tcshによるコマンドライン補完

池田, 大輔
九州大学大型計算機センター研究開発部

https://doi.org/10.15017/1470344

出版情報：九州大学大型計算機センター広報. 32 (1), pp.43-49, 1999-03. 九州大学大型計算機センター
バージョン：
権利関係：

tcshによるコマンドライン補完

池田 大輔*

tcshはUNIX上で稼働するシェルの一つであり，非常に強力な補完機能が備わっています．補完機能とは，コマンドラインにおいて，入力中に[TAB]キー[1]を押すと，tcshが残りの入力を補う機能です．この機能を使うことにより，コマンドラインでの入力が非常に楽になります．

1節では，tcshに標準で備わっている補完機能を説明します．この節の最後では補完機能に関連したシェル変数についても説明します．次に2節で，tcshの内部コマンドであるcompleteを使ったより高度な補完機能の使い方を説明します．3節では，Fortranコンパイラfrtを使って，具体的なcompleteコマンドの使い方を説明します．最後に九州大学大型計算機センターのマシンでtcshを利用する方法を説明します．

UNIXの基本的な使い方は知っているものとします．知らない方は「大型計算機センターでのUNIX入門」(九州大学大型計算機センター広報 Vol. 31, No. 2, 1998),「続・大型計算機センターでのUNIX入門」(同 Vol. 31, No. 3, 1998) などを参照してください．

これ以降，tcshが起動されているとして説明を進めます．また，"%"によりtcshのプロンプトを表すことにします．

1 基本的な補完機能

最初に簡単な例で，補完機能の説明をします．まず，カレントディレクトリ以下のディレクトリ構成が

```
% ls -CF
X11R6/          lib/            obj/            src/
bin/            libdata/        packages/       tmp/
compat/         libexec/        ports/
games/          local/          sbin/
include/        mdec/           share/
```

となっているとします．ここで，%ls localと入力したいとします．loで始まるディレクトリはlocalしかないので，loまで入力してから，[C-i]を押すと，

```
% ls lo[C-i]
% ls local
```

*九州大学大型計算機センター研究開発部 Email: daisuke@cc.kyushu-u.ac.jp

[1][TAB]は[C-i](Ctrlキーを押しながら"i"を押す)で代用できます．これ以降では，[C-i]を用います．

のように，自動的にtcshが補完してくれます．lで始まるディレクトリは複数あるので，lだけ入力しただけでは，これ以上の補完は行ないません．ここで，C-dを押せば，そのときの補完の対象を

```
% ls l[C-d]
lib/     libdata/ libexec/ local/
```

のように表示します．ただし，C-dをコマンドラインの途中で押した場合にはカーソルのある文字を削除します．例えば，

```
% ls local
```

と入力して，カーソルがlsの後にあったとします．ここでC-dを押すと，

```
% lslocal
```

と，カーソルのあった場所の文字が削除されます．

補完の対象は，コマンド名，ファイル名，シェル変数と環境変数です．カーソルが行頭にある場合には，tcshはコマンド名を補完しようと試みます．ドル記号($)のあとであればシェル変数か環境変数を補完しようとします．環境変数は，定義済みのものだけが補完対象ですが，シェル変数は，tcshが理解するすべてのものが対象となります．それ以外の場所では，ファイル名を補完しようとします．

具体例で見てみましょう．以下のように，LaTeXのソースファイルがあるディレクトリで作業をしているとします．

```
% ls
all.aux  all.lof  all.lot  all.tex
all.dvi  all.log  all.ps   all.toc
```

ここで，ファイルall.texに対しコマンドlatexを実行します[2]．

```
% lat[C-i]                    #コマンド名補完
% latex                       #latexが補完された
% latex␣[C-i]                 #ファイル名補完
% latex all.                  #途中まで成功.
```

まず，コマンド名の補完を行なっています．この場合，latで始まるコマンドがlatexしかなかったので，補完が成功しました．さらに，スペースを追加したあと，ファイル名の補完を行なっています．C-iを押した場所からファイル名が補完の対象となります．しかし，拡張子を除いてすべて同じファイル名なのでall.までしか補完しません．

次に環境変数の補完を行ないます．

[2]#以降はコメントです．

```
  % printenv $TE[C-d]                          #TEで始まる定義済みの環境変数の表示
TERM      TEXINPUTS
  % printenv $TEX[C-i]                                    #環境変数補完
  % printenv $TEXINPUTS
```

TEで始まる環境変数は複数ありますが，TEXで始まるものは一つしかないので，ここまで入力すると補完が成功します．

1.1 シェル変数

tcshによる補完の挙動を変えるシェル変数について説明します．説明するシェル変数は，fignore, autolist，completeの3つです．

シェル変数fignoreには，補完の対象としたくないファイルのパターンを指定します．例えば，emacsが作るバックアップファイルとオブジェクトファイルを補完の対象から除きたいとします．バックアップファイルのファイル名は~で，オブジェクトファイルのファイル名は.oで終ります．

```
  % set fignore=('.o' '~')
  % ls
tmp.f   tmp.f~  tmp.o
  % ls t[C-i]
  % ls tmp.f                                  #tmp.f~とtmp.oは無視される
```

上の例では，fignoreの各要素をシングルクオートで囲んでいるのは，"~"がホームディレクトリと解釈されないようにするためです．ただし，[C-d]による候補一覧表示とfignoreで指定したパターンは無関係です．

シェル変数autolistをセットすると，補完が失敗した時点で，自動的にその時の補完候補を表示してくれます．

```
  % set autolist
  % ls
UNIX.aux        UNIX.log        UNIX.tex
UNIX.dvi        UNIX.ps         UNIX.tex~
  % ls U[C-i]
  % ls UNIX.
UNIX.aux   UNIX.dvi   UNIX.log   UNIX.ps    UNIX.tex   UNIX.tex~
```

tcshはUNIX.まで補完しましたが，それ以降は候補が複数あるために補完でず，自動的に補完の候補を表示します．

シェル変数completeには，値として "enhance" を設定しておくと[3]，

[3] completeに，他に設定できる値はないようです．

- 大文字・小文字を区別しない
- ピリオド (.)，ハイフン (-)，アンダーバー (_) を単語の区切りとして扱い，かつ，ハイフンとアンダーバーを区別しない

ようになります．単語の区切りが増えるので，ファイル名補完の時に，ピリオドの前と後を同時に補完することが可能になります．例えば，

```
% set complete=enhance
% ls
UNIX.aux        UNIX.log        UNIX.tex
UNIX.dvi        UNIX.ps         UNIX.tex~
% ls U.d[C-i]
% ls UNIX.dvi
```

のようなことができます．これで，U.d がUNIX.dvi へ補完されました．

2　complete コマンド

1 節で説明した補完機能は，シェル変数で挙動を変える以外は，特別な設定は必要なく，tcsh に標準で備わっている機能でした．これだけでも充分便利ですが，一般にコマンドとその引数の間にはある程度のルールがあることを利用し，補完対象を狭めるのが，complete というtcsh の内部コマンドです．

例えばlatex コマンドの対象となるのは，拡張子が “tex” という名前のファイルです．この規則を，

```
% complete latex p/1/f:\*.tex/
```

として，tcsh に教えます．これは，「latex というコマンドの 1 番目の引数 (p/1/) にはテキストファイル (f) で，拡張子が “tex” である任意のファイル (*.tex) がきます」という意味です．“p/1/” は，引数の位置 (Position) が 1 番目ということです．ファイルの種類やパターンを指定する時には，コロン (:) 以降は省略しても構いません．アスタリスク (*) の前のバックスラッシュ (\) は任意のファイル名と解釈されるのを防ぎます．設定ファイル中に書く場合は，バックスラッシュは特に必要ありません．これ以降，latex の引数としては*.tex のみが補完の対象となります．

拡張子が “tex” であるファイルを処理対象とするプログラムがtex, platex と他にもある場合は，同じ規則を適用するコマンドを以下のようにまとめることができます．また，拡張子が複数の場合にも同様にまとめることができます．具体的には，

```
% complete {tex, latex, platex} p/1/f:\*.tex/
% complete gs p/1/f:\*.{ps, eps}/
```

のようにします．

限定子	意味
a	エイリアス
c	コマンド
d	ディレクトリ
e	環境変数
f	ファイル名
s	シェル変数
t	テキストファイル
u	ユーザ名

表 1: 補完対象の限定子

補完対象を限定するもの (上の例では f:*.tex の f) は，表 1 であげたものなどがあります．ただし，実行可能ファイルを表すものはないようです．

補完の場所を表わすパターン (さきほどの例では p/1/) は，p/*n*/, c/*word*/, n/*word*/が使えます．

"p/*n*/" は，*n* 番目の引数の位置 (Position) で補完を行ない，*n* として数字以外にもアスタリスク (*) が使えます．アスタリスクは任意の場所を表します．一般に "p/*n*/" は，引数の順序が決まっている場合に使います．例えば，cd は引数としてディレクトリを一つとり，オプションは指定しません．よって，補完のルールは「一番目の引数ではディレクトリを補完する」となります．setenv も，引数として環境変数のみをとるので，同様の設定が可能です．

```
% complete cd p/1/d/
% complete setenv p/1/e/
```

d と e は，それぞれディレクトリと環境変数を表します (表 1 参照)．

"c/*word*/" は，現在 (Current) のカーソル位置で補完を行ない，補完対象となるのは *word* で始まるものです．*word* として，任意の語を表す "*" を指定することもできます．ここでは，lpr コマンドでプリンタ名を補完することを考えます．lpr は，オプション-P の直後に空白をいれずにプリンタ名を指定します．他のオプションも指定する可能性があるので，先程の "p/*n*/" による場所の指定はできません．よって，補完のルールは「-P という文字列がきたら，それに続けてプリンタ名を補完する．それ以外はファイル名を補完する」になります．これは以下のように書けます．

```
% complete lpr c/-P/"(ps kkprint tower)"/          #プリンタ名が補完対象
% lpr -Pk[C-i]                                     #k で始まるプリンタ名の補完
% lpr -Pkkprint
```

これにより 3 つのプリンタ名が補完できるようになります[4]．

プリンタ名は他のコマンド (lpq, lprm など) で使用する可能性があります．このように何度も使用する場合には，

[4]単なるファイル名補完はデフォルトの機能なので，特に complete で指定する必要はありません．

```
% set printers=(ps kkprint tower)
% complete lpr c/-P/\$printers/
```

と，変数を用います．変数を参照するには，変数名にドル記号 ($) をつけます．ドル記号の前のバックスラッシュは，設定ファイル中でも必要です．

"n/*word*/" は，現在のカーソル位置の次 (Next) の位置で補完を行ない，補完対象となるのは *word* で始まるものです．*word* として，任意の語を表すアスタリスク (*) を指定することもできます．"n/*word*/" は，オプションと補完したいものとの間に空白記号がある場合によく用いられます．例えば，`lp` コマンドのプリンタ名指定は`-d␣printer` のように，空白がはいります．よって，補完のルールは「`-d` がきたら，次にプリンタ名を補完する」となるので，以下のようにします．

```
% set printers=(ps kkprint tower)
% complete lp  n/-d/\$printers/
```

この例では，"n/-d/" のかわりに "c/-d/" とすることはできません．次も "n/*word*/" の例です[5]．

```
% complete dvips n/-t/"(landscape seascape)"/ \
?n/-o/f:\*.ps/ n/\*/f:\*.dvi/
```

これは「`dvips` コマンドの時には，`-t` という文字列の次には，用紙の方向を示す文字列を補完し，`-o` という文字列の次には，PS ファイルを補完し，任意の位置で拡張子が "`dvi`" であるファイルを補完する」という意味です．任意の位置 "n/*/" という指定は，デフォルトで補完したいものを指定する時に用います．

3　Fortran コンパイラ `frt` を使った例

この節では，大型計算機センターでの利用が多いと思われる，Fortran コンパイラ`frt` に対する`complete` コマンドの使い方の例を示します．

Fortran のソースファイルの拡張子は，"`f`" または "`f90`" なので

```
% complete frt n/\*/f:\*.{f,f90}/
```

とします．オプションをいくつ指定するかは事前に分かっていないので，任意の場所を表す "n/*/" を指定しています．このままでは，任意の場所で拡張子が "`f`" または "`f90`" であるようなファイルしか補完されません．よって，"`-o`" で実行ファイルを指定する時や，"`-Z`" でログファイルを指定するときにも，Fortran のソースファイルが補完されてしまうという問題があります．そこで，これらのオプションを入力した次には，任意のファイルが補完できるようにします．

[5] みやすいようにバックスラッシュで行を継続しています．"?" が継続行のプロンプトです．

```
% complete frt n/-o/f/ n/-Z/f/ n/\*/f:\*.{f,f90}/
```

他にもファイル名と共に指定するオプションが多くある場合は，個別に指定するのは面倒なので，

```
% complete frt n/-/f/ n/\*/f:\*.{f,f90}/
```

とすれば，“-” でオプション指定した次は任意のファイル名を補完できます．

使うライブラリがある程度決まっていれば，それらを補完するのも便利でしょう．ライブラリのリンクは “-l” のあとに空白なしで，ライブラリ名を指定するので，

```
% complete frt c/-l/"(ssl2vp numpac)"/ n/\*/f:\*.{f,f90}/
```

と書いておけば，SSL II ライブラリや NUMPAC ライブラリの指定が楽になります．

次にオプションそのものの補完ですが，frt コマンドのオプションは，“-Free” と “-Fixed” を除いて，“-c” や “-Ob” のように 1 文字または 2 文字のアルファベットで指定します．そこで，“-F” まで入力したら，その後を補完するようにします．

```
% complete frt c/-F/"(ree ixed)"/ n/\*/f:\*.{f,f90}/
```

実際には，“-F” の後の 1 文字を指定する必要があります．

4 センターでのtcshの利用

九州大学大型計算機センターのマシンでは，kyu-cc, kyu-vpp, wisdom において，tcsh が利用可能です．これらのマシンにログインし，

```
kyu-cc%tcsh
>                                                    #tcshが起動された
```

と入力すれば，tcsh が利用できます[6]．これらのマシンのデフォルトのプロンプトは “>” です．

これらのマシンにおいては，デフォルトではcshがログインシェルですが，tcsh をログインシェルとして使用することも可能です．ログインシェルにtcshを利用する場合は，以下の内容をログインシェルを変更したいマシンの~/.utmsrc に保存してください．

```
SHSEP:ON
SHELL:/usr/local/bin/tcsh
```

これにより，次回のログインからtcshがログインシェルとなります．

[6]tcsh の絶対パスは/usr/local/bin/tcsh です．コマンドサーチパスにない場合は，パスに追加するか絶対パスで指定してください．