

# 情報科学演習／データベース入門 A 資料 5

## 文字列のマッチングとファイルのアクセス権

平成 29 年 5 月 8 日

### 目次

<b>1</b>	<b>コマンド行における特殊文字の扱い</b>	<b>1</b>
1.1	メタキャラクタ	1
1.1.1	？と＊	1
1.1.2	その他のメタキャラクタ	3
1.2	特殊文字の無効化	3
1.2.1	例	4
1.3	練習問題	4
<b>2</b>	<b>正規表現 — 文字列検索における検索パターン指定</b>	<b>5</b>
2.1	練習問題	5
<b>3</b>	<b>ファイルの詳細情報とアクセス権</b>	<b>6</b>
3.1	コマンドの実体とパス名を用いたコマンド実行	6
3.1.1	練習	6
3.2	ユーザーとグループ	7
3.3	ファイルの詳細情報	7
3.4	ファイルのアクセス権	8
3.4.1	アクセス権の読み方	8
3.4.2	アクセス権の変更 — chmod コマンド	9
3.4.3	発展 — アクセス権の 8 進数指定 (絶対指定)	10
3.5	特権ユーザーとセキュリティ	11
3.6	問題	11

## 1 コマンド行における特殊文字の扱い

コマンドを入力する行(コマンド行)の内容を解釈して、コマンドを実行するのはシェルという種類のプログラムでした。ここでは、コマンドに引数を与えて実行する際に必要となる、シェルにとって特別な意味のある文字の取り扱いを学びます。また、検索等でも必要となる、文字列のマッチングに関する考え方の基礎を習得します。

### 1.1 メタキャラクタ

カレントディレクトリに、仮に a1, a2, ..., a5 という 5 つの通常のファイルと distdir というディレクトリのみが存在するとし、a1, ..., a5 を distdir に移動したいとします。ファイル移動のコマンドは mv であり、mv の引数には複数のファイルを指定できますから、

```
mv a1 a2 a3 a4 a5 distdir
```

を実行すれば希望は叶えられますが、移動するファイルをすべて列挙するのは面倒です。

この例のように、一度のコマンド操作で複数のファイルを処理したいとき、メタキャラクタ (*meta-character*) という、シェルにとって特別な意味のある文字を利用すると便利です。メタキャラクタを用いると、上記の操作は

```
mv a? distdir
```

や

```
mv a* distdir
```

で済みます。a? と a\* は、シェルによって共に a1 a2 a3 a4 a5 に展開 (置き換え) されてから、引数として mv コマンドに渡されます。

#### 1.1.1 ? と \*

? と \* は、一言でいえば、

? : 任意の 1 文字

\* : 任意の文字列

にマッチする<sup>1</sup>メタキャラクタです。その具体的な使い方を次の例で見えます。

##### 1. 準備

- (a) カレントディレクトリを /pub/eis/metawork に変更してください。うまくいったかどうか、pwd コマンドで確認してください。
- (b) ls コマンドを使ってカレントディレクトリに存在するファイルを確認してください。次のファイルがあります。

```
A A1 B C a a1 a11 a12 a13 a2 a3 b b1 b11
```

<sup>1</sup>後に説明しますが、? と \* とともに、ファイル名先頭の. (ドット) や / にはマッチしません。

## 2. ? の使い方

echo コマンドは、引数に与えた文字列をそのまま表示するコマンドでした。例えば、

```
echo abc d e
```

を実行すると、画面に abc d e と表示されます。この echo コマンドを使い、? を含む引数がどのようなファイル名に展開されるのかを調べましょう。

(a) 次のコマンドを順番に実行してください。

```
echo ?
echo ??
echo ???
echo ????
```

echo の引数に ? を与えても、? とは表示されません。

シェルは ? をカレントディレクトリに存在する 1 文字からなるファイル名のリストに展開し、引数として echo コマンドに渡します。その結果として 1 文字からなるファイル名がすべて表示されます。

同様に、?? は 2 文字の、??? は 3 文字のファイル名に展開されます。4 文字のファイル名をもつファイルはカレントディレクトリに存在しないので、最後の例では、その旨のメッセージが出ます。

(b) ? を他の文字と組み合わせて使うこともできます。

```
echo a?
echo ?1?
```

最初の例では、ファイル名の先頭が a であって、その後に何という文字でも構わないから、1 文字が続くものに展開されます。その次の例は、1 の前後に任意の 1 文字が存在するファイル名に展開されます。

(c) ? はパス名の一部として使うことができます。例えば、

```
echo /bin/????
```

だと、/bin というディレクトリに存在するファイルのうち、4 文字のファイル名をもつものが該当します。

なお、? はファイル名の先頭にある . や、パス名の区切り記号 / にはマッチしません。

## 3. \* の使い方

\* は ? に似ていますが、1 文字のみではなく、任意の長さの文字列にマッチする点が異なります。

次の例は、各々 a と a1 で始まるファイル名に展開されます。

```
echo a*
echo a1*
```

最初の例で注意すべきは、展開されたファイル名の中に a が含まれていることです。すなわち、任意の長さには、0 文字の長さも含みます。同じ理由から、2 番目の例でも、a1 が結果に含まれます。

次の結果も、容易に予想できるでしょう。

```
echo *3
echo *
echo /bin/*
```

なお、? の場合と同様に、\* はファイル名の先頭にある . や、パス名の区切り記号 / にはマッチしませんので、これらは明示的に記述する必要があります。例えば、カレントディレクトリに存在する . で始まるファイル名をすべて扱うには .\* とする必要があります。

以上、? や \* がどのようなファイル名に展開されるのかを調べるために echo コマンドのみを使ってきましたが、メタキャラクタはシェルが解釈する特殊文字なので、コマンドの引数一般に利用することができます<sup>2</sup>。

例えば、カレントディレクトリに存在する a で始まるファイルを調べたければ、ls a\* を実行するのが自然でしょう。なお、ls の引数にファイル名を与えると、そのファイルの情報のみが表示されます。

### 1.1.2 その他のメタキャラクタ

ファイル名やディレクトリ名に展開される、\* や ? 以外のメタキャラクタとして、次のものもあります。

```
[c1c2...]      [ ] 内の 1 文字 ([c1-c2] を用いて文字範囲を指定することも可能)
{string1,string2,...}  { } 内の文字列
```

これらは次のように使います。

```
echo [abc]
ls [abc]
echo [A-Z]*
echo /{bin,usr,var}
ls /{bin,usr,var}
```

[abc] は a, b, c のいずれかのファイル名に、[A-Z]\* は大文字で始まるファイル名に展開されます。/{bin,usr,var} は /bin /usr /var に展開されます。

また、~ は自分のホームディレクトリの絶対パス名に展開される特殊文字であり、ほとんどのシェルで利用できます。次のように使えます。

```
echo ~
ls ~
echo ~/a*
```

## 1.2 特殊文字の無効化

echo コマンドを使って、画面に What is your name? と表示するにはどうすればいいでしょう。

<sup>2</sup>Windows では \* や ? はワイルドカードと呼ばれます。Windows では、スタートボタンからコマンドプロンプトを起動すれば、キーボードからのコマンド入力が可能であり、そこでワイルドカードを利用できます。ただし、コマンドプロンプトでのワイルドカードは、シェルではなく、各コマンドが解釈するものなので、ワイルドカードが利用できるかどうかは実行するコマンドに依存します。

```
echo What is your name?
```

を、結果を予想してから実行してみましょう。

メタキャラクタである `?` を、普通の文字として `echo` コマンドで出力するには、シェルによる `?` の展開を抑止（エスケープ）してから、`echo` コマンドの引数に与える必要があります。

また、シェルが特殊な意味を持つ記号として解釈する文字には、`$` や `"`、空白等、これまでに紹介した以外にも存在します。コマンド行において、それらを普通の文字として扱う場合にも、次のようにして特殊文字の持つ意味を無効にする必要があります。

1. 単一の特殊文字 `c` に対する特殊な意味の無効化

```
\c
```

2. 文字列 `string` 内の特殊文字に対する特殊な意味の無効化

```
'string'
```

- `string` 内に `'` を含む場合、この方法は使えません。
- 空白を含む文字列を、空白で区切られた複数の引数としてではなく、単一の引数としてコマンドに与えたい場合、この方法は便利です。(引数の区切り文字としての空白文字の意味を無効にします)

### 1.2.1 例

```
echo What is your name\?
echo 'What is your name?'
```

前者の場合、`echo` の引数が 4 個であるのに対し、後者では一つです。

時間に余裕があれば、`tcsh` のオンラインマニュアルを読むことも有意義です。

## 1.3 練習問題

1. カレントディレクトリを、ホームディレクトリ内の `eis17` に変更した上で、ディレクトリ `/pub/eis/metawork` に存在する `b` で始まる名前のファイルを、一度の操作で全てカレントディレクトリに複写してください。
2. ホームディレクトリにファイルがたくさんある人はホームディレクトリ内を整理してください。ディレクトリを作ってから、メタキャラクタを使ってファイルを移動したり、不要なファイルを削除しましょう。

注意：コマンドの実行結果を取り消すことは基本的にはできませんので、ファイル操作は慎重に行ってください。特にメタキャラクタを使ってファイル操作を行う際には、メタキャラクタの展開結果を事前に確認する習慣を身につけましょう（例えば `ls pp*.c` をまず実行してみるなど）。

3. `echo` コマンドを使って画面に `What's your name?` と表示する方法を考え、いろいろと試してみてください。

4. `cp a b c` と `cp 'a b' c` は、各々、何をするための操作かを考えてください。

実際に操作を試したければ、これらを実際に行うために必要なファイル等を `echo` コマンド等で作成して行ってください。

(ヒント：ファイル名には空白文字を含めることも可能です。ただし、この問を解く以外の場面では推奨しません。)

## 2 正規表現 — 文字列検索における検索パターン指定

`grep` の *pattern* 引数や `less` における */pattern* 等では、正規表現 (regular expression) と呼ばれる検索パターンを指定できる。その代表的なものを以下に示す。これ以外の正規表現については、`grep` のマニュアル等を参照のこと。

<code>^</code>	行の先頭
<code>\$</code>	行の終わり
<code>.</code>	任意の一文字
<code>[...]</code>	... のうちの任意の一文字。a-z や 0-9 のような範囲指定も有効
<code>[^...]</code>	... がない任意の一文字。a-z や 0-9 のような範囲指定も有効
<code>r*</code>	ゼロ回以上の <i>r</i> の繰り返し。2 文字以上からなるパターン <i>str</i> の繰り返しを指定したければ $(str)^*$
<code>r+</code>	1 回以上の <i>r</i> の繰り返し。2 文字以上からなるパターン <i>str</i> の繰り返しを指定したければ $(str)^+$
<code>\c</code>	文字 <i>c</i> の特殊な意味をなくす

次の例を実行して正規表現の基本的な使い方を確認しよう。なお、ここでは、初めに `cd` コマンドを使って、ファイル `75.60k.vocab.romaji` が存在するディレクトリをカレントディレクトリとした後に、`grep` コマンドで正規表現を使ってみる。

```
cd /pub/eis/data
grep kuki 75.60k.vocab.romaji
grep n..kuki 75.60k.vocab.romaji
grep kuki$ 75.60k.vocab.romaji
grep 's.*kuki$' 75.60k.vocab.romaji
```

`.*` は長さが 0 以上の任意の文字列にマッチするので、最後の例では、`s` を含み行末が `kuki` である行 (パターン `s.*kuki$` を含む行) のみが出力される。

**注意 1** 正規表現に使われる特殊な文字 (メタキャラクタ) と、シェルのメタキャラクタ (ファイル名に展開される) では、一般に意味が異なる。

**注意 2** シェルのメタキャラクタでもある `*` 等を、正規表現として `grep` の引数に与えるには、シェルによる `*` の展開を抑止 (エスケープ) する必要がある。

### 2.1 練習問題

以下は全てディレクトリ `/pub/eis/data` に存在するファイル `75.60k.vocab.romaji` に対して行いなさい。

1. ローマ字表記の読みが z を含み maru で終る行をすべて抽出しなさい。
2. ローマ字表記の読みが a で始まり maru で終る行をすべて抽出しなさい。
3. ローマ字表記の読みが母音で始まり母音以外で終る行の数を求めなさい。

## 3 ファイルの詳細情報とアクセス権

### 3.1 コマンドの実体とパス名を用いたコマンド実行

1. ディレクトリ `/bin` にどんなファイルがあるか調べましょう。`ls /bin` を実行してください。  
`cat`, `cp`, `ls`, `pwd` などのコマンドが、`/bin` の中のファイルの名前として表示されます。
2. コマンド行に `/bin/pwd` とタイプして `<ENTER>` を押してみてください。  
`/bin/pwd` でも `pwd` コマンドを実行できましたね。

このことから予想されるとおり、`pwd` コマンドの実体は、ディレクトリ `/bin` に存在する通常のファイル `pwd` です。UNIX コマンドの多くは、コマンドと同名のファイルとして存在していて、`/bin` に存在するファイルはすべて実行可能なコマンドです<sup>34</sup>。

コマンドの引数にパス名が使えるのと同様に、コマンド自身を指定する際にもパス名が使えます。先ほど `/bin/pwd` で `pwd` コマンドを実行できたのは、そのためです。相対パス名でも可能です。

より正確にいうと、ファイルとして存在するコマンドを実行するには、本来、コマンドのパス名を使う必要があります。しかし、それでは不便ですので、`/bin` のような特定のディレクトリに存在するコマンドを、コマンド名のみで実行できるようにする仕組みが設けられています<sup>5</sup>。そのおかげで、`/bin/pwd` や `/bin/ls` 等を `pwd` や `ls` とだけタイプして実行できるのです。

#### 3.1.1 練習

1. カレントディレクトリを、ホームディレクトリ内の `eis17` に変更してから `hostname` コマンドを実行してください。  
コンピュータのホスト名（または FQDN (Full Qualified Domain Name)）が出力されます。
2. `ls /bin/h*` を実行して、`/bin` に `hostname` コマンドがあることを確認しましょう。
3. 絶対パス名を使って `hostname` コマンドを実行しましょう。  
`/bin/hostname` とタイプして `<ENTER>` です。
4. `/bin/hostname` をカレントディレクトリにコピーしてください。ただし、コピー先のファイル名を `myhostname` とします。  
`cp /bin/hostname myhostname` でいいですね。実行したら `ls` コマンドで正しくコピーされたかを確認してください。

<sup>3</sup>`/bin` に存在するファイルの大部分は実行可能なバイナリ (binary) ファイルです。これらはテキストファイルではないので、`cat` 等で中を読むことはできません。

<sup>4</sup>ファイルとしては存在しないコマンドとして、シェルが持っているコマンド (シェルの組み込みコマンド) があります。例えば `exit` はシェル組み込みコマンドです。シェル組み込みコマンドの使い方はシェルのマニュアルに記載されています。

<sup>5</sup>どのディレクトリに存在するコマンドが、コマンド名のみで実行できるのかは、設定に依存します。具体的には、`echo $PATH` で出力される、シェルの環境変数 `PATH` に設定されたディレクトリに存在するコマンドはファイル名 (コマンド名) のみで実行できます。

5. `myhostname` をコマンドとして実行しましょう。

(a) `myhostname` とタイプして `<ENTER>` を押してください。

特定のディレクトリに存在するコマンドを除き、コマンドはパス名で実行する必要がありますのでしたね。`eis17` は、その特定のディレクトリではありませんので、ここに存在するコマンドをコマンド名で実行することはできません<sup>6</sup>。

(b) カレントディレクトリにファイルがあることを明示した相対パス名を使って、`myhostname` を実行しましょう。

`./myhostname` とタイプして `<ENTER>` を押してください。

### 3.2 ユーザーとグループ

UNIX システムの利用者は、そのシステム内で重複しないユーザー名 (ログイン名) でシステムに登録されている必要があります。利用者は正しいユーザー名とパスワードを入力することで、システムの利用が許可されます (ログイン)。

さらに UNIX にはグループ (*group*) という概念があります。通常、UNIX のシステムには複数のグループが登録されており、各ユーザーは少なくともそのうちの一つに属しています。自分が所属するグループは

```
groups
```

というコマンドで調べることができます。試してみましょう。

### 3.3 ファイルの詳細情報

UNIX のファイルには、ファイル名の他にも種々の情報が付与されています。`ls` にオプション `-l` を付けて実行することにより、ファイルの詳細情報 (long format) を表示することができます。また、`ls` の引数にディレクトリやファイルの名前を与えると、`ls` はそれらの情報のみを出力します。複数のファイル名やディレクトリ名を `ls` に与えることも可能です。

したがって、例えばファイル `.emacs` が存在するディレクトリにおいて

```
ls -l .emacs
```

を実行すると<sup>7</sup>、ファイル `.emacs` のみの詳細情報を得ることができます<sup>8</sup>。

種類	リンク数	グループ	更新日時	
↓	↓	↓	<----->	
<code>-rw-r--r--</code>	<code>1</code>	<code>user1 group1</code>	<code>500 Nov 25 16:55</code>	<code>.emacs</code>
<----->	↑		↑	↑
アクセス権	所有者		サイズ	ファイル名

<sup>6</sup>Windows のコマンドプロンプトでは、カレントディレクトリに存在するコマンド (プログラム) はコマンド名 (プログラムのファイル名) のみで実行できます。

<sup>7</sup>`ls` の引数にファイル名を指定した場合には、`.` (ドット) で始まる名前のファイル情報を出力する場合であってもオプション `-a` を付ける必要はありません。

<sup>8</sup>引数にファイル名やディレクトリ名を与えなければ、カレントディレクトリ内のファイルに関する詳細情報一覧が得られます。

**種類** ファイルの詳細情報における一番左の記号はファイルの種類であり、`-` は通常のファイルを、`d` はディレクトリであることを表します<sup>9</sup>。

**所有者** そのファイルを所有するユーザーの名前です。デフォルトでは<sup>10</sup>、ファイルを作成したユーザーが、その所有者となります<sup>11</sup>。

**グループ** この項目はファイルの所属グループを表します<sup>12</sup>。ファイルに所有者があるのと同様に、ファイルは第 3.2 節で説明したグループにも属します。デフォルトでは、ファイルの所有者が所属する第一番目のグループが、そのファイルの所属グループとなります。

**サイズ** byte 単位でのファイルサイズです。テキストファイルでは 1 文字が 1 byte です。ただし、日本語の文字では、通常、いわゆる半角文字を除き 1 文字あたり通常 2 byte です。

**更新日時・ファイル名** ファイルが最後に変更された日時とファイル名です。

### 3.4 ファイルのアクセス権

UNIX にはファイルに対する読み書き等の操作の権限を、ファイルの所有者や、ファイルが属するグループ内のユーザーなどに対して設定するための仕組みがあります。それらの権限をファイルのアクセス権 (*permission*) といいます。アクセス権をパーミッションやアクセス許可、使用許可等と呼ぶこともあります。

#### 3.4.1 アクセス権の読み方

ファイルのアクセス権は `ls -l` で表示されるファイルの詳細情報から調べることができます<sup>13</sup>。そのために必要なのはアクセス権と所有者およびグループです。本節では、第 3.3 節で例示したファイル `.emacs` の詳細情報

```
アクセス権: rw-r--r--, 所有者: user1, グループ: group1
```

を例に、アクセス権の読み方を説明します。

1. アクセス権を構成する 9 文字を 3 文字ずつの組に分けます。`.emacs` の場合は `rw-` と `r--` と `r--` の 3 組です。

各組は左から順に

- (a) ファイルの所有者 (user1) に対するアクセス権 (`rw-`)
- (b) ファイルのグループ (group1) に属する、所有者 (user1) 以外のユーザーに対するアクセス権 (`r--`)

<sup>9</sup>ファイルの種類はこれだけではないので、この欄が `-` や `d` 以外になることもあります。ここに表示される可能性のある記号は `ls` のマニュアルに掲載されています。

<sup>10</sup>「特に何もしなければ (by default)」

<sup>11</sup>ファイル `.emacs` は各ユーザーのホームディレクトリに予め用意されていたファイルであり、所有者も予め各ユーザーに設定されています。

<sup>12</sup>グループを表示するために、`ls` にオプション `-g` が必要な UNIX システムもあります

<sup>13</sup>ここで紹介するのは UNIX 系 OS に共通の基本的なアクセス権のみです。最近の UNIX 系 OS ではアクセス権をさらに細かく設定できるように独自の拡張が行われています。

(c) 上記以外のユーザーに対するアクセス権 (r--)

を表しています。

2. 3つの組それぞれについて、各組を構成する3つの文字からファイルのアクセス権を調べます。3つの文字は左から順に、次に示す3種類のアクセス権を表しています。

(a) ファイルの内容の読み取り権<sup>14</sup> (r または -)

r : 読み取り可能 (readable)  
- : 読み取り不可能

(b) ファイルへの書き込み権<sup>15</sup> (w または -)

w : 書き込み可能 (writable)  
- : 書き込み不可能

(c) ファイルの実行権<sup>16</sup> (x または -)

x : 実行可能 (executable)  
- : 実行不可能

以上より、.emacs のアクセス権は

1. 所有者 user1 に対して、読み取り可・書き込み可・実行不可 (rw-)
2. グループ group1 に属する、user1 以外のユーザーに対して、読み取り可・書き込み不可・実行不可 (r--)
3. その他のユーザーに対して、読み取り可・書き込み不可・実行不可 (r--)

であることがわかります。すなわち、.emacs の所有者 user1 は、このファイルの内容を読んだり、ファイルに書き込みを行うことができます。グループ group1 に属する user1 以外のユーザーと、その他のユーザーは、このファイルの内容を読み取ることのみが許されています。

### 3.4.2 アクセス権の変更 — chmod コマンド

ファイルの所有者は、chmod (change file mode) コマンドを使って、ファイルのアクセス権を変更することができます。chmod の基本的な使い方は

```
chmod mode file ...
```

であり、mode には、file ... の現在のアクセス権をどのように変更するのかを指定します。mode に指定する要素は「誰に」「どの権限を」「与える(または取り除く)」の三つですが、記述の順番は次のとおりです。

1. 「誰に」: u, g, o, a の何れかを指定します<sup>17</sup>。

- u : ファイルの所有者 (user<sup>18</sup>)

<sup>14</sup>ディレクトリの場合は、ディレクトリ内のファイル一覧を得る権限

<sup>15</sup>ディレクトリの場合は、ディレクトリ内にファイルを作ったり、ファイルを削除する権限

<sup>16</sup>ディレクトリの場合は、ディレクトリ内のファイルにアクセスする権限 (検索権)

<sup>17</sup>u, g, o を組み合わせるなど、より複雑な記述も可能ですが、何れか一つを指定する方法を知っていれば困らないでしょう。詳細を知りたいければ man や jman で調べてください。

<sup>18</sup>owner と呼ぶべきですが、other と区別するために user と記します

- g : 所有者を除くファイルのグループに属するユーザー (group)
  - o : その他のユーザー (other)
  - a : u, g, o の全て (all)
2. 「与える」または「取り除く」: + か - を指定します。
- + : 与える
  - - : 取り除く
3. 「どの権限を」; r, w, x の何れかを指定します。
- r : 読み取り権
  - w : 書き込み権
  - x : 実行権

例えば

```
chmod g+w file
```

だと、*file* のグループに所属する (所有者以外の) ユーザーに、書き込みの権限を与えることになります。また、

```
chmod a-w file
```

では、全てのユーザーに対して、*file* への書き込みを禁止します。

### 3.4.3 発展 — アクセス権の 8 進数指定 (絶対指定)

`chmod` における *mode* を `g+w` などの記号で指定する代わりに、数 (8 進数) で指定することも可能です。数による指定では、まず次の表にあるように、読み取り・書き込み・実行の可または不可を表す 3 文字の記号を、- のときは 0 に、それ以外の場合は 1 に対応させます。そうしてできあがった 0 と 1 の並びを 3 桁の 2 進数として解釈し、8 進数で表現します。

記号	0,1 の並び (2 進数表記)	8 進数	意味
---	000	0	読取不可・書込不可・実行不可
--x	001	1	読取不可・書込不可・実行可
-w-	010	2	読取不可・書込可・実行不可
-wx	011	3	読取不可・書込可・実行可
r--	100	4	読取可・書込不可・実行不可
r-x	101	5	読取可・書込不可・実行可
rw-	110	6	読取可・書込可・実行不可
rwx	111	7	読取可・書込可・実行可

上記の一桁の 8 進数を、所有者・グループ・その他のユーザーに対するアクセス権として順に並べると、アクセス権の 8 進数表記ができあがります。この方法による `chmod` の例を次に示します。

- `chmod 644 file` (すべてのユーザーが読み取り可能, 所有者のみ書き込み可能)
- `chmod 755 file` (すべてのユーザーが読み取りと実行可能, 所有者のみ書き込み可能)

この方法では、現在のアクセス権とは無関係に、指定したアクセス権が設定されることになります (絶対指定)。

### 3.5 特権ユーザーとセキュリティ

UNIX には「特権ユーザー」、「スーパーユーザー」等と呼ばれるシステム管理のための特別なユーザーが存在します。UNIX の場合、通常、特権ユーザーのユーザー名は root です。ユーザー root にはシステムに対するすべての操作が許されていて、root はシステム内のどんなファイルでも見ることができるし、変更したり削除することもできます<sup>19</sup>。

システムが動作するのに必要な諸設定は、一般のユーザーが書き込めないファイルに保存されているので、管理者がシステムの設定変更を行うためには root として作業する必要があります。一方で、root が操作を誤ればシステムが壊れることもあり得ます<sup>20</sup>。

さて、もし root のパスワードが悪意ある人に知られてしまったら、どういう事が起こり得るか、想像できますね。

root を含むユーザーのログイン用パスワードは、暗号化されてパスワードファイルに格納されています。したがって、パスワードファイルの中を見ることができたとしても、暗号を解読できなければパスワードはわかりませんが、一般利用者のパスワードが漏れて悪意ある人がシステムに侵入すると、root のパスワードを解読される危険性は高まります。すなわち皆さんのパスワードが漏ることによって被害を被るのは、皆さん自身だけとは限らないということです。

パスワードの取り扱いには注意しましょう。また、安易なパスワードを設定しないでください。

### 3.6 問題

1. ホームディレクトリに存在するファイルの詳細情報を一覧表示して、観察してみましょう。通常のファイルとディレクトリの違いや更新日時、所有者、アクセス権等を確認してください。
2. ファイル `/bin/cat` や `/bin/ls` の詳細情報を表示してください。適切なアクセス権になっていることが確認できますか？
3. `ls /bin` と `ls -F /bin` の出力を比較してください。実行可能なファイルを表す印は何でしょう？
4. 空行（改行）が一つだけ入ったファイル `now` を作ります。ここでは `echo` コマンドとリダイレクトを使って

```
echo > now
```

を実行しましょう。既に `now` が存在したら、中身は上書きされて改行だけになります。

ファイル `now` の詳細情報を `ls -l now` で表示し、アクセス権やファイルサイズ、更新時刻などを確認してください。

5. 現在の日付や時刻をファイル `now` に書き込みましょう。date コマンドとリダイレクト (`>` または `>>`) を使ってください。
6. 再度 `now` のファイルサイズや更新時刻などを確認しましょう。now の内容も見てみましょう。

<sup>19</sup>従って、他人に知られては本当に困る情報を、共用の計算機システムに保管するのは避けるべきです。これは Windows 機にも言えます。ただし、システム管理者は、システムの運用上本当に必要な場合を除き、個人のファイルを覗き見ることはしませんし、道義上するべきではありません。

<sup>20</sup>システムの管理者が、管理目的以外で root としてシステムを利用するのは避けるべきです。

7. `chmod` コマンドを使って `now` のアクセス権を `-w-r--r--` に変更してください。(ファイル所有者の読み取り権を外します)

8. `now` の中を `cat` コマンドなどで見てみましょう。

見えないはずですね。

9. `now` のアクセス権を `r--r--r--` に変更してから中身を見てみましょう。(ファイル所有者に読み取り権を与えてから書き込み権を外すなど)

10. 再度, `now` に現在の日付や時刻を書き込みましょう。

できないはずですね。書き込み権を外すと, 誤操作からファイルを保護できます。

11. `now` の削除を試みましょう。

書き込み許可のないファイルは, すぐには削除されず,

```
rm: remove write-protected 通常ファイル 'now'?
```

のような表示がでます。

ここでは `n` で答えてください。ファイルは削除されません。

12. `now` のアクセス権を `rw-r--r--` に戻してから, `now` を削除しましょう。

13. 自分の生まれた月のカレンダーを表示するコマンドを作って実行しましょう。

複雑な処理をするコマンドを作成するにはプログラミングの知識が必要ですが, 簡単な処理をするコマンドであれば, 既存のコマンドを組み合わせることでテキストファイルに記入し, そのファイルに実行の許可を与えるだけで作成することができます<sup>21</sup>。

以下の手順に従って作業をしてください。

(a) 次のコマンドを順に実行して, 結果を確認してください。

```
echo name was born in
cal month year
```

ここで, `name` には自分の名前を, `month year` には生まれた月と西暦を与えてください。

(b) 前項で実行したコマンド行の内容 (`echo ...` と `cal ...`) 2 行が入ったテキストファイルを作成してください。ファイル名は `mybirth` としましょう。ファイルの作成には `emacs` などのテキストエディタを用いるか, `echo` コマンドによるリダイレクトを使ってください。

(c) そのファイルに, すべてのユーザーが実行できる実行権を与えてください。

(d) ファイルに実行権を与えたので, ファイルをコマンドとして実行できるようになりました。作成したコマンドを実行してください。

<sup>21</sup>このテキストファイルをシェルスクリプトといいます。ここでは扱いませんが, シェルスクリプトでは制御構造や変数など, より高度なプログラミングの機能を使うことも可能です。