情報科学演習 資料 5
grep と awk を用いたテキストデータの処理

令和元年 5 月 20 日

目 次

1 教材テキストファイル **1**

2 less を使ったファイル閲覧と文字列検索 **1**
- 2.1 ページャーと less . . . 1
- 2.2 less の起動法 . . . 1
- 2.3 less の主要コマンド . . . 2
- 2.4 練習 . . . 2

3 grep を用いた特定行（レコード）の抽出 **3**
- 3.1 grep の基本的な使い方 . . . 3
  - 3.1.1 実行例 . . . 3
- 3.2 grep の出力を less で読む . . . 3
- 3.3 grep の一般書式とオプション . . . 4
- 3.4 問題 . . . 4

4 正規表現 — 文字列検索における検索パターン指定 **4**
- 4.1 問題 . . . 5

5 AWK を用いた特定フィールド（列）の抽出 **5**
- 5.1 AWK について . . . 5
- 5.2 awk を用いたフィールド操作の基本 . . . 6
  - 5.2.1 実行書式 . . . 6
  - 5.2.2 出力するフィールドを変更する . . . 6
  - 5.2.3 入力フィールドの区切り文字指定を変更する . . . 7
- 5.3 ファイル名を与えずに awk を実行する—標準入力の利用 . . . 7
- 5.4 コマンド出力を別のコマンドの標準入力に与える—パイプ . . . 8
  - 5.4.1 例 1: grep と awk を組み合わせて特定行の特定列のみを抽出する . . . 8
  - 5.4.2 例 2: grep して awk して sort する . . . 8
- 5.5 問題 . . . 9

6 （参考）AWK における pattern 指定処理 **9**

1　教材テキストファイル

この資料で教材として扱うファイルは新聞記事から頻出単語約 60,000 語を抽出したテキストファイルであり，ファイル名は 75.60k.vocab.romaji である。このファイルは次のディレクトリにある。

/pub/eis/data （ディレクトリの絶対パス名）

75.60k.vocab.romaji には，各行につき単語の

- 表記
- カタカナ表記の読み
- 活用語の見出し語
- 品詞タグ (品詞を数値でコード化したもの)
- ローマ字表記の読み

が記述されていて，各項目は + (プラス) 記号で区切られている。このような行と列を持つ表形式のデータにおいて，行を**レコード**と呼び，列を**フィールド**と呼ぶことがある。各データ（ここでは単語）を表すのがレコードであり，各データの属性を表すのがフィールドである。

この資料では，このような表形式のテキストデータをコマンドで効率的に処理する方法の基本を学ぶ。

2　less を使ったファイル閲覧と文字列検索

2.1　ページャーと less

テキストファイルの中身を閲覧する道具として，cat コマンドやエディタの他に，ページャー (pager) という種類のコマンドがある。ページャーを使えば，比較的大きなテキストファイルの中身を手軽に閲覧することができる。more コマンドは UNIX に標準的に備わっている代表的なページャーであるが，ここでは more の機能拡張版である less を使ってファイルを閲覧する。さらにテキストファイル内から必要な情報を取得するために検索操作を行ってみる。

なお，コマンドの使いかたを調べる際に使う man コマンドは，マニュアルの表示に more か less を使うことが多い。また，この授業の後半で使うデータベースソフトウェアにおいても，検索結果の表示にページャーを使う。

2.2　less の起動法

less の基本的な起動法は次のとおりである。

```
less filename
```

ここで，*filename* には内容を見たいファイルの名前（ファイル名）を指定する。なお，カレントディレクトリに存在しないファイルを扱いたいければ，*filename* にはファイルのパス名を与える。

表 1: less の主要コマンド

機能	コマンド	コメント
終了	**q** (または **:q**)	
ヘルプ	**h**	
ページ移動	<SPACE> (または CTRL-f または **f**)	次ページに移動する
	b (または CTRL-b)	前ページに移動する
	j (または <ENTER> または ↓)	1 行下に移動する
	k (または ↑)	1 行上に移動する
	G	最後の行に移動する
	*n***G**	第 *n* 行に移動する
テキストを検索する	/*pattern* <ENTER>	*pattern* が最初に現れる位置に移動する (下方検索)
	?*pattern* <ENTER>	*pattern* が最初に現れる位置に移動する (上方検索)
	n	一番最近行った検索を繰り返す
	N	一番最近行った検索を逆方向に繰り返す

2.3 less の主要コマンド

表 1 は，less でテキスト文書を閲覧している最中に使える，less が持っているコマンドである。

2.4 練習

1. まず，cat コマンドで教材テキストファイル 75.60k.vocab.romaji の内容を表示してみよう。なお，このファイルは大きなファイルであるため，cat コマンドを実行したらすぐに CTRL-c を押して，コマンドを終了させるのがよい。

2. less コマンドでファイルの中身を見てみよう。less では，<SPACE> を押して，次のページを閲覧できる。その他の操作は第 2.3 節の less の主要コマンドを参照のこと。

3. `kaiba` を検索語として検索することによって，カイバという単語を漢字ではどう記すのかを調べなさい。less で検索するには / に続いて検索語 (pattern) を記入し <ENTER> を押す。検索語の記入途中でタイプミスをしたら，CTRL-c で検索語入力を中止できる。

4. 新聞記事に現れた「教育」を含む単語にはどのようなものがあるか，検索を繰り返して調べなさい (n コマンドを使う)。検索語は `kyoiku` である。

5. 時間があれば，`man less` で less のコマンドを調べたり，less の操作をいろいろと試してみよう。

3　grep を用いた特定行（レコード）の抽出

この章では，grep コマンドを用いて，テキストファイルから特定の文字列を含む行のみを取り出して閲覧する方法を学ぶ。

3.1　grep の基本的な使い方

grep の基本的な実行の形式は次のとおりである。

grep 文字列 ファイル名

この形式では，grep はファイルの内容から，特定の文字列を含む行のみを出力する。

3.1.1　実行例

以下の実行例において，ファイル名の指定には，シェルの入力補完機能（tab キーを押す）を使うのがよい。

まず，asshukukuki （圧縮空気）を含む行だけをファイルから取り出してみる。

```
grep asshukukuki /pub/eis/data/75.60k.vocab.romaji
```

次に asshuku （圧縮）を grep する。

```
grep asshuku /pub/eis/data/75.60k.vocab.romaji
```

「圧縮 (asshuku)」は欲しいけれども「合宿 (gasshuku)」はいらないならば，教材テキストでの項目（列）区切りが +（プラス記号）であることを利用して，

```
grep +asshuku /pub/eis/data/75.60k.vocab.romaji
```

を実行すればよい。

3.2　grep の出力を less で読む

まず教材テキストのファイルを，長い絶対パス名に代え，ファイル名だけで指定できるように，カレントディレクトリを変更しておく。

```
cd /pub/eis/data
```

さて,「あめ」を調べたいと思って

```
grep ame 75.60k.vocab.romaji
```

とすると，出力行が多すぎて結果を画面に表示しきれない。grep の出力を，パイプ (|) を使って less に入力すると，結果を less で 1 ページずつ見ることができる。

```
grep ame 75.60k.vocab.romaji | less
```

less の終了方法などについては，第 2.3 節の less の主要コマンドを参照のこと。

3.3 grep の一般書式とオプション

一般書式

grep [options] *pattern* [*file...*]

grep は *file* で名前を指定された入力ファイル (*file* が指定されてないか， *file* の部分に - が指定された場合は標準入力) を読み込み，与えられた *pattern* にマッチする部分を含む行を探す。

以下にしばしば使う grep のオプションを挙げるので試してみよう。さらに詳しく知りたいときには `man grep` を実行すればよい。

よく使う grep のオプション

`-n`：各出力行の前に，入力ファイルにおける行番号を表示する
`-r`：ディレクトリ下のすべてのファイルを再帰的に読み取る
`-v`：マッチした行を表示しない。（マッチしない行を表示）
`--help`：簡単なヘルプメッセージを出力する
`--version`：grep コマンドのバージョンを出力する

上記のうちで，`--help` と `--version` は多くのコマンドで共通に使えるオプションであり，これらでは引数の *pattern* は不要である。

3.4 問題

1. 教材テキスト (75.60k.vocab.romaji) から yuki を含む行のみを表示しなさい。
2. grep のオプションを使って，教材テキストに yuki を含む行が何行あるかを表示しなさい。必要なオプションは man コマンドで調べること。
3. 教材テキストのうち，ame を含む行のみを，リダイレクト (>) を使ってファイルに格納しなさい。ただし，格納先のファイルは，ホームディレクトリに存在するディレクトリ eis19 の下の ame とする。eis19 が存在しなければ，まず作成すること。
4. ファイル ame の中から 44 を含まない行のみを抽出し，パイプと less で閲覧しなさい。

4 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。

`^`	行の先頭
`$`	行の終わり
`.`	任意の一文字
`[...]`	... 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
`[^...]`	... にない任意の一文字。`a-z` や `0-9` のような範囲指定も有効
r`*`	ゼロ回以上の *r* の繰り返し。2 文字以上からなるパターン *str* の繰り返しを指定したければ (*str*)`*`
r`+`	1 回以上の *r* の繰り返し。2 文字以上からなるパターン *str* の繰り返しを指定したければ (*str*)`+`
c	文字 *c* 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，ファイル 75.60k.vocab.romaji が存在するディレクトリがカレントディレクトリであるとする。

```
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。

4.1 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に 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。

5 AWK を用いた特定フィールド（列）の抽出

この章では，改行記号で区切られたレコード（行）から，特定の記号で区切られたフィールド（列）を抜き出す方法を学ぶ。

5.1 AWK について

AWK は，与えられたテキストデータの各行（レコード）に対する処理を容易に記述できるのプログラミング言語である。AWK では，特定の列（フィールド）に対する操作も容易に指定でき，簡単な処理であれば，プログラムをファイルに保存せずとも，コマンド行の操作だけで行うことができる。

この授業で利用する AWK は GNU 版の Gawk であり，awk あるいは gawk コマンドとして実装されている。例えば，awk を使って教材テキスト

単語表記+カタカナ読み+活用見だし語+品詞コード+ローマ字読み（改行）

のうち，1 列目の単語表記フィールドのみを取り出して less で閲覧するには，

```
awk -F+ '{print $1}' 75.60k.vocab.romaji | less
```

を実行すればよい。

AWK とその使用方法全般に興味があれば，例えば，GNU Awk のユーザーズガイド (Effective AWK Programming) を読んでみるとよい。

5.2 awk を用いたフィールド操作の基本

5.2.1 実行書式

awk の基本的な実行の書式は次のとおりである。

```
awk [オプション] 'awk のプログラム命令' [対象とするファイル]
```

この授業では，awk の各種オプションおよびプログラム命令のうち

1. レコード内フィールドの区切り記号を指定する -F オプション
2. 抽出したいフィールド番号を指定して出力する print 命令
3. そのフィールド番号記述方法（$番号）

の利用法の習得は必須である。

5.2.2 出力するフィールドを変更する

以下，教材テキスト 75.60k.vocab.romaji はカレントディレクトリに存在するとする。

最初の awk の実行例

```
awk -F+ '{print $1}' 75.60k.vocab.romaji | less
```

において，`$1` は区切り記号+で区切られた第 1 番目のフィールドを表している。では，`$1` を `$2` や `$3` 等に変更するとどうなるか，試してみよう。

さらに，$番号をカンマで区切って並べれば，フィールドの出力をいろいろと変更することができる。

```
awk -F+ '{print $2, $1}' 75.60k.vocab.romaji | less
```

次の例も試してみよう。

```
awk -F+ '{print $0}' 75.60k.vocab.romaji | less
```

実行結果からわかるとおり，$0 は処理中の行全体を格納する変数である。

注意　「$番号」の間にカンマを入れると awk は列を空白で区切って出力するが，カンマを入れずに

```
awk -F+ '{print $2 $1}' 75.60k.vocab.romaji | less
```

とすれば，$2 と $1 の間は区切られない（$2 と $1 を結合して出力する）[1]。

5.2.3　入力フィールドの区切り文字指定を変更する

教材テキストでは，列（フィールド）が + で区切られているが，/ を区切り文字とみなすように awk に指示してみよう。

```
awk -F/ '{print $2}' /pub/eis/data/ame | less
```

ここで，/pub/eis/data/ame は，教材テキストから ame を含む行のみを取り出したファイルである。これと同じ中身のファイルは前の練習問題でも eis19 に作成したので，代わりにそちらを使ってもよい。

注意　フィールドの区切り文字指定 -F は awk のオプションなので，この指定は省略することもできる。そのとき awk は（連続する）空白文字やタブ文字を列の区切りとみなす。

5.3　ファイル名を与えずに awk を実行する—標準入力の利用

コマンド行にファイル名を指定せずに，

```
awk '{print $1, $2}'
```

とだけ打って awk を実行してみよう。

新しいプロンプトがすぐに現れないのは，awk がまだ動いているためである。続いて，キーボードから

```
abc def ghi
```

と打ってエンターを押してみよう。`abc def` と出力される。さらに，

```
a bcd ef g
```

と打ってエンターを押すと，`a bcd` と出力される。

これらの出力が得られたのは，実行中の awk が `'{print $1, $2}'` の命令に従って，キーボードからの入力行を処理したためである。オプション -F を指定していないので，awk は空白を列の区切り文字とみなすことに注意しよう。

このように，ファイル名を指定しないで awk を実行すると，awk は標準入力 (通常はキーボード) からの入力を処理して出力する。この awk を正常終了させるには ctrl-d (EOF; 入力の終わり) を押せばよい。

注意　多くの UNIX コマンドは，ファイル名を省いて実行すると，awk と同様に標準入力から入力を受け取る。

[1] 空白以外の文字を出力フィールドの区切り文字にしたければ，`'{OFS = "+"; print $2, $1}'` のように awk の組み込み変数 OFS に区切り文字を代入するか，`'{print $2 "+" $1}'` とすればよい。

5.4 コマンド出力を別のコマンドの標準入力に与える—パイプ

先に実行した

```
awk -F+ '{print $1}' 75.60k.vocab.romaji | less
```

と同じことを，awk の引数にファイル名を与えずに行うには，どうしたらよいだろうか。前節では，ファイルの代わりにキーボードから abc def ghi 等の入力を与えたが，75.60k.vocab.romaji の中身を全部キーボードから打ち込む訳にはいかない。

答えは，cat とパイプを使って，

```
cat 75.60k.vocab.romaji | awk -F+ '{print $1}' | less
```

とすればよい。

これを実行したときの動作は次のとおりである。ここでは「`| less`」については考えず，その左側だけに注目する。

- cat が教材テキストの中身を出力する。
- awk の引数にはファイル名が与えられていないので，awk は標準入力からデータを受け取る。
- パイプ | は cat と awk の仲立ちをし，cat の出力（教材テキストの中身）を awk の標準入力に直接流し込む。

  awk 単体での実行時にファイル名を略すと標準入力はキーボードになるが，この実行例ではパイプによって awk の標準入力が cat の (標準) 出力に結ばれた。

パイプを使うときには，ファイル名を指定すべきコマンドがどれなのかに注意すること。

5.4.1 例 1: grep と awk を組み合わせて特定行の特定列のみを抽出する

grep の実行例

```
grep asshukukuki 75.60k.vocab.romaji
```

とパイプおよび awk を組み合わせて，教材テキストの中の asshukukuki を含む行 (レコード) の，単語表記列 (フィールド) だけを出力する。

```
grep asshukukuki 75.60k.vocab.romaji | awk -F+ '{print $1}'
```

5.4.2 例 2: grep して awk して sort する

sort は文字通り，行を辞書順に並べ替えて出力するコマンドである。sort コマンドに *file* を読み込ませて，辞書順に行を並べ替えて出力するには

```
sort file
```

とすればよい。*file* を省略すると sort も標準入力からデータを読み込む。ここでは，他のコマンドからの出力を sort コマンドに与え，辞書の逆順に並べ替えてみる。

さて，次の例が何をするのか，よく考えてから実行してみよう。sort コマンドのオプション `-r` は逆順に並べ替えるものである。

```
grep ame 75.60k.vocab.romaji | awk -F+ '{print $5, $1}' | sort -r | less
```

5.5 問題

1. 教材テキストの中の ame を含む行のみから第 1 フィールドのみを出力し，less で閲覧しなさい。grep と awk をつかうこと。
2. 教材テキストから，yuki を含む行（レコード）の中の「ローマ字読み」の列（第 5 フィールド）のみを抽出し，それをアルファベット順に並べ替えて出力しなさい grep, awk, sort をつかうこと。
3. 教材テキストのうち，ame を含み、かつ、44 を含まない行のみを取り出して less で閲覧しなさい。パイプと grep のオプションを使うこと。
4. ホームディレクトリで ls -al を実行しなさい。その結果から，4 を含む行のみを表示しなさい。パイプを使うこと。
5. `ls -l` の出力のうち,「ファイル名」(10 列目) と「ファイルサイズ」(5 列目) の列のみを出力しなさい。パイプを使うこと。

6 （参考）AWK における pattern 指定処理

awk への「プログラム命令」を

pattern {*action*}

の形式[2]で与えれば，awk は指定したパターン (*pattern*) の行に対してのみ動作 (*action*) を行う。ある文字列を含む行のみを awk で処理したければ，*pattern* として,「/文字列/」 を指定すればよく，例えば，

```
awk -F+ '/ame/ {print $1}' 75.60k.vocab.romaji
```

は，入力行に ame を含む行の第 1 フィールドのみを出力する。これは

```
grep ame 75.60k.vocab.romaji | awk -F+ '{print $1}'
```

を実行するのと同じである。
また,「/文字列/」に代え,「/正規表現/」の指定も可能である。
次の例が何をするのか考えて実行してみよう。

```
awk -F+ '/ame/ {print $0}' 75.60k.vocab.romaji
```

```
awk -F+ '/n..kuni/ {print $0}' 75.60k.vocab.romaji
```

また，これまでの例や問題において，grep と awk を使ったものを，awk だけで実行してみよう。

[2] この形式を *pattern1* {*action1*} *pattern2* {*action2*} のように複数並べることで，異なる *pattern* に対して異なる処理を行わせることも可能である。