11. 標準入出力の利用 — リダイレクトとパイプ

2013 年 7 月 4 日

目 次

1 シェル

1.1 シェル

シェル (shell) は，ユーザーがコマンド行に打ち込んだコマンド等を解釈し，コンピュータに実行させる役割をもつプログラムです[1]。ユーザーがコンピュータにログインすると，そのユーザーのためにシェルが動き出します。これをログインシェル (login shell) といいます。ユーザーがコマンド行にコマンドをタイプして実行できるのは，シェルが動いているからです。

シェルには幾つかの種類があり，ユーザーがログインシェルとして何というシェルを用いるかは，予め管理者が設定しています。シェルが持つ機能や使い方はシェル毎に違いますので，`echo $SHELL` を実行して，利用中のシェルを確認しましょう[2]。/bin/tcsh と出力されますね。皆さんが使っているシェルは tcsh (ティーシーシェル) です。

1.2 コマンド入力の支援機能

tcsh が持つ機能の一例として，コマンド入力を助ける機能を幾つか紹介します。なお，ここで紹介する機能は，他の最近よく使われるシェルでも利用できます。

1.2.1 ヒストリの一覧と利用

過去に実行したコマンド行の履歴をヒストリ (history) といいます。Windows のコマンドプロンプトと同様に，tcsh でも上向きや下向きの矢印キーを使えば，過去に実行したコマンド行を再度使うことができます。

次のコマンドはヒストリ一覧を表示します。

```
history
```

ヒストリ一覧から，過去に実行したコマンド行を選んで再実行するには

`!`n

に続いて `<ENTER>` を押します[3]。ここで n はヒストリ一覧中のヒストリ番号です。

1.2.2 コマンド行の補完

コマンド名やファイル名の一部をタイプしてから `<TAB>` を押すことで，残りを自動的に補う (補完) ことができます。この機能を使うとコマンドやファイル名のタイプミスを防ぐこともできます。

1. [コマンド名の補完] コマンド行に `his` とタイプして `<TAB>` を押してみましょう。続いて `<ENTER>` を押しましょう。

[1] これは，シェル利用の一形態です。他の形態 (シェルスクリプトのためのコマンドインタプリタ) でシェルを利用することもあります。

[2] echo は引数をそのまま出力するコマンドですが，コマンド行における $ はシェルにとって特別な意味があるため，画面に $SHELL とは表示されません。

[3] 知っていて便利なヒストリ機能には次のものもあります。`!-`n (n だけ前に実行したコマンド行), `!!` (一つ前に実行したコマンド行 (`!-1` と同じ)), `!`*str* (*str* で始まる最も最近のコマンド行)。ここで n には正の整数を，*str* には適当な文字列 (通常はコマンド名またはその始めの一部) を記述します。これらに続いて `<ENTER>` を押すと，当該コマンド行の内容が実行されます。

2. [補完候補の表示] コマンド行に `hi` と打ってから `<TAB>` を押してみましょう。

   hi で始まるコマンドは複数ありますので，まだ補完はされません。

3. 続いて s をタイプしてコマンド行を `his` としてから `<TAB>` を押してみましょう。続いて `<ENTER>` を押しましょう。

4. [ファイル名の補完] まず，カレントディレクトリに存在するファイルの名前を ls コマンドを実行して確認してください。続いて，`cat` の引数に，既存のファイル名の一部をタイプしてから `<TAB>` を押してみましょう。ファイル名が補完されたら `<ENTER>` を押しましょう。

2 リダイレクトとパイプ

2.1 復習 — コマンド出力の保存

2.1.1 コマンド出力のファイルへの保存

コマンド *command* が画面に表示する内容を，ファイル名が *file* であるファイルに格納するには

command [*argument* ...] > *file*

とします。ここで [*argument* ...] は，*command* に引数を与えたければ [*argument* ...] の箇所に記述できることを意味します。*file* として，存在しないファイルを指定すれば，*file* が自動的に作成されます。*file* が既存のファイルならば，その内容は *command* の出力で上書きされます[4]。

例 `cal 2013 > year` を実行してから，year の内容を確認してください。

2.1.2 コマンド出力の既存ファイルへの追加

既に存在するファイルにコマンド出力を追加したい場合には > の代わりに >> を使い，

command [*argument* ...] >> *file*

とします。

例 `date` を実行してから `date >> year` を実行し，ファイル year の内容を確認してください。続いて `date > year` を実行し，year の内容を確認してください。再度 `cal 2013 > year` を実行してください。

2.2 標準出力・標準エラー出力とリダイレクト

2.2.1 標準出力と標準エラー出力

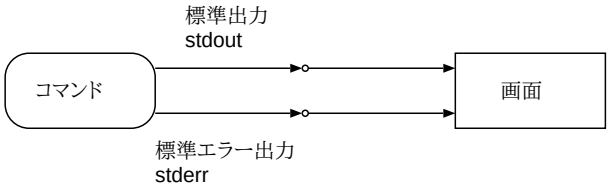
ここでは，コマンドの実行結果が画面に表示される仕組みをより正確に説明します。
多くの UNIX コマンドは “標準出力” と “標準エラー出力” と呼ばれる情報の出口を持っています。

1. **標準出力 stdout** (standard output): コマンドの実行結果が出力される

[4] `set noclobber` を実行することで，> による既存ファイルの上書きを禁じる設定にすることができます。その場合，> の代わりに `>!` を使えば，強制的に既存のファイルを上書きできます。

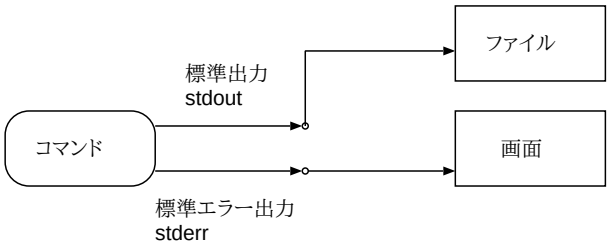
2. **標準エラー出力 stderr** (standard error): エラーメッセージ等のコマンド実行に係わる付加的な情報が出力される

コマンドは，実行結果などを画面ではなく，これらの出口に出力します。画面にコマンドの実行結果等が表示されるのは，図に示すように，通常は標準出力と標準エラー出力が画面に接続されているためです。



2.2.2 標準出力のリダイレクト

コマンドを実行する際，標準出力の接続先をファイルに変更するようシェルに対して指示することができます。このような接続先の変更操作 (切り替え) を**リダイレクト** (redirection) といいます。第 2.1 節で紹介した `>` と `>>` は，共に実行するコマンドの標準出力の接続先をファイルに切り替えるリダイレクトです。標準出力のリダイレクトを用いれば，通常は画面に表示されるコマンドの実行結果を，ファイルに保存することができます。



例えば，`cal 2013 > year` を実行して結果がファイル year に保存されるのは,「`> year`」の指定により cal コマンドの標準出力が，画面ではなくファイル year に接続されるためです。

なお，tcsh で標準出力と標準エラー出力の接続先を共にファイルに切り替えるには `>&` や `>>&` を使います。

注意 `cal 2013 > year` において,「2013」は cal コマンドに対する動作の指示なので，cal コマンドの引数です。一方,「`> year`」はシェルに対して標準出力の接続先変更を指示しているのであり，cal コマンドの動作を変更するものではありません。したがって,「`> year`」は cal コマンドの引数ではありません。

2.3 入力の終わりとコマンドの強制終了

2.3.1 キーボードからの入力と入力の終わり — CTRL-d

次の操作を試してみましょう。

1. `wc year` を実行してください。

   wc はファイルのバイト数・単語数・行数を表示するコマンドです。

2. 次に，引数無しに `wc` とだけ打って実行してみましょう。

   何も起きないどころかプロンプトも表示されません。

3. ここで，キーボードから `ls` と打って，`<ENTER>` を押しましょう。

   プロンプトが表示されないままでコマンドを打っても，そのコマンドは実行されません。

4. キーボードから `I like you.` と打ち込んで `<ENTER>` を押しましょう。

   まだ何も起こりませんね。

5. 今度は Ctrl キーを押しながら d キーを押してください。これからは CTRL キーを押しながら d キーを押すことを CTRL-d と表記します。

   wc の実行結果が出力され，プロンプトが現れました。wc は実行を終えました。

ファイル名を指定せずに wc を実行すると，wc はキーボードからの入力を処理します[5]。2. においてプロンプトが表示されなかったのは，wc が実行中であり，キーボードからの入力を待っている状態だったからです。

4. で使った CTRL-d は「**入力の終わり** (正確には，**ファイルの終り; EOF** = End Of File)」を意味するキーです。wc は CTRL-d が入力されたのでキーボードからの読み込みを終了し，実行結果を画面に出力しました[6]。

なお，上の操作ではキーボードから 2 行 4 単語を入力しましたから，行数と単語数に関しては wc は正しい結果を出力しています。一方，文字数については，画面に表示されている入力文字数 (空白文字を含む) より 2 文字分多いはずです。これは，行末に入力した `<ENTER>` を，wc が文字として数えているからです。実際，`<ENTER>` で入力したものは特殊な文字であって，通常の文字として表示される代りに，画面上では改行として表示されているのです。

2.3.2 コマンドの強制終了 — CTRL-c

CTRL-c はコマンドの実行を**強制的に終了**させる働きをします。もし，先ほどの wc の実行例で CTRL-d の代わりに CTRL-c を押すと，wc は読み込みを完了する前に終わってしまいます。したがって文字数等の出力は行われません。両者の違いに注意しましょう。

コマンドを実行したけれど動作がおかしい，とか，プロンプトが戻るまでの時間が長すぎる，という場合には CTRL-c によるコマンドの終了を試してみるといいでしょう。コマンド行でタイプミスをしてしまい，最初から打ち直したい場合にも，CTRL-c が使えます。ただし，コマンドを正常終了できる場面では，正規の終了操作をすべきです。

2.4 標準入力とリダイレクト

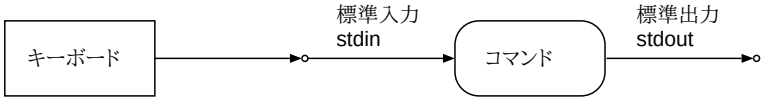
2.4.1 標準入力

wc コマンドのように，キーボードからデータを受け取って処理できる UNIX コマンドは，多くの場合，正確には **標準入力 stdin** (standard input) という入口から入力を読み込みます。標準入力は，

[5]正確に言えば，標準入力からの入力を処理します。標準入力については第 2.4 章で学習します

[6]プロンプトに対してコマンドを入力せずに CTRL-d だけを打つと，設定によってはログアウトするかもしれません。注意しましょう。

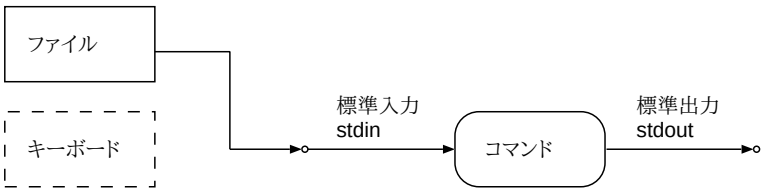
通常はキーボードに接続されているので，その結果として，コマンドはキーボードから入力を受け取るのです。



この図では，コマンドの標準エラー出力と標準出力の接続先 (画面) を省略しています。また，これ以降の図において，標準エラー出力の描画は略します。

2.4.2 標準入力のリダイレクト

標準出力の接続先を切り換えできるのと同様に，標準入力の接続先を切り換えること (リダイレクト) も可能です。



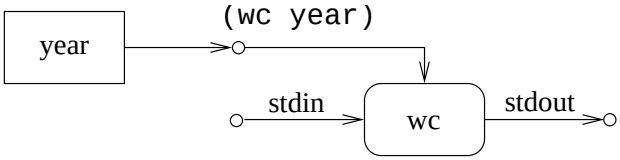
command が標準入力からの入力を受け付けるコマンドであるとき，キーボードの代わりにファイル *file* から入力を受け取るようにするには

command [*argument* ...] < *file*

を実行します。

例 `wc < year` を実行しましょう。これは wc の標準入力の接続先を，キーボードからファイル year に切り換える例です。

注意 wc の引数に既存のファイル名を指定して `wc year` を実行すると，wc は標準入力とは異なる入り口を用意して，そこから year を読み込みます。実行結果は標準入力からの読み込みである `wc < year` と同じですが，wc の動作は異なります。



一般に，ファイル名を引数にできるコマンドを，ファイル名を与えずに実行すると，データの入口として標準入力が使われることが多いです。wc コマンドを `wc` や `wc < year` と実行すれば，wc にはファイル名の引数はありませんので，入力に標準入力が使われます。

練習

1. 標準入力のリダイレクトを使って，ファイル year を wc に読み込ませましょう。ただし，今回は year の行数のみを wc に出力させてください。

   これは wc にオプション引数を指定することにより可能です。何というオプションが必要か，は `man wc` で調べてください。man コマンドを終了するには `q` を打つのでしたね。

2.4.3 標準入出力の同時リダイレクト

コマンドの標準入力と標準出力を同時に切り換えることもできます。例えば，標準入力を year に，標準出力を wc.out に切り換えて wc コマンドを実行するには

```
wc < year > wc.out
```

とします。これを試したら，不要な wc.out は削除しましょう。

2.5 パイプ

2.5.1 準備 — 一時ファイルを使った複数コマンド処理

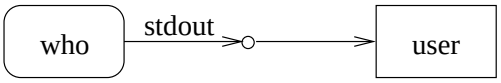
who コマンドを使うと，同じコンピュータを使っているユーザーの一覧が得られます。who と wc を使ってコンピュータにログインしているユーザーの人数を調べてみましょう。

1. まず，who コマンドを引数なしで実行し，どんな出力が得られるかを確認してください。

   一行につき 1 ユーザーの情報が出力されますね。who の出力から行数を求めれば，ログインしている延べユーザー数がわかります。

2. who コマンドの出力を，リダイレクトを使ってファイル user に保存します。

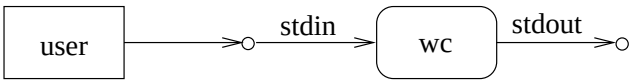
   ```
   who > user
   ```

   cat コマンド等で user の中身を確認してください。

3. wc コマンドにオプション -l (エル) を指定して，ファイル user の行数を調べます。ここでは wc にファイル名の引数を与えずに，wc の標準入力から user を読み込むこととします。

   ```
   wc -l < user
   ```

   ユーザー数がわかりましたね。

4. ファイル user は不要になりましたので，削除してください。

   ```
   rm user
   ```

2.5.2 コマンドの連結 — パイプ

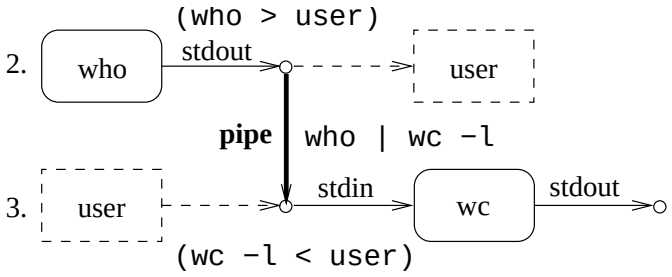
あるコマンドの標準出力を，ファイルではなく，別のコマンドの標準入力に直接接続することが可能です。その操作を**パイプ**と呼びます。パイプは，コマンド行において，一連のコマンドを「縦棒 (|)」で区切って記述することによって行います。パイプによって連結された一連のコマンドを**パイプライン** (pipeline) と言います。

なお，コマンドの標準出力と標準エラー出力の両方を，続くコマンドの標準入力に接続するには，tcsh の場合，| の代わりに |& とします。

例 1: who と wc でログイン中のユーザー数を得る　前節の操作をパイプを使って行います。

```
who | wc -l
```

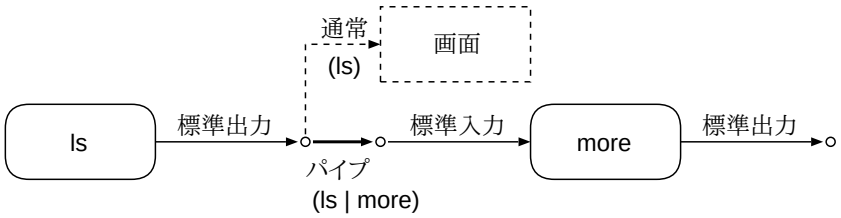
この状況を図示したのが次の図です。wc は，ファイル名の引数が無いときに，標準入力 (stdin) から読み込みをするのでしたね。パイプと共に wc を実行する場合，wc は標準入力 (stdin) から入力を受け取る必要がありますので，wc の引数にファイル名を指定してはいけないことに注意しましょう。



例 2: ls の出力を more で見る　more はページャ (pager) という種類のコマンドの一つで，画面に収まらないファイルの内容を 1 ページずつ見るために使われます[7]。ここでは，/etc ディレクトリ内の全ファイルの詳細情報を ls コマンドで出力し，これをパイプと more で表示します。

```
ls -l /etc | more
```

more コマンドは，`more year` のように引数にファイル名が与えられれば，そのファイル用の入り口を作って読み込みますが，ファイル名を与えずにリダイレクトやパイプと共に用いると，標準入力から入力を受け取ります。次の図では ls の引数 (–l /etc) の記述は省略しています。



動作の概要は以下のとおりです。

1. ls が実行結果を標準出力に出力します。

   ls の標準出力をパイプによって more の標準入力に接続しているので，ls の実行結果は画面には出力されません。

2. more が ls の実行結果を標準入力から受け取り，標準出力を通じて画面に表示します。

   (a) more は一画面分以上の入力を受け取ると対話的な動作をしますので，一画面分の表示を終えたところで表示を止め，ユーザーからの命令を待っている状態になります。

   (b) ユーザーがスペースキー等を押して全ページを表示させ終えるか，q 等を押すことで more を終了させると，パイプライン全体が終了し，プロンプトが表示されます。

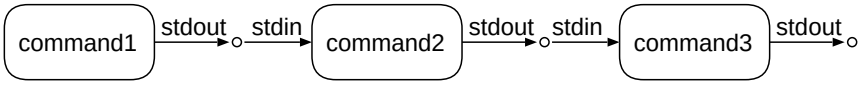
[7] more コマンドを実行すると，実際には more の機能を拡張した less というプログラムが動くシステムもあります。

2.5.3 コマンドの多段連結

command1 が標準出力 (stdout) に結果を出力するコマンドであり，*command2* と *command3* が標準入力 (stdin) から入力を読み取って結果を標準出力 (stdout) に出力するコマンドならば，

command1 | *command2* | *command3*

により，次の図に示す多段のパイプが可能です。これにより，*command1* の出力が *command2* で処理され，さらに *command3* で処理されます。



なお，これまでと同様に，パイプラインの中の各コマンドには引数を与えることもできます。また，4つ以上のコマンドをパイプすることもできますし，その出力をリダイレクトでファイルに保存することも可能です。

command2 や *command3* のように，標準入力と標準出力を使った入出力が可能なコマンドを**フィルタ** (filter) といいます。フィルタはパイプラインの中で利用できるコマンドです。これまでに紹介したコマンドでは cat と wc はフィルタです。

3 問題 1

1. echo コマンドは引数に与えた文字列を標準出力に出力するコマンドです。echo とリダイレクトを使って，内容が hakodate であるファイル campus を作ってください。

2. echo コマンドを複数回実行して，ファイル campus の内容を

   hakodate
   sapporo
   asahikawa
   kushiro
   iwamizawa

   にしてください。

3. tr は文字の置換を行うコマンドです。tr は標準入力から入力を読み込み，引数での指定に従って文字を置換し，結果を標準出力に出力します。

   (a) tr コマンドの動作を確認するために，`tr a A` を実行してから

   ```
   hakodate
   sapporo
   ```

   を標準入力（キーボード）から入力してください。入力を終えたら tr コマンドを正常終了させてください。

   (b) tr では文字集合の置換もできます。`tr a-z A-Z` を実行してから

   ```
   hakodate
   sapporo
   ```

を標準入力 (キーボード）から入力してみましょう。

では，ファイル campus の内容をすべて大文字にしたものを，ファイル CAMPUS に保存しましょう。tr コマンド，および標準入力と標準出力のリダイレクトを使ってください。

4. sort コマンドは，入力を行毎にソート (並べ替え) して標準出力に出力するコマンドです。sort *file...* の形式で実行すると，sort は *file...* の内容を直接読み込みますが，*file...* を略すと標準入力から読み込みます。

   (a) まず，`sort campus` を試してください。

   これは，sort がファイル campus を直接読み込んで処理する例です。

   (b) 引数無しで sort コマンドを実行し，標準入力（キーボード）から

   hak
   sap
   asa

   と入力してください。sort コマンドを正常に終了させて，出力を観察してください。

   (c) sort コマンドと標準入力のリダイレクトを使って，ファイル campus の内容を並べ替えて画面に表示しましょう。

   (d) 標準入力と標準出力のリダイレクトを同時に使って，ファイル campus の中身を並べ替えたものを，ファイル campus_s に保存しましょう。

5. カレントディレクトリに存在するファイル（. で始まるものは除く）の個数を求めましょう。

   (a) まず，ls の出力を一度ファイルに保存してから，中身を確認してください。そのファイルの名前を引数として wc コマンドを実行し，期待する結果が得られるかを確認してください。

   (b) パイプを使って上記と同様のことをしてください。

   (c) (a) で作成したファイルを削除してください。

6. grep は，引数で与えた文字列パターンを含む行のみを抽出して，標準出力に出力するコマンドです。

   `grep` *pattern file...*

   の書式で実行すると，grep は *file...* から *patten* を含む行のみを抽出しますが，*file...* を省略すると，標準入力から入力を読み込みますので，フィルタとして利用できます。

   まず，grep ko campus や grep sa campus を実行して，grep の動作を確認してください。

   次に who の出力の中から自分の情報のみを出力しましょう。grep とパイプを使います。

7. cat コマンドは，引数にファイル名を指定せずに実行すると，標準入力から入力を読み込みこんでそのまま出力します。オプション -n をつけると行番号をつけて出力します。

   さて，who コマンドの出力をを辞書順に並べて表示しましょう。さらに，その結果に行番号を付けて出力しましょう。

4 問題2

1. cat コマンドの使い方をもう少し詳しく見てみましょう。

   cat コマンドは，引数にファイル名が与えられると，そのファイルを読み込んで，内容を標準出力に出力します。一方，引数にファイル名が与えられなければ，標準入力から読み込みこんだものを，そのまま標準出力に出力します。このことに注意して，次の操作を行ってください。

   (a) 引数を何も与えずに cat コマンドを実行してください。cat コマンドが標準入力つまりキーボードからの入力を待っている状態になります。

   (b) ここで，キーボードから hakodate と打ち込んで <ENTER> を押しましょう。すると，今打ち込んだ hakodate の次の行に，もう 一行 hakodate が表示されます。これは cat コマンドが標準出力を通じて画面に出力したものです。

   (c) さらにキーボードから sapporo と打ち込んで <ENTER> を押し， 結果を観察してください。

   (d) cat コマンドへの入力を CTRL-d で終了させてください。cat の実行が正常に終了してプロンプトが現れます。

   (e) さて，cat コマンドを用いて，中身が

   ```
   hakodate
   ```

   であるファイルを作成してください。ファイル名は mycampus とします。

2. cat コマンドは，引数にファイル名を与えて実行すると，そのファイルの内容を標準出力に出力します。cat の標準出力をリダイレクトすることによって mycampus と同じ内容を持つファイル mycampus2 を作りましょう。

3. cat コマンドは，引数に複数のファイル名を与えて実行すると，与えられたファイルの内容を連結 (conCATenate) して，標準出力に出力します。

   (a) まず，このことを確かめるために

   ```
   cat campus mycampus
   ```

   を実行してみましょう。

   (b) 次に，ファイル campus と mycampus の内容を連結したファイル campus_mycampus を作りましょう。

4. uniq コマンドは，連続した重複行から重複を取り除くためのコマンドです。uniq file の形式で実行すると，uniq は file の内容を直接読み込みますが，file を略すと標準入力から読み込みます。

   (a) まず，sort コマンドを使ってファイル campus_mycampus の内容を並べ替え，それをファイル campus_mycampus_s に入れてください。

   (b) uniq コマンドの標準入力に，リダイレクトを使ってファイル campus_mycampus_s の内容を与え，uniq の動作を確認してください。次にファイル campus_mycampus を与えた場合との違いを確認してください。

   (c) sort と uniq コマンドのパイプラインを使って campus_mycampus の内容を並べ替えるとともに，重複行を取り除いて出力してください。