のタブを一つ取り除く。
- @@　最後に実行した@文字(あるいは数字)を実行する。
- "　画面にある文字をレジスタに保存できる。ビジュアルモードで選択した後 b に保存するなら、"by で b に保存できる。カーソルのある行を b に保存したい なら"byy で保存できる。b に保存したものを呼び出すのは、"bp である。これで保存したものは@で呼び出すことは

できない。q を使った保存と同じ文字 を使うと上書きされてしまう。"の保存は q の保存より保存が簡単であるが、呼び出しは q の保存より少しキー操作が多くなる。
レジスタに保存するのは、:let @b="" でも保存できる。""の中に保存したいものを記入する。この時、@b と = の間にスペースを置くと、エラーになる。これで保存したものを呼び出すのも、"bp である。この場合は @b でも最初の 1 字がない状態なら呼び出すことができる。

- ! この後ろにコマンドを書くことで、そのコマンドを実行できる。例えば V でビジュアルモードにして、いくらかの文を選択する。その後！を入力すると、下に :'<,'>! と表示される。この後ろに sort を入力すると、アルファベットの順に並べ替えることができる。数字を選択して、文字コードの順でなく、数字の小さいもの から大きい順に並べるには、sort -n と n オプションを書く必要がある。
  計算もすることができる。数式を記入し、あるいはコピーし、その数式の行にカーソルを置いた状態で、!!と入力する。すると下に :.! と表示される。この後ろに bc と記入すると、その数式が計算された値に変わる。ただし割算は整数部分しか出ない。割算以外は電卓より便利である。電卓は複雑な計算だと、入力間違いがあ りうるが、入力をしたものが消えてしまうから、どんな入力をしたのかがわからない。この場合は数式をコピーしておけるから、どんな数式を計算したのかがす ぐにわかる。
  :r !date で現在の時刻を挿入でき、:r !pwd で現在使っているディレクトリを挿入できる。

:から始まるものは、入力したものが下に表示される。コマンドを入力した後に Enter キーを押す必要がある。

- :数字 数字の行番号に移動する。
- :%s/検索文字列/置換後文字列/gc s は substitue の略である。g は global の略である。c は confirm の略である。g がない と、1 行の内見つかった一番左の文字列しか置換しない。c がないと確認することなくすべて置換してしまう。置換したい場合は y、置換しない場合は n を入力 すると、置換したい所だけ置換することができる。すべて置換したいなら a(all の略)を入力すればよい。これでウインドウズテキストエディタの Ctrl キー+H キーと同じことができる。:s/検索文字列/置換後文字列/gc と%をなくすと、カーソルのある行だけを置換する。変換する所を例えば 10 行目と 45 行目の間に限定する時は、:10,45s/検索文字列/置換後文字列/gc とすればよい。:1,$s/検索文字列/置換後文字列/gc と書くと、:%s/検索文字列/置換後文字列/gc と同じことになる。検索後背景色が残り、見にくいなら:noh を使うと背景色の表示がなくなる。
- :b buffer の略。この後にバッファ番号を入力することで、そのファイルが開く。
- :bn b は buffer の略。n は next の略。次のバッファ番号のファイルが開く。
- :bp b は buffer の略。p は previous の略。前のバッファ番号のファイルが開く。
- :co copy の略。行をコピーする。co の前にコピーする行の範囲を指定し、co の後にコピーする行を指定する。co は、t あるいは copy と書いてもよい。例えば 3 行目から 6 行目までを 12 行目の下にコピーするなら
  3,6 co 12

と書く。現在カーソルのある行から 8 行目を 17 行目の下にコピーするなら
.,8 co 17
10 行目から最後までを 17 行目の下にコピーするなら
10,$ co 17
と書く。

- :copy　行をコピーする。copy の前にコピーする行の範囲を指定し、copy の後にコピーする行を指定する。copy は、t あるいは co と書いてもよい。例えば 3 行目から 6 行目までを 12 行目の下にコピーするなら
  3,6 copy 12
  と書く。現在カーソルのある行から 8 行目を 17 行目の下にコピーするなら
  .,8 copy 17
  10 行目から最後までを 17 行目の下にコピーするなら
  10,$ copy 17
  と書く。
- :d　delete の略。行を削除する。d の前にコピーする行の範囲を指定する。例えば 3 行目から 6 行目まで削除するなら
  3,6 d
  と書く。現在カーソルのある行から 8 行目削除するなら
  .,8 d
  10 行目から最後までを削除するなら
  10,$ d
  と書く。
- :e　edit の略。スペースをあけて、ファイル名を書きそのファイルを開く。
- :exit　:help の画面を開いた時などにその画面を閉じる。
- :help　:help の後ろに空白をあけて、調べたいものを入力すると、その説明が出る。閉じたい時は:q を入力する。
- :ls　:list の略。開いたファイルの一覧が出る。一番左側に表示されているのがバッファ番号である。前に開いたファイルをもう一度開くには、:b バッ ファ番号 と入力する。例えばバッファ番号が 2 なら、:b2 と入力し、Enter キーを押すとそのファイルが開く。ファイル名をすべて入力しなくていいのだから、 キー操作が速くなる。
- :m　move の略。行を移動する。m の前に移動する行の範囲を指定し、m の後に移動先の行を指定する。m は、mo あるいは move と書いてもよい。例えば 3 行目から 6 行目までを 12 行目の下に移動するなら
  3,6 m 12
  と書く。現在カーソルのある行から 8 行目を 17 行目の下に移動するなら
  .,8 m 17
  10 行目から最後までを 17 行目の下に移動するなら
  10,$ m 17
  と書く。

- :mo　move の略。行を移動する。mo の前に移動する行の範囲を指定し、mo の後に移動先行を指定する。mo は、m あるいは move と書いてもよい。例えば 3 行目から 6 行目までを 12 行目の下に移動するなら
  3,6 mo 12
  と書く。現在カーソルのある行から 8 行目を 17 行目の下に移動するなら
  .,8 mo 17
  10 行目から最後までを 17 行目の下に移動するなら
  10,$ mo 17
  と書く。
- :move　行を移動する。move の前に移動する行の範囲を指定し、move の後に移動先の行を指定する。move は、m あるいは mo と書いてもよい。例えば 3 行目から 6 行目までを 12 行目の下に移動するなら
  3,6 move 12
  と書く。現在カーソルのある行から 8 行目を 17 行目の下に移動するなら
  .,8 move 17
  10 行目から最後までを 17 行目の下に移動するなら
  10,$ move 17
  と書く。
- :n　n は next の略。開いている次のファイルに移動する。
- :new　上に新しいウインドウができる。:new の後ろにファイルを指定すると、そのファイルが開く。yy や dd によるウインドウ間の移動ができる。カーソルをウインドウの間で移動するには、Ctrl キーを押した状態で W キーを押した後に、J キーを押せば下に移動し、K キーを押せば上に移動する。一つのファイルを参照にしながら、もう一つファイルをつくるのに、便利である。しかし最近は横長のディスプレイになっているから、ウインドウを左右に分けたほうが見やすいように思う。左右に分けるに は:vnew を使う。
- :noh　no hilight の略。検索などで背景色がついた時、その背景色をなくす。
- :rew　rew は rewind の略。開いている前のファイルに移動する。
- :q　quit の略。vi を終了する。文書を変更して保存せずに、この操作をすると、「最後の変更が保存されていません」と警告画面がでて、この操作では終了できない。
- :r　read の略。r の次にスペースをあけて、ファイル名を入力する。カーソルの下にそのファイルの内容が挿入される。これは非常に役に立つ機能である。 プログラムを書く時最初からすべて書くと入力ミスがあったりして、予期しない不具合が出ることがある。それで問題なく動くとわかっている部品のひな形をつ くっておき、それをコピーして必要な所だけ変更して用いる。何回も使っているといろんなケースに遭遇するから、まれに起こるようなことも起こり、その部品 のひな形自体の不具合がわかることもある。人間は失敗して学ぶものだから、うまく動かないケースに遭遇して始めてその部品のひな形の欠陥に気づくのであ る。「ウインドウズテキストエディタでは、部品のひな形のファイルをいちいち開いて、コピーして貼り付けることになるが、この機能を使うとファイルを開く こともなく、貼り付けることができる。

:48 r ファイル名
とすると、49行目にファイルの内容が挿入される。 何も書かれていないファイルに
:r ファイル名
で読み込むと1行目の下に読み込まれるから、2行目から読み込まれて1行目が空白になってしまう。その時は
:r 0 ファイル名
とすると1行目から読み込まれる。ただしこれで読み込むと一番最後に空白行がひとつできる。

- :s/検索文字列/置換後文字列/g　s は substitue の略である。g は global の略である。現在カーソルがある行を置換する。g がないと、 1行の内見つかった一番左の文字列しか置換しない。確認が必要な場合は g の後ろに c を追加する。検索後背景色が残り、見にくいなら:noh を使うと背景色の表示がなくなる。
- :set nonu　nonu は no number の略。行番号を表示させない。
- :set nu　nu は number の略。行番号を表示させる。
- :t　行をコピーする。t の前にコピーする行の範囲を指定し、t の後にコピーする行を指定する。t は、co あるいは copy と書いてもよい。例えば3行目から6行目までを12行目の下にコピーするなら
  3,6 t 12
  と書く。現在カーソルのある行から8行目を17行目の下にコピーするなら
  .,8 t 17
  10行目から最後までを17行目の下にコピーするなら
  10,$ t 17
  と書く。
- :vnew　左に新しいウインドウができる。:vnew の後ろにファイルを指定すると、そのファイルが開く。yy や dd によるウインドウ間の移動ができる。カーソルをウインドウの間で移動するには、Ctrl キーを押した状態で W キーを押した後に、H キーを押せば左に移動し L キーを押せば右に移動する。一つのファイルを参照にしながら、もう一つファイルをつく るのに、便利である。上下に分けるには:new を使う。
- :w　write の略。文書を保存する。w の後ろにスペースをあけて、ファイル名を入力することで、そのファイル名で保存できる。スペースは一つでなくても 大丈夫である。指定したファイル名のファイルがすでにある時は「ファイルが存在します」という警告が出る。指定したファイル名で上書きする時は、w!の後 ろにスペースをあけて、そのファイル名を書く。
- :wq　write and quit の略。文書を保存して vi を終了する。変更がない場合は保存する。
- :x　文書を保存して vi を終了する。変更がない場合は保存しない。ウインドウズで文書を終了する時、x をクリックするが、そのイメージである。

1. 今ディレクトリ名を Shiken とすると、:e Shiken/ で Shiken ディレクトリの中のファイル名の一覧を表示できる。開きたいファイルにカーソルを移し、Enter キーを押すことでそのファイルを開くことができる。

2. 未入力の空白行は~がついている。
3. 入力モードは下に挿入と表示される。入力モードは最終行の右端のカーソルが文字の右側に行くが、コマンドモードは文字の上にとどまり、文字の右側には行かない。入力した文字はカーソルの左側に入っていくようになる。
4. 画面の右下に現在カーソルのある行と行の中の位置が表示され、文書が一番上から何パーセントぐらいの所にあるかも表示される。
5. vi で開いた文書に行番号をつくようにするには、端末で、
   echo "set number" >> ~/.exrc
   を実行する。
6. 端末に vi Shiken.txt と打ち込んで Shiken.txt を開いて、そのファイルを変更したのだけれど、保存せずに、端末を閉じてしまった時、そのファイル はスワップファイルとして自動的に保存される。次に vi Shiken.txt で Shiken.txt を開いた時、以前に保存していないファイルをどうするか聞いてくる。以前のファイルを削除するには、D を入力 する。
7. 今 shiken.pl を開いており、これを一応保存して、shiken2.pl をつくりたい時
   :w shiken2.pl
   でできる。この時、shiken.pl を変更して保存していなかったとしても、今の状態で保存される。shiken.pl に次の変更をすると問題が起こる可能性がある時、問題のない今の状態を保存しておくと、何か問題の起こった時に、
   gg
   dd
   G
   :r shiken2.pl
   で問題のなかった状態にもどすことができる。
8. vi で文書を開くと、^M の表示が文末に見られることがある。これは Windows のテキストエディターでつくったものを開いたり、Windows のテキ ストエディターでつくったものをコピーしたりした時に見られる。Windows は改行コードが CRLF であるが、Linux は LF である。CR は Carriage Return の略。LF は Line Feed の略。Windows には CR がついているため、これが^M として出てくる。vi で編集する時はこれは不要なものであり、これがついていると見に くい。これを削除するには、:%s/^M//g でする。^ M は制御文字であるため、キーボードの^ M から入力しても認識しない。Ctrl キーを押しながら V キーを打ち、Ctrl キーを押しながら M キーを打って入力する必要がある。
9. vi のコマンドモードは gedit などのテキストエディターに比べて、圧倒的な機能を持つ。しかし入力モードでは、gedilt のほうが上である。vi の 入力モードはマウスによるカーソルの移動ができないし、Ctrl-Z による undo もできない。だから入力が遅くなる。日本語入力の時は、半角/全角漢字 キーをしばしば押す必要があるが、Esc キーと半角/全角漢字キーは上下に並んでおり、ブラインドタッチでしているとしばしば Esc キーと半角/全角漢字 キーを押し間違える。Esc キーを押すと、コマンドモードにもどる。コマンドモードにもどったのに気づかずに入力すると、コマンドを入力するのだから、い ろいろと変なことが起こる。日本語をまったく入力しないなら、半角/全角漢字キーと間違えて Esc キーを押

すことはないから、gedit に比べる不便だ が、多くの入力でなければ許容範囲である。
論語に「備わるを一人に求めるなかれ」という一句がある。一人の人間に完璧を求めてはならないということである。テキストエディターもまた一つに完璧を求 めないほうがよいと思う。コマンドモードは vi を使い、まとまった入力は gedit を使えば、最強のテキストエディターになると思う。
端末で
vi henshu.txt
で開く。
ggdG
で henshu.txt をすべて削除する。
:x
で閉じる。 編集しているファイルのコピーをする最初の所にカーソルを移す。
v
ビジュアルモードにしてコピーする。
:e henshu.txt
で henshu.txt を開く。
P
で henshu.txt に貼り付ける
入力をする。入力が終われば保存して閉じる。
カーソルをコピーを開始した所から一行あげる。
:r henshu.txt
で読み込む。
dd
で以前の不要のものを削除する。

10. vim の設定ファイルは CentOS では、/etc/vimrc にある。デフォルトでは、行番号がつかないし、タブの幅が大きい。これを変更するには、/etc/vimrc に書き加える。
set ruler の下あたりに、
set number
と書き加えると vi で開いた時に行番号がつく。
タブの設定は、
set tabstop=5
でできる。
タブの設定変えるだけでは、< << > >> でタブを設定した時と入力モードでタブの設定をした時とのタブの幅が変わってしまう。それで、
set shiftwidth=5
で shiftwidth も変えておく必要がある。tabstop を 5 に設定した時は、shiftwidth も 5 にすると、同じタブ幅になる。