3.テキスト編集をする。

シェが使えるようになると次にやることはテキストファイルを編集することです。原稿を書くにしてもプログラム開発をするにしてもはじめはテキストファイルとして作る必要があります。

3.1.テキストエディターを使う。

現在UNIXで標準的に使われているスクリーンエディターはviとemacsです。ラインエディターとしてはexがあります。ほかにウインドウベースのエディターとしてSUNのtexteditやXのxeditなども使えます。UNIXの標準エディターとしてはviとexがすべてのシステムで使用できます。emacsもGNU emacsがほとんどのシステムにインストールされていると思います。これから覚えるのであればviとemacsでしょう。

テキストエディターの機能としては文字の入力、消去、挿入、移動などの編集、画面やカーソルの移動、文字列の検索と置換、ファイルの読み込み、書き出しができればとりあえず仕事はできます。それ以外にも様々な機能を持たせたものがありますが当面無視します。

3.1.1.viを使う。

viはUNIXの標準エディターとしてすべての機械で利用できます。色々なコマンドの中から呼び出され　ることも多いので知っておく必要があります。ただ、おいたちも古いので最近の種々のエディターに慣れている人には少し変に思えるかも知れません。スクリーンエディターですが、そのままタイプした文字がカーソルのあるところに書き込まれて行くわけではありません。というのも、エディターのモードというものがあって、そのモードによってタイプされた文字の扱いが変わるからです。はじめはコマンドモードになっています。aやbといった文字がそのままコマンドになります。文字列をテキストとして入力したいときは入力モードに移らなければなりません。一旦入力モードになるとタイプされた文字はすべてテキストとして扱われます。エディターに対してコマンドを与えたいときはコマンドモードに戻る必要があります。

もう一つ大事な点はviはexという行エディターを含んでいるということです。:ではじまるコマンドはexのコマンドで、最下行に表示されます。viのコマンドはエコーされません

3.1.1.1.開始と終了

はじめるときは、編集したいファイルを`test.doc`とすると、

```
% vi test.doc
```

編集をした後ファイルにセーブして終わりたいときは、コマンドモードで`ZZ`とタイプします。`:wq`でも同じ効果があります。
編集をしたあと訳がわからなくなって編集結果を残さずにやめたいときは`:q!`です。まだ編集を加えてない状態で終わるときは`:q`です

3.1.1.2.入力モードとコマンドモードの切り替え

コマンドモードから入力モードへ移るには、`i`もしくは`a`を使います。`i`はinsert、`a`はappendです。`i`ではカーソルの前に入力されていきます。`a`ではカーソルの後に入力されていきます。コマンドモードに移るにはESCキーを押します。現在どちらのモードにいるのかわからないときなど、とにかくESCキーをたたけばコマンドモードに戻ります。
`c`や`o`コマンドでも入力モードに入ります

3.1.1.3.カーソルの移動

普通の端末では矢印キーがついているので、それを使えばよいと思いますが、それは邪道だと書いている本もあります。矢印キーだけでは辛気臭いのでもっと簡単にカーソルを動かす方法を知っておくべきでしょう。以下すべてコマンドモードです。

一文字右へ	`l`	もしくはSPACE	
一文字左へ	`h`	もしくはBS	
一行上へ	`k`		
一行下へ	`j`	もしくはRETURN	
一語右へ	`w`	句読点や記号を無視するときは	`W`
一語左へ	`b`	句読点や記号を無視するときは	`B`
語の末尾へ	`e`	句読点や記号を無視するときは	`E`

これらのコマンドには数字を加えて、例えば４文字右へは`4l`などとできます。

行先頭に	0	空白文字以外	^
行末尾に	$		
n桁目へ	n|		

これらのコマンドはcシェルの引数の順序を表す記号と同じです。最初が0最後が$。これらの移動命令は、他の編集命令と組み合わせて使われます。例えばd$は行末まで消去します。cwはカーソルからその語の終わりまでを変更します。

ファイルの最後に	G	
ファイルの先頭に	1G	
n行目へ	nG	もちろんnは数字
次の行の先頭に	+	
前の行の先頭に	–	

スクリーン単位では

１画面分下へ	ctrl+F	
１画面上へ	ctrl+B	
1/2画面分下へ	ctrl+D	
1/2画面分上へ	ctrl+U	
画面の一番上へ	H	上からn行へ nH
画面の真ん中へ	M	
画面の一番下へ	L	下からn行へ nL
現在行を一番上へ	z<RETURN>	
現在行を真ん中に	z.	
現在行を一番下へ	z	

編集中にあとでここに戻ってきたいとかここから先を消したいとかでマークを付けたくなります。マークには名前を付けます。任意の一文字の名前です。ここでは例えばaとしましょう。

マークaを付ける	ma	
マークaのある行の先頭に飛ぶ	’a	シングルクオート

マークaに飛ぶ	‘a	バッククオート

飛ぶ前の位置もviは覚えていますその位置に戻るには

以前の位置の行の先頭に戻る	’’	シングルクオート
以前の位置に戻る	‘‘	バッククオート

これらは範囲指定に色々なコマンドと組み合わせて使えます。exコマンドの中でも有効です。

3.1.1.4.文字列の消去と変更

　文字を消したいときもただBSキーを押すだけでは消えません。BSキーはカーソルを一文字前に移動するだけです。コマンドモードで

カーソルから右へ１文字消去	x	
カーソルから左へ１文字消去	X	
次の語の頭までを消去	dw	
カーソルのある語の末まで消去	de	
カーソルから行末まで消去	D	
カーソルのある行一行消去	dd	
カーソルからマークaの行の先頭まで消去		d’a
カーソルからマークaまで消去		d‘a
コマンドの取り消し	u	
２行を１行に連結	J	

消去された文字列は消滅するのではなく削除バッファにコピーされています。pコマンドで張り付けることができます。ペーストするともいいます。

削除バッファの内容をカーソルの後ろに張り付ける。	p
削除バッファの内容をカーソルの前に張り付ける。	P

ということは、文字列を消去せずに削除バッファにコピーしたいこともあるでしょう。ヤンクというそうです。

一語削除バッファにコピー	yw	
行末まで削除バッファにコピー	y$	
一行削除バッファにコピー	yy	
マークa行先頭まで削除バッファにコピー		y’a
マークaまで削除バッファにコピー		y‘a

文字列を変更したいとき、消去して入力するのが手っ取り早いかもしれません。すでにaとiは出てきましたが、入力モードに入るコマンドをまとめると、

カーソルの後に入力	a	
カーソルの前に入力	i	
行の末尾に入力	A	
行の先頭に入力	I	
１行カーソルの下に開いて入力	o	
１行カーソルの上に開いて入力	O	
１文字を変更	s	
１語を変更	cw	
行末までを変更	c$	もしくはC
行先頭までを変更	c0	
１行を変更	cc	もしくはS

入力モードに入らずに１文字変更するのはrとタイプしたあと置き換えたい文字を入力します。

　同じ文字を沢山入力したいときがあります。この時は入力コマンドの前に回数を書いておきます。ESCが入力された時点で文字列がその回数だけ反復して表示されます。例えば

```
70i*ESC
```

とやると、*を70個連続して入力したことになります。

　制御文字を入力したい場合もあります。入力モードでctrl+Vに続けて制御文字をそのままタイプします。ctrl+V ctrl+Lでctrl+Lが入力されます。^Lと表示されます。同じようにctrl+V ctrl+Gとやればctrl+Gが入力できます。

3.1.1.5.文字列の検索と置換

文字列の検索は/コマンドでおこないます。/をタイプすると最下行に/が表示されます。それに続いてタイプしたパターンがRETURNを押すまで検索文字列となります。パターンには正規表現が使えます。逆方向に検索したいときは/の替わりに?を使います。

検索を繰り返すときは、

同じ方向に次を探す	n
逆の方向に次を探す。	N
下の方向に次を探す。	/RETURN
上の方向に次を探す。	?RETURN

同じ行の中の１文字の検索は、探す文字をxとすると、

その行の次のxを探す。	fx
その行の前のxを探す。	Fx
次のxの前の文字を探す。	tx
次のxの後の文字を探す。	Tx

検索のパターンには正規表現と呼ばれる特別の文法による表現が使えます。これは例えばシェルにおいてワイルドカードと呼ばれる任意の文字を表す記号*等と似ています。そういった特殊文字をメタキャラクターと呼びます。次のようなものがあります。

. ピリオド。改行以外の任意の一文字。空白も含む。
* 直前の文字の0個以上の繰り返し。bugs*はbug、bugs、bugssにもマッチ。bugがマッチするのはsが0個あると考えるから。.*は0個以上の任意の文字列に。
^ 次の文字が行の先頭にある。^Partは行の先頭にあるPartという文字列。^.は先頭の一文字にマッチ。
$ 直前の文字が行の最後にある。end$は行末にあるendと言う文字列。.$は行末の一文字を表す。
\ 次の文字を普通の文字として扱う。\.はピリオド、*はアスタリスクとマッチする。
[] 括弧内の文字のいずれか。-で範囲を表す。[a-zA-Z]は英文字一字。[の次の文字が^の場合は否定(それ以外の文字)。
\< 単語の先頭にマッチ。\<aはaではじまる単語にマッチ。
\> 単語の末尾にマッチ。z\>はzで終わる単語にマッチ。
\(\(と\)で囲まれた部分を保存。あとで\1から\9で呼び出せる。
\1 1番初めに定義された\(と\)で囲まれた部分の呼出し。\2は2番目、\3は3番目。

同じ方向に次を探す。	;
逆方向に次を探す。	,

現在行のあるパターンを置換をするにはexコマンドを使って

現在行の最初の一致パターンの置換	:s/old/new/RETURN
現在行のすべての一致パターンの置換	:s/old/new/gRETURN
ファイル内のすべての一致パターンの置換	:%s/old/new/gRETURN

ここで%は1,$と同じ、つまり１行めから最後の行までの意味です。/を含む文字列の置換をしたいときは他の文字を/の替わりに使えます。

3.1.1.6.ファイルへの書き込み、ファイルからの読み込み

最初にも少し触れましたが、ファイル関係のコマンドは、exコマンドで、

ファイルへの書き出し	:w
他のファイルへの書き出し	:w newfile.txt
他のファイルへの追加	:w >> newfile.txt
現在の行へのファイルの読み込み	:r oldfile.txt
ファイル先頭への読み込み	:0r oldfile.txt
ファイル末尾への読み込み	:$r oldfile.tx

3.1.1.7.その他

名前付きバッファの使い方、正規表現の詳細、exコマンドの多くについては省略しました。viにはヘルプ機能はないようです。manでもまともな説明は得られません。これ以上は参考書を読んでください。例えば「vi入門」Linda Lamb,福崎俊博訳、アスキー出版

3.1.2.emacsを使う。

emacsもよく使われるエディタです。一部のアプリケーションはemacsの中に組み込まれたものもあります。例えばニュースへのインターフェースなどです。それらを利用

するためにもemacsは知っておかなければならないエディタです。

emacsはモードのないエディタなのでviに比べればわかりやすいかも知れません。タイプした印刷可能文字は全て入力文字として現在のカーソルのある位置に書き込まれていきます。ですからエディタへの命令はコントロールキーやメタキー(ないときはエスケープキーを先に押すことで代用)とアルファベットの組み合わせで与えてやります。コントロールキーを押しながらアルファベットキーを押すことを例えばC-aと書き表すことにします。メタキーを押しながらアルファベットキーを押すことをM-aで表します。キーボードにメタキーがないときは先にエスケープキーを押してから次にアルファベットのキーを押します。場合によってはC-M-a等となっていることもあります。コントロールキーとメタキーを同時に押しながらということですが、エスケープキーを使う場合はエスケープキーを押してからC-aをタイプすることになります。

emacsの場合は制御キーの割付は自由に変更できます。いろいろなemacs命令に標準的に割り付けられたキーをここでは解説します。キーに割り付けられた命令をあわせて()内に上げておきます。キーを覚えるための参考にもなると思います。また、キーの割り付けられていないコマンドもあります。この場合はコマンド名で呼び出すか、自分の好みでキーを割り付けて使います。

emacsの場合も編集中のテキストはバッファに記録されています。ファイルはバッファに読み込まれ編集された後ファイルへ書き込まれます。

3.1.2.1.開始と終了

はじめるときは、編集したいファイルをtest.docとすると、

```
% emacs test.doc
```

emacsを起動すると画面の下の方に白黒反転した行が表示されます。現在編集中のファイルの名前や状態が表示されるモード行と呼びます。モード行の次の行はコマンドの引数などを入力するときの編集行として使われます。

編集を終了するときはC-xに続いてC-cと入力します(save-buffers-kill-emacs)。編集中のファイルがあれば保存するかどうか聞いてくれます。C-x C-s(save-buffer)で保存してからC-x C-cで終了すると保存すべきファイルがないので問い合わせずに終了します。

3.1.2.2.カーソルの移動

端末の矢印キーを使って移動することももちろん出来ます。それでももっと簡単に移

動したくなるものです。

一文字右へ	C-f	(forward-char)
一文字左へ	C-b	(backward-char)
一行上へ	C-p	(previous-line)
一行下へ	C-n	(next-line)
一語右へ	M-f	(forward-word)
一語左へ	M-b	(backward-word)
行の先頭へ	C-a	(beginning-of-line)
行の末尾へ	C-e	(end-of-line)

これらの命令は数引数を与えて例えば5行先に行け等とやりたいものです。emacsの命令はすべて数値を引数として与えることが出来ます。C-uに続いて(universal-argument)数字を入力し次に実行したい命令をタイプします。例えば25行先に行きたいときはC-u 25 C-nとやります。数字を省略すると4が仮定されます。C-u C-fは4文字先へ進めです。C-uが繰り返されると、引数の値は掛け算されていきます。C-u C-u C-fは16文字先へ進めという意味になります。

バッファの先頭へ	M-<	(beginning-of-buffer)
バッファの最後に	M->	(end-of-buffer)
カーソルを中央に	C-l	(recenter)
順方向スクロール	C-v	(scroll-up)
逆方向スクロール	M-v	(scroll-down)
左にスクロール	C-x <	(scroll-left)
右にスクロール	C-x >	(scroll-right)

3.1.2.3.文字列の消去と変更

文字を消すには普通の感覚でDELキーで消すことが出来ます。それ以外にもいろいろ用意されています。

カーソルの前一文字	DEL	(delete-backward-char)
カーソル位置の文字	C-d	(delete-char)
単語の末尾まで	M-d	(kill-word)
単語の先頭まで	M-DEL	(backward-kill-word)
行末まで	C-k	(kill-line)

文字charまで	M-z char	(zap-to-char)
もとに戻す	C-x uもしくはC-_	(undo)
マーク	C-SPCもしくはC-@	(set-mark-command)
領域を削除	C-w	(kill-region)
領域を削除バッファにコピー	M-w	(copy-region-as-kill)
削除したテキスト取り込み	C-y	(yank)

間違って消してしまいました。もとに戻すにはC-x uもしくはC-_とやります(undo)。emacsは変更をずっと覚えているのでかなり以前に変更したところまで戻ってくれます。ある特定の領域を消したいときは、その一端にカーソルを移動してC-SPCもしくはC-@でマークを付けて(set-mark-command)、別の端までカーソルを移動してC-wで領域ごと消去します(kill-region)。カット・ペーストのカットです。この時M-wとやると消去はせず削除バッファにコピーを作ります。カーソルを移動してC-yでペーストしてやることが出来ます。C-yは領域だけでなく全ての削除命令の結果最後に消されたものが使われます。

emacsではモードがありませんから入力された文字はそのまま書き込まれていきます。同じ文字を沢山入力したいときはどうしたらよいでしょうか。すでにC-uを知っています。C-u70*と入力すると、*という文字の入力が70回繰り返されることになります。制御文字を入力したい場合もあります。このときはそれがemacsのコマンドではなく入力したい文字そのものであることを教えてやらねばなりません。そのためにC-qを使います。(quoted-insert)例えばctrl+Lを入力したいときはC-q C-lとなります。

3.1.2.4.文字列の検索と置換

文字列の検索は

下方向に検索	C-s ESC 文字列 RET	(search-forard)
上方向に検索	C-r ESC 文字列 RET	(search-backward)

この二つが通常のエディタで使われる検索命令に当たります。emacsの場合よく使われるのは増分検索(incremental search)という方法です。

順方向増分検索	C-s	(isearch-forward)
逆方向増分検索	C-r	(isearch-backward)

まずC-sと入力すると増分検索に入ります。探したい文字を一文字入力します。すると

最初に見つかったその文字のところへ進みます。探したい文字の2文字目を入力します。するとその2文字の組み合わせの最初のもののところへ進みます。まだ目的の所へたどり着いていないときはさらに3文字目を入力してやります。このようにして1文字づつ順に入力して行く過程でemacsは候補地を探してゆくので、少ないタイピングで検索が出来るというわけです。目的地にたどり着いてそれ以上探す必要のないときはESCキーを押してやります。

正規表現による検索も可能です。C-M-sです(isearch-forward-regexp)。逆方向への検索もできますが(isearch-backward-regexp)、キーには割り当てられていません。使える正規表現はviとほぼ同じです。3.1.1.6.を参照してください。

現在のカーソルから先にある一致する文字列を全て置き換えるには、replace-stringやreplace-regexpコマンドを使います。キーには割り当てられていませんから次のように使います。

M-x replace-stringRET旧文字列RET新文字列RET

「旧文字列」が全て「新文字列」に置き換えられます。一つ一つ確認しながら置き換えていきたいときは

M-%旧文字列RET新文字列RET

とやります。query-replaceです。正規表現で検索する場合はquery-replace-regexpコマンドを使います。

3.1.2.5.ファイルへの書き込み、ファイルからの読み出し

現在編集しているファイルをセーブするにはC-x C-sです(save-buffer)。別のファイルに書き出したいときはC-x C-wでファイル名を入力します(write-file)。選択した領域を書き出すにはwrite-regionコマンドを使います。領域の設定はその一端にC-SPCもしくはC-@でマークを付けてカーソルを書き出したい領域の別の端に移動します。

ファイルを読み込むときはC-x C-fです(find-file)。読み込みたいファイル名を指定してRETを押します。この場合は新たなバッファが用意されます。現在使用中のバッファの中身をクリアしてそこに読み込むにはC-x C-vとやります(find-alternate-file)。現在編集中のカーソル位置にファイルを読み込むにはinsert-fileコマンドを使います。

3.1.2.6.バッファとウィンドウ

emacsではいくつものバッファを使います。複数のファイルを同時に編集をすることも出来ますし、emacsのシステム自体が使うバッファもあります。それに対応して複数のウインドウを表示して編集することが出来ます。どのようなバッファが存在するかを表示させるのにC-x C-bとやります(list-buffers)。画面に別のウインドウが表示されてバッファのリストが出てきました。この画面は通常の編集は出来ませんが、バッファのリストのカーソルのあるバッファに対して次のコマンドが入力できます。

d	バッファの消去をマーク(xコマンドで実行)
s	バッファの保存をマーク(xで実行)
u	dやsのマークの取り消し。
x	dやsの実行
f	カーソルのあるバッファを現在のウインドウに表示
1	表示ウインドウを1つだけにしてカーソルのあるバッファを表示
2	2つ目のウインドウにバッファを表示

ウインドウの間を行き来するコマンドも必要です。

C-x o	他のウインドウを選択。oはオーです。
C-M-v	他のウインドウをスクロール。
C-x 0	現在のウインドウを消去。0はゼロです。
C-x 1	他のウインドウを全て消去。
C-x ^	現在のウインドウの縦の寸法を拡大。
C-x }	現在のウインドウの横の寸法を拡大。

3.1.2.7.その他

キー割付のされていないコマンドの使い方を説明していませんでした。M-xでemacsコマンドの入力が可能になります。綴りのとおりに入力すると文字列がエコー行に表示されます。最後にRETを押してやります。実はemacsは自分の理解できるコマンドを知っているので全ての文字を入力する必要はありません。例えばinsert-fileコマンドを入力するにはinsまでタイプしてあとTABキーを押すと、insert-までemacsが続きを補ってくれます。そこでさらにfTABとやるとinsert-fileにしてくれます。こういう機能を補完機能と呼びます。

emacsはヘルプ機能が充実しています。C-hでヘルプ機能を呼び出します。次の一文字がヘルプへのオプションになります。C-hもオプションの一つで、どのようなオプションがあるか表示してくれます。オプションとしては次のものなどがあります。

C-h a文字列RET　　「文字列」を名前に含むコマンド一覧表示
C-h b　　キー割当表を表示(describe-bindings)
C-h cキー　　「キー」に割り当てられたコマンドを短く表示
C-h kキー　　「キー」に割り当てられたコマンドを詳しく表示
C-h wコマンドRET　　「コマンド」に割り当てられたキーを表示
C-h i　　emacsマニュアルを表示
C-h m　　現在の主モードに関する文書を表示
C-h n　　最近のemacsに関する文書を表示
C-h s　　現在の構文表を表示
C-h t　　emacsチュートリアルを表示

emacsは非常に大きなシステムなのでヘルプ機能を使いこなすことが非常に有効です。ますはC-h tでチュートリアルに付き合ってみてはいかがでしょうか。

ここに書いたことはemacsに出来ることのうちのごくわずかな部分です。それでもテキスト編集には十分使えると思います。実際にはemacsの中にいたままでプログラム開発をしたり、メイルを読んだりニュースを読んだりもできます。自分の好みに合わせて改造したりすることもできますし、新たな機能を付け加えることも出来ます。それらのことはとてもここでは書ききれません。「GNU Emcasマニュアル」Richard Stallman著、竹内郁雄・天海良治監訳、共立出版など参考にしてください。

3.1.3.sedを使う

例えば送られてきたメイルにPostscriptやTeXのデータが含まれている場合、最初の数行だけ消したいというときがあります。このような場合、わざわざviやemcasを立ち上げるのも大層だというときに使える簡単なエディタがあると便利です。sedというエディタがそれです。sedのsはストリームの意味らしいです。edはまさにエディタの意味です。ストリームとはすでに見てきた標準入力や標準出力のことです。標準入力のデータに適当な加工をして標準出力に出すというのがsedの働きです。

3.1.3.1.基本的な使い方・一部の行の取り出し

sedのコマンドの使い方には、シェルコマンド行に直接sedのコマンド(スクリプト)を書く場合と、ファイルにスクリプトを書いておいてそれを呼び出す形式とがあります。ちょっとしたことには前者が手間がかからなくていいわけですし、何度も繰り返すものについてはファイルにしておいたほうが楽かも知れません。

```
% sed -e {スクリプト} {入力ファイル}

% sed -f {スクリプトファイル} {入力ファイル}
```

この場合の出力先は標準出力にされます。結果をファイルに残したいときはリダイレクトすることになります。>出力ファイル名をコマンドのあとに付ければいいわけです。ではさっそく試してみましょう。
　とりあえずファイルの中身をそのまま出力します。

```
% sed -e p test.txt
```

この例ではtest.txtというテキストファイルが処理の対象です。何でもいいので適当にやってみてください。ここで-eはこの次の引数がスクリプトであることを表しています。この-eは省略できます。つぎのpがスクリプトで、プリントしなさいという意味です。不思議なことにtest.txtの中身が2度づつ表示されました。これはsedが入力行をデフォールトで出力するうえに、pスクリプトによって全部の行を表示せよと指示された結果だからです。これはちょっと困ります。そこでデフォールト出力を禁止します。

```
% sed -n -e p test.txt
```

ここで-nがデフォールト出力抑制のスイッチです。コマンドpは全部の行を表示します。実はpの前に範囲が指定されていないので全域を対象としました。色々な範囲指定の仕方があります。

```
% sed -n -e 1,5p test.txt
% sed -n -e '15,$p' test.txt
% sed -n -e /sakanaya/p test.txt
% sed -n -e /begin/,/end/p test.txt
```

最初の例では1行目から5行目までを範囲とします。次は15行目から最後($で表す)までを範囲とします。その次の例はsakanayaという文字列を含む行をすべて、最後の例ではbeginを含む行からendを含む行までを範囲とします。ファイルの中身を調べるためにも非常に重宝です。メイルの中からPostscriptやTeXのデータだけを取り出すには、

```
% sed -n -e '/%\!PS/,$p' tsdemo.txt > psdemo.ps
% sed -n -e '/\\documentstyle/,$p' texdemo.txt > texdemo.tex
```

最初の例はPostscriptファイルが通常%!PS-Adobeというコメント行で始まっていることを使っています。TeXの場合は最初の行は\documentstyle文であることからこのような方法が使えます。(もちろんその前にコメントを書いてある場合もありますが...)

3.1.3.2.文字や文字列の変更や消去

文字列の変更もsedの重要な使い方の一つです。viの置換と良く似ています。

```
% sed -e 's/sakanaya/iwashi/g' test.txt
% sed -e 's/^  *//' test.txt
% sed -e 's/int/integer/g' test.txt
% sed -e 's/\<int\>/integer/g' test.txt
```

最初の例ではtest.txtの中にあるsakanayaがすべてiwashiに置き換えられます。2つ目の例では行の先頭にある(^が行の先頭を表す。)1つ以上の空白(正規表現で空白2つと*)がすべて取り除かれます。3つ目の例ではintがすべてintegerに置き換えられます。これはちょっと乱暴です。printもprintegerになってしまいます。最後の例はintという単語だけをintegerに変えます。\<が単語の最初の文字であることを\>が単語の最後の文字であることを指定します。これらのsコマンドには範囲指定がないので全域が対象になります。-nを指定しませんので変更後の結果がすべて出力されます。

3.1.3.3.高度な使い方

ここではpとsしか説明しませんでしたが、他にも色々なコマンドが用意されています。ただ-eオプションで一行で済すことができるのはこの程度のものです。スクリプトを書いて-fで呼び出すことが必要になります。この場合はほとんどプログラムを書くのと同じようなものです。詳しくはman sedで見てください。

sedの他にawkやperlといったストリーム処理用のコマンドもあります。

3.1.4.日本語を使う

ここまでテキストの編集は英数字を用いたものだけを説明してきました。プログラムを書く場合はそれでいいかも知れませんが、実際には日本語による文章を書く必要が多くあると思います。電子メールを送る場合でも日本語で書いたほうが気楽でしょう。

3.1.4.1.日本語の取り扱いかた

UNIXでの日本語の取り扱いはパソコンなどと基本的に同じです。キーボードから入力された文字はローマ字からかなに変換され、適当な変換キーを押すことによって漢字仮名交じりに変換されます。スクリーンに表示されるときは端末が漢字表示可能なものであればそのまま表示されます。日本語を使うために覚えておく必要があるのは、ローマ字・かな・漢字変換をする仕組みだけです。

日本語変換システムで最も良く使われているのはWnn(「私の名前は中野です。」の頭の文字)といものです。変換プログラムの名前はuum(Wnnを上下ひっくり返した)です。このプログラムはシェルに対するフロントエンドプロセッサとして働きます。キーボードから入力された文字は一旦uumを通って漢字に変換されてからシェルに渡されます。シェルが起動したプロセスに対してもuumを通した文字が渡されることになります。ですからシェルでuumを呼び出した後、そのシェルからviを起動することによってファイルの編集に日本語が使えることになります。

通常UNIXで使われる文字はASCII文字で、すべて1バイト、表示される大きさは半角です。それに対して日本語は2バイトからなり、表示される大きさも全角となります。日本語の取り扱い方を知らないプログラムに漢字を渡しても正しく取り扱われないことがあります。例えばDECstationではviエディタは/usr/ucb/viと/usr/jsy/bin/viの二つがあります。想像できるように後者は日本語を扱えるように修正されたviです。そうでないほうは漢字一文字の右半分を消したり変なことをしてしまいます。

日本語の編集にはnemacsがよく使われます。これはemacsの日本語拡張で、上述の場合と違ってemacsの中で日本語変換システムを起動します。変換システムの名前は「たまご」です。(「大変待たせてごめんなさい」の頭)キーの操作方法は基本的にuumと一緒です。

3.1.4.2.uumの開始と終了

それでは実際にuumを使ってみましょう。日本語が表示できる端末やウインドウからuumと入力します。uumは普通/usr/local/bin/Wnn4にあります。自分のpathに入れておく必要があります。わからないときはシステム管理者に聞きましょう。

```
% uum
ｕｕｍ（かな漢字変換フロントエンドプロセッサ）
%
```

と同時に画面の一番下に

```
[---]
```

と表示されます。[---]は変換機能がオフであることを示しています。これまでと同様入力した文字がそのままシェルにわたって行きます。コマンドを入力する場合はこのままでよいわけです。変換機能をオンしましょう。ctrlキーと\キーを同時に押します。(以後ctrl+\と表記します。)画面の下の表示が

```
[あr]
```

に変わりました。変換機能がオンになったことを表します。ここで文字を入力すると画面最下行[あr]の横にひらかなに変換されて表示されます。ctrl+wで漢字に変換されます。正しく変換されていればctrl+lで確定します。シェルのプロンプトの後に確定した文字列が移りました。シェルに渡されたわけです。

　変換をしてもらいたくないときは再びctrl+\と入力します。画面下の表示が再び

```
[---]
```

に戻りました。変換機能がオフになりました。ここでexitとやると

```
% exit
ｕｕｍを終わります。
%
```

となってuumは終了しますがシェルはまだ生きています。uumはシェルの子プロセスだったわけです。

3.1.4.3.日本語変換の方法

　uumの日本語変換は連文節変換です。自動変換ではありませんからそのつど変換をし

てやらなければなりません。変換のためのキーには次のものが使われます。

ctrl+w	変換。とにかく変換させてみます。
ctrl+l	確定。正しく変換されていれば確定します。
ctrl+n	次候補。正しいものが出るまで探します。
ctrl+p	前候補。行き過ぎてしまったとき。

「ここでは着物を脱ぐ」と「ここで履物を脱ぐ」のように変換は文節単位で行なわれます。「ここでは」から「ここで」に文節を縮めたり、逆に延ばしたりするときは、

ctrl+o	文節延ばし。「ここで」を「ここでは」に
ctrl+i	文節縮め。「ここでは」を「ここで」に
ctrl+f	次の文節へ。「ここでは」から「着物を」へ
ctrl+b	前の文節へ。「着物を」から「ここでは」へ
ctrl+a	最初の文節へ。「ここでは」へ。
ctrl+e	最後の文節へ。「脱ぐ」へ。

英数字を変換せずに入力したいときはctrl+\で変換機能をオフにしてもいいですが、

q	半角英数字モードに。[あr]が[q]に変わる。
Q	全角英数字モードに。[あr]が[Q]に変わる。
ctrl+g	qモードやQモードから[あr]に戻す。

これはたまごにはありません。

　平かなのままにしたいときは変換せずに確定。片仮名にしたいときはctrl+wで変換してすぐに前候補ctrl+pで文節が片仮名になるのでここで確定。JISコードで入力したいときはctrl+_とするのですが、普通の端末では_はシフトキーを押しながら入力する必要があるのでctrl+_は正しく送られないようです。この場合キーバインドの変更が必要です。

3.1.4.4.より快適な変換システムへ

　より快適な日本語変換にしていくには、一つには辞書を鍛えること、もう一つは変換システムのパラメータやキー設定を使いやすいように変更することです。

　辞書登録は、登録したい文字列を[あr]の横に表示させた状態でctrl+tを押し、登録始め:と登録終わり:をリターンキーで設定した後、読み:でひらかなの読みを入力して品詞

を選択します。途中でわからなくなったりしてやめるときはctrl+gです。ctrl+^が使えると辞書操作のメニューを使って辞書の保守が出来ます。

変換システムのカスタマイズは入門とは言えないので詳しくは触れませんが、自分専用のuum初期設定ファイルは.uumrcです。(たまごの場合は.eggrcです。)自分専用のキーバインドは.uumkeyと言うファイルで、いずれも自分のホームディレクトリに置きます。システム標準のファイルは/usr/local/lib/wnnにuumrcやuumkeyという名前で置かれています。自分でもってきて編集すればよいわけです。man uumrcやman uumkeyで見てください。

3.1.4.5.漢字コード変換

ネットワーク経由で日本語のファイルを持ってきたとき、場合によっては自分の計算機の使う漢字コードと違うコードで書かれている場合があります。現在ワークステーションで使われている漢字コードにはEUC(ujis)、シフトjis(sjis)、7ビットjis(jis)、DEC漢字などがあります。電子メールなどはネットワークを経由するとき自動的に7ビットjisコードに変換され相手計算機でその計算機標準のコードに変換されるので気にすることはありません。そうでない方法でバイナリーファイルとして送られて来た場合などコード変換をしてやらないとうまく表示されない場合があります。そのためのコマンドは計算機毎に違っていますが、概念的には同じようなものがあるはずです。例えばSUNではEUCが標準ですが、

euctojis	EUCからjisへの変換。
jistoeuc	jisからEUCへの変換。
euctosj	EUCからシフトjisへ。
sjtoeuc	シフトjisからEUCへ。
jistosj	jisからシフトjisへ。
sjtojis	シフトjisからjisへ。

が用意されています。DECではdec_jis、dec_sjis、dec_ujis、jis_dec、sjis_dec、ujis_decなどが、HPではstoj、jtos、utoj、jtou、stou、utosなどがあります。コマンド名から想像がつくでしょう。これらはフィルターコマンドとして使えます。

nkfというコマンドがインストールされた機械も多いと思います。このコマンドの優れているところは入力側の漢字コードを自動判定してくれることです。ですから出力コードとして何を選ぶかだけ考えればよいわけです。nkf -eで出力はEUCに、nkf -sでシフトjisに、オプション無しでjisに変換します。

3.2.ファイルを管理する。

ファイルの編集ができるようになりました。自分のサブディレクトリにいくつもファイルができはじめるとそれらを管理する必要が出てきます。関係するコマンドを見てみましょう。

3.2.1.ファイル管理関係のコマンド

ファイルの削除はrmコマンドを使います。removeでしょう。

```
% rm filename [...]
```

rmの後には複数のファイル名を指定できますし、正規表現が使えます。

ファイルの複写はcpです。copyですね。

```
% cp file1 file2
% cp ../doc/* .
```

最初の例ではファイルfile1をfile2にコピーしています。次の例では、相対ディレクトリ../docにあるすべてのファイルを現在のディレクトリにコピーしています。

ファイルの移動はmvです。moveですね。

```
% mv file1 file2
% mv ../doc/* .
```

２つめの例では相対ディレクトリ../docにあるすべてのファイルを現在のディレクトリに移動しています。最初の例はというと、file1という名前のファイルをfile2という名前のファイルに「移動」しました。つまり名前を付け替えたということです。

非常にたくさんのファイルができてくると探し方を知っておく必要が出てきます。あるディレクトリにいるときその下のどこかにあるファイルを探すコマンドがfindです。findはこれまでのコマンドと雰囲気が違います。例えばファイル名の最後に.txtという文字列を含んだファイルを現在のディレクトリの下にみつけるには、

```
% find . -name '*.txt' -print
```

ちょっと説明が必要でしょう。findの次の.は現在位置から下に向かってファイルを探せという指定です。次の-nameオプションはファイルを名前で探せというものです。その名前は'*.txt'で与えています。''で囲まないとそのディレクトリの中のファイルで*が置き換えられてしまうので目的をたっせません。最後の-printは得られた結果を表示しなさいという指定です。findを使うと、名前のほか日付や属性などでもファイルを探せます。詳しくはmanを見てください。

ある特定の文字列を含んだファイルを探すこともできます。grepを使います。

```
% grep pattern filename
```

filenameというファイルにpatternという文字列が含まれているかどうかを探します。どちらの表現にも正規表現が使えます。そのときはpatternについては''で囲うほうが安全です。

ファイルの属性を見るにはls -lでサイズ、日付、所有者、属性などを見ることができますが、fileコマンドもファイルがどんなものか知るうえで重宝します。

自分がどのくらいファイルを作ったかを知りたいこともあります。ファイルシステムに使用量制限が設定されているときはquota -vでわかります。設定されていない場合quotaはなにも言ってくれません。duコマンドはファイルが使用しているディスクサイズを表示してくれます。単位はkBです。例えば

```
% du -s ~
5815    /home/iwashi
%
```

自分のホームディレクトリから下にあるファイルが使っているディスク容量の合計を-sオプションで表示させました。今の場合5815kB使っています。ファイルシステムの混み具合をみるためにはdfというコマンドも用意されています。

3.2.2.ファイルの印刷

ファイルの内容を表示させるにはcatを使いました。moreも使えます。でも紙に印刷したいこともあります。印刷にはlprコマンドを使います。

```
% lpr myfile.txt
```

これでmyfile.txtというファイルが印刷のためにスプールされます。プリンタ出力待

ち行列に登録されるという意味です。実際に印刷されたかどうかは別問題です。どうなっているかを聞いてみましょう。待ち行列queueの状態を表示させるlpqを使います。

```
% lpq
lp is ready and printing
Rank   Owner      Job  Files                 Total Size
active iwashi    451  standard input       124672 bytes
%
```

印刷を取りやめたいこともあります。queueから取り除きます。今の例では

```
% lprm 451
```

ジョブ番号451のジョブを待ち行列からremoveします。

ここまでの例ではデフォルトのプリンタを使ってきました。名前はlpです。実際にはネットワーク経由などでいろんなところのプリンタに出したいことがあります。どのようなプリンタが接続されているかは、/etc/printcapというファイルに書かれています。このファイルを読めば希望するプリンタの名前を見つけることができます。例えば近くのプリンタの名前がlwplusだとすると、上のlpなにがしコマンドに-Pオプションを付けてプリンタを指定します。

```
% lpr -Plwplus myfile.txt
% lpq -Plwplus
% lprm -Plwplus 451
```

といった具合になります。

テキストファイルを印刷するときに、ページ数を振ったり段組みしたくなります。そのためのフィルターが用意されています。prです。あまり気が利いているようには見えませんが。

ワークステーションにつながっているプリンタのほとんどがPostscriptプリンタになってきています。世の中にはPostscriptに変換するもっとスマートなテキストフォーマッタを使っている人が多いようです。

プリントをするとき、スプールされるといいました。印刷されるファイルはこの時一旦自動的にスプールディレクトリにコピーされます。印刷するファイルが非常に大きいとスプールディレクトリにコピーできない場合があります。このときはファイルをコ

ピーせず後で解説するシンボリックリンクを張ることによってプリントさせることが出来ます。

```
% lpr -s very.big.file
```

オプション-sがシンボリックリンクを張るよう指示します。この時は印刷中のファイルを消したり編集したりすると印刷結果がおかしくなります。

3.2.3.ファイルのバックアップ

ファイルがたくさんできだすと時々バックアップを取っておきたくなると思います。ワークステーションがテープドライブを持っているときはテープに落としますが、UNIXではテープは標準のファイルシステムの範疇に入りません。それゆえcpではコピーできません。専用のコマンドを使う必要があります。ファイルシステム全体をバックアップするときはdumpを使いますが、個人のファイルの管理のためにはtarを使うのがよいでしょう。tape archiverの意味です。tarではテープの替わりに通常のファイルをバックアップセットに設定できます。例えば自分のたくさんのファイルを一つのバックアップセットにしてから（さらに圧縮してから）よその機械にネットワーク経由で転送するなどの使い道があります。

```
% tar cvf /dev/rmt0 .
```

この場合はrmt0というテープドライブのテープに現在のディレクトリのファイルをセーブします。cはアーカイブファイルの生成、vは冗長モード、fはアーカイブファイル指定の意味があります。テープの中身を知りたいときは

```
% tar tvf /dev/rmt0
```

この場合のtは内容表示をしめします。またファイルをもとに戻すには

```
% tar xvf /dev/rmt0
```

/dev/rmt0はテープです。tarの場合通常テープの初期化やマウントなどの手続きは必要ありません。直接書き始めます。アーカイブファイルとして通常のファイルを指定しても構いません。例えば次の例では

```
% tar cvf mytext.tar *.txt
```

現在のディレクトリにあるすべての.txtファイルがmytext.tarというアーカイブファイルに保存されました。さらに

```
% compress mytext.tar
```

とやりますとmytext.tarが圧縮されてmytext.tar.Zというファイルになります。よそに持っていったりするときにはこの形式でもっていくのがもっともコンパクトになります。もとに戻すには

```
% uncompress mytext.tar.Z
```

これでもとのmytext.tarになりました。

3.2.4.ファイルのリンク

VMSでは論理名(ロジカルネーム)を使ってファイルやディレクトリを参照することがよくあります。論理名の示す内容を変えることによって同じプログラムから別のファイルを処理できるようになります。環境などに依存する一連のファイルの中から現在の状況にあったファイルを選んで使う場合など論理名を与えておいて論理名で参照したほうが便利です。プログラムを書き直したり、データとして与えたりする必要がないからです。UNIXで同様のことをするにはシンボリックリンクという方法を使います。

シンボリックリンクはファイルとして作られます。lnというコマンドを使います。

```
% ln -s run124.dat input.dat
% ls -l input.dat
lrwxrwxrwx  iwashi  10 Sep  9 10:23 input.dat -> run124.dat
% ls -F input.dat
input.dat@
%
```

一行目でシンボリックリンクを張りました。-sは必ず付けるようにしてください。ないとハードリンクと言って本物のファイルと同じ扱いになってしまいます。リンクが張られたかどうかをlsコマンドで確かめています。-Fオプションで@がついたものがシンボリックリンクです。run124.datというファイルは実在しなくてもリンクは張れま

す。

　リンクを終わるときは`rm`を使います。リンクもファイルですから`cp`や`mv`もできます。注意すべきことは`cp`の時はリンクが指している先のファイルがコピーされますが、`mv`の時はリンクそのものが改名されます。

　シンボリックリンクはUNIXのシステムファイルでは多用されています。例えば`ls -F`で`/usr/lib`などを見てみましょう。`@`のついたファイルがたくさんあります。