FSL course: シェルスクリプト *

FMRIB Group
翻訳: 根本 清貴 †

2014 年 2 月 20 日

スクリプトを知っていると…

- よく使うコマンドをファイルひとつにまとめられます
- 処理を自動化/バッチ化できます
- 柔軟で設定可能な新たなツールを作成できます
- 他の人が作ったスクリプトやツールを理解できます

"3 年前にスクリプトを無理にでも教えてくれたらよかったのに!" - Dr. Parry

目次

* http://fsl.fmrib.ox.ac.uk/fslcourse/lectures/scripting/
† FMRIB の Steve Smith 教授の許諾を得て翻訳しました

1 はじめに

- よく使われるスクリプト言語には以下のようなものがあります。
  - シェルスクリプト sh, bash, csh, tchs
  - 他のスクリプト言語 TCL, Perl, Python
- ここでは、**sh** を使います。シンプルで、移植性が高いのに加え、強力でコマンドラインのように使えるからです。
- 繰り返しのタスクは**すごく**速く処理できます。
- スクリプトはどのコマンドが実行されたかの正確な記録ともいえます。
- スクリプトを使うことで手入力で起こる小さなエラーを避けることができます。
- スクリプトを知っていると画像解析だけでなく様々なところで便利です。
  - 異なるオプションの BET を自動で呼び出す
  - コマンドひとつで被験者の画像の様々な値を求める
  - 刺激提示データや行動データから刺激のタイミングを抽出する
  - ファイル名をまとめて変更する
  - ファイルの拡張子をまとめて変更する (例 png を tiff に)

2 データ管理

- スクリプトで悩みすぎて気が狂わないように系統だったファイル名とディレクトリ名をつけることをおすすめします。
  - 例: Con0001 , Con0002, Con0003 , ... , Con0020, Pat0001, Pat0002, ... , Pat0020
- Con20 ではなく、Con0020 のようにゼロを並べるようにすると一貫性をもたせることができ、スクリプトが容易になります。
- イギリスでは (他の国でもそうでしょうが)、ファイル名に個人を特定できる情報 (氏名、生年月日、イニシャル) を入れることは禁止されています。
- 元のファイル名をより意味があって、一貫性のあるファイル名に変更しましょう。
- ディスク量をチェックし、正しくない解析ファイルは削除しましょう。(多くの人が、res.feat, res+.feat, res++.feat, res+++.feat のようなファイルを保存していて、これらは混乱の元ですし、ディスクの無駄遣いです)

3 シェルスクリプトの基本

- Bourne シェル (sh) スクリプトは、ひとつのファイルの中にコマンドが何行かにわたって記載されており、bash の原型である Bourne シェルで動作します。もっとも単純なスクリプトは、ターミナルのプロンプトから入力するコマンドと同じです。
- **sh スクリプトの最初の行は必ず** `#!/bin/sh` です。
- 以下を覚えていてください。
  - スクリプトには実行権限をつけることを忘れないでください。

    ```
    chmod a+x filename
    ```
  - スクリプトはカレントディレクトリ (pwd) の中で走ります。
  - 同じ環境を引き継がないかもしれません。特に他人の環境で動作させる場合には気をつけてください。
- 例

  ```
  #!/bin/sh
  bet im1 im1_brain -m
  mv im1_brain_mask.nii.gz mask1.nii.gz
  ```
- スクリプトはいつでも `return` を使うことで停止できます。例: `return 0`

演習

ここで**演習: 最初のスクリプト-BET を使う**をやってみましょう。

3.1 シェルスクリプトの便利なテンプレート

- 系統的に学ぶ前に、シンプルだけれども非常に強力で有用なスクリプトを提示しましょう。これはいろいろな状況にあわせて修正できます。

```
#!/bin/sh
for filename in *.nii.gz ; do
  fname=`$FSLDIR/bin/remove_ext ${filename}`
  fslmaths ${fname} -s 2 ${fname}_smooth2
  mv ${fname}.nii.gz ${fname}_smooth0.nii.gz
done
```

- このスクリプトがすることは… 個々の画像 (*.nii.gz) に対して、平滑化を行い、_smooth2 で終わる別名のファイルとして保存し、元の平滑化されていない画像を、_smooth0 で終わるファイルとして保存します。
- どのように動作するか:
  - *.nii.gz に合致するファイルがひとつずつ *for* ループの中で変数 `filename` にセットされます。
  - `filename` から拡張子 (nii.gz) を除いたファイル名が変数 `fname` にセットされます。どうやってこんなことができるかは心配しないでください。詳細は後で説明します。
  - `${filename}`と`${fname}`は変数の値 (内容) を得るために使われます。
  - `fslmaths` は平滑化を行うために使われています。
  - `mv` はファイル名の変更のために使われています。(ここでは.nii.gz が必要なことに注意してください。fsl の様々なツールは.nii.gz があってもなくても動作するようになっています)

3.2 スクリプトのコツ

スクリプトを書くにはいくつかのコツがあります。

- コメントを入れましょう。(何ヶ月後もしくは何年後でも記憶を呼び起こせるようにです)
- スクリプトが走っているときに何をしているかがわかるように `echo` コマンドを入れておきましょう。
- もしスクリプトが変なことをはじめたり、何も動いていないようだったら、`Ctrl-C` で止めることができます。
- もし、ファイルを新規に作成したり、ファイルを変更したり、ファイルを削除したりするスクリプトを書くのだったら、まず最初にそれらの大事なコマンドの前に `echo` を使ってください。このバージョンを走らせると、そのスクリプトはコマンドをただスクリーンに表示します。これを注意深く確認してください。きちんと書けていたら、`echo` をとってスクリプトを走らせてください。
- 万が一のために大事なファイルのバックアップはとっておきましょう。

4 基本的なスクリプトの理解

- これから系統的に以下の項目をみていきます。
  - ワイルドカード
  - echo (スクリーンやファイルに表示)
  - 変数
  - かっこ
  - コマンドラインの引数
  - シングルクオートとバックスラッシュ
  - ダブルクオート
  - バッククオート
  - パイプ
  - ファイルのリダイレクション
- その後に for 構文のような便利なツールやプログラムの構造をとりあげていきます。

4.1 ワイルドカード

ワイルドカードはファイル名のパターンマッチングに使うことができます。たとえば…

- *は任意の文字列をあらわします。
- ?は任意の一文字をあらわします。
- [abgj] は括弧内にある文字列のリストもしくは範囲内にある任意の一文字をあらわします。

```
$ ls（ディレクトリ内のファイルを表示）
  sub1_t1.nii.gz sub1_t2.nii.gz sub2_t1.nii.gz sub2_t2.nii.gz sub3_pd.nii.gz
$ ls sub*（sub から始まるファイルを表示）
  sub1_t1.nii.gz sub1_t2.nii.gz sub2_t1.nii.gz sub2_t2.nii.gz sub3_pd.nii.gz
$ ls sub1*（sub1 から始まるファイルを表示）
  sub1_t1.nii.gz sub1_t2.nii.gz
$ ls sub*t1*  （sub から始まり途中に t1 があるファイルを表示）
  sub1_t1.nii.gz sub2_t1.nii.gz
$ ls sub[13]*（sub から始まり次に 1 か 3 があるファイルを表示）
  sub1_t1.nii.gz sub1_t2.nii.gz sub3_pd.nii.gz
$ ls sub?_t2.nii.gz（sub から始まり次に任意の 1 文字が来て、その後に_t2.nii.gz となるファイルを表示）
  sub1_t2.nii.gz sub2_t2.nii.gz
```

4.2 ECHO

- echo はそれ以降の文字列を画面 (標準出力) に表示します。
- これは出力を表示したり、スクリプトの進捗状況を表示するのに便利です。

- ファイル名に対するワイルドカードや変数は echo が結果を表示する**前に**置き換えられます。

- 例

```
$ echo Hello All!
  Hello All!
$ echo sub*t1*
  sub1_t1.nii.gz sub2_t1.nii.gz
$ echo j*k
  j*k
```

演習

ここで**演習: Echo とワイルドカード**をやってみましょう。

4.3 変数

- たいていのプログラミング言語と同じように、シェルは様々なものを変数として保存することができます。
- シェル変数はすべて**文字列**です。
- 変数は次のようにして設定します。
  変数名=値
- 変数の名前はアルファベットで始まらなければいけませんが、その後には数字や下線_を使うことができます。
- 変数の値は`$`を変数の前につけることによって、使うことができます。
- 例

```
$ var1=im1.nii.gz
$ echo $var1
  im1.nii.gz
$ echo var1
  var1
$ ls $var1
  im1.nii.gz
```

4.4 かっこ

- アルファベットから始まる名前はなんでも変数名として使うことができます。
- 例:`v, v1, v1_1, v_filename_4`
- 変数の直後に文字列をつけると混乱の元になります。
- こんなとき、変数名をかっこの中にいれると問題がなくなります。
- 例:

```
$ v=im1
$ echo $v_new （v_new という変数を意味してしまう）
  （何も表示されない）
$ echo ${v}_new （変数 v の後に_new がついた文字列として認識される）
  im1_new
```

- 注意:まだ使われていない変数はデフォルトで空となっています。(エラーは出力されません)

4.5 引数

- スクリプトの中で、変数$1 $2 $3 などはコマンドラインの引数の値を保存しています。
- たとえば、reg_vol というスクリプトが次のように実行されたとします。

  ```
  $ reg_vol im1 3 abc
  ```

  このとき、$1 = im1, $2 = 3, $3 = abc となります。
- その他の特殊変数としては以下のようなものがあります。
  - $0 = スクリプト名 (しばしばパスも含まれます)
  - $# = 与えられた引数の数
  - $@ = すべての引数　(すなわち、$1 $2 $3 ...)
  - $$ = プロセス ID (このプロセスに一意の値)

演習

ここで**演習: コマンドラインの引数**をやってみましょう。

4.6 シングルクオートとバックスラッシュ

- シェルは変数とワイルドカードをコマンドを実行する**前に**展開してしまいます。
  - ときにこれは望ましくないことがあります。
- 展開されてしまうことを避けるために以下の方法があります。
  1. 特殊記号 (ワイルドカードや$) の前にバックスラッシュ \ をつける
  2. シングルクオート ’ で囲む
- 例:

```
$ var1=im1.nii.gz
$ echo $var1 （$var1 の値が展開される）
  im1.nii.gz
$ echo \$var1 （バックスラッシュによって$がただの文字となり展開されない）
  $var1
$ echo '$var1' （シングルクオートで囲まれた文字列は展開されない）
  $var1
```

4.7 ダブルクオート

- いくつかの文字列をまとめてひとつの引数として扱う場合には、ダブルクオート"を使う必要があります。
- 例:

```
$ v=Hello World
$ echo $v
  Hello
$ v="Hello World"
$ echo $v
  Hello World
```

- 注意:変数の置き換えはダブルクオートの中で行われますが、ワイルドカードは展開されません。
  例. `echo "*"`はただ*と表示されますが、
  `echo "$v"`は `echo $v` と同一です。

4.8 バッククオート

- コマンドの実行結果 (の文字列) はバッククオート ‘を使うことで取り込むことができます。
- これは変数の設定にとても便利です。
- 例:

```
$ v=‘ls sub[13]*‘
$ echo $v
  sub1_t1.nii.gz sub1_t2.nii.gz sub3_pd.nii.gz
$ echo ‘fslval sub1_t1 pixdim2‘
  4.0
```

- 注意:結果はたとえスペースを含んでいたとしても、単一の文字列として扱われます。

演習

ここで**演習: コマンドと変数**をやってみましょう。

4.9 パイプ

- シェルの非常に使い勝手の良い特徴のひとつに、複数のコマンドをつないで実行できるというものがあります。前のコマンドの出力結果を次のコマンドに渡します。
- このためにはパイプ|を使います。
- 例 (単語を数える wc というユーティリティを使います)

```
$ cat .bashrc | wc
  7 83 534 (7 行 83 単語 534 文字)
```

```
$ echo "Hello World" | wc
  1  2  12 (1 行 2 単語 12 文字 (空白と改行コードも 1 文字としてカウント))
```

- もう少しテクニカルに説明すると、パイプはあるコマンドの標準出力を別のコマンドの標準入力にリダイレクトします。
- 標準エラー出力に出力されるエラーメッセージはパイプでリダイレクト**されません**。

4.10 リダイレクション

- あるファイルからコマンドに入力したい場合は<を使います。
- あるファイルにコマンドの結果を出力したい場合は>を使います。
- コマンドの結果をあるファイルに追加したい場合は>>を使います。
- 例:

```
$ echo "smoothing=10mm" > settings.txt
$ echo "No lowpass" >> settings.txt
$ cat settings.txt
smoothing=10mm
No lowpass
```

5 便利なシェルのプログラム

- シェルスクリプトに共通かつほぼ必須のプログラムは以下のとおりです。

  基本
  - `test` (または `[ ]`)
  - `if`
  - `for`
  - `while`

  アドバンス
  - `grep` (検索)
  - `bc` (計算機)
  - `sed` (検索と置換)
  - `awk` (列の選択)

5.1 TEST

- `test` コマンドは 2 つの文字列もしくは整数を比較します。
- 別の簡便な表し方は引数を `[`と`]` でくくることです。
  注意:スペースを正しくいれるようにしてください！
- 文字列を比較するのか整数を比較するのかで構文はかなり異なります。(構文のオプションについては、`help test` をみてください)

- test はファイルの状態 (ファイルが存在するか、書き込み可かなど) を確認するのにも使えます。
  test $a = my 文字列が等しいか
  [$a = my] 上と同じ
  [$a -eq 2] 数字が等しいか
  [$a -gt 2] 数字の比較 (より大きい)
  [11 > 2] 数字の比較をして**いない**
  [-e im1.nii.gnii.gz] ファイルが存在するかを確認
  注意:整数でない数の比較は、bc を見てください。

5.2 IF

- if コマンドは他のプログラミング言語と同じように動作します。
- たいていの場合、test コマンドの結果を利用します。構文は以下のようになります。

```
if [ 条件 ] ; then
コマンド ;
else
コマンド 2 ;
fi
```

- else の部分は必須ではありません。
- 例

```
if [ $a = 2 ] ; then
  b="y-axis";
fi
```

演習

ここで**演習: 引数が妥当かを確認する**をやってみましょう。

5.3 FOR

- for コマンドはリストにある個々の単語に対して、一連のコマンドを実行するというものです。
- 構文:

```
for 変数 in 値のリスト ; do
    コマンド ;
done
```

- コマンドはリストにある一つ一つの項目に対して実行されます。
- つまり、リストで現在使われている単語が、変数としてセットされています。

- 例:

```
for filename in im1 im2 im3 ; do
    bet $filename $filename_brain ;
done
```

演習

ここで**演習: 画像のバッチ処理**をやってみましょう。

5.4 BC

- bc コマンドは計算機のように動作します。
- たいていは *echo*, **パイプ**, **バッククオート**を使って変数をセットするときに使われます。
- -l オプションを使うと、正確な浮動小数点演算を行うことができます。
- 例:

```
$ a=2;
$ a=‘echo "3 * $a + 1" | bc -l‘;
$ echo $a
  7
```

- 注意: ダブルクオートを使うことで、*がワイルドカードとして使われないようにしています。
- 整数でない数字の比較は bc を使えばできます。このとき、結果が偽 (false) ならば 0 となり、真 (true) ならば 1 となります。
  例 `echo "-1.2 < 0.5" | bc` は 1 を返します
  例 `echo "-1.2 > 0.5" | bc` は 0 を返します
  ですから、if 文においては以下のようになります:

```
if [ ‘echo "$a < $b" | bc -l‘ = 1 ] ; then ...
```

演習

ここで**演習: ボクセルの容積を計算する**を行なってみましょう。

5.5 WHILE

- while コマンドは条件が真である限り一連のコマンドを実行します。
- 構文:

```
while 条件; do
    コマンド;
done
```

- 条件はたいていは test 文です。

- 例:

```
a=1
while [ $a -lt 4 ] ; do
    bet im$a brain$a ;
    a=‘echo $a + 1 | bc‘ ;
done
```

演習

ここで**演習: Lightbox Viewer (画像一覧の作成)** をやってみましょう。

5.6 GREP

- grep コマンドは文字列のパターンを見つけます。
- たいていはファイルの中からある単語やフレーズを抽出します。
- パイプと一緒に使うと強力なフィルターとなります。
- 例:
  fslhd の出力から pixdim1 の値を見つけます。

```
fslhd im1 | grep pixdim1
```

5.7 AWK

- awk コマンドは汎用性の高いパターンマッチングツールです。
- シンプルながら非常に有用な使い方として、文字列から特定の列を抽出することが挙げられます。
- 刺激提示ファイルなど、タブ区切りのファイルを操作する時は非常に便利です。

- 第 N 列を選ぶ構文は:

```
awk '{print $N}'
```

- クオートとかっこは**示された通り**に使わなければなりません。
- 例:

```
$ v="im*.nii.gz"
$ echo $v
  im1.nii.gz im2.nii.gz im3.nii.gz
$ echo $v | awk '{print $2}'
  im2.nii.gz
```

演習

ここで**演習: 刺激提示ファイルを処理する**をやってみましょう。

5.8 SED

- sed コマンドは文字列を置換します。
- たいていは文字列を追加したり、削除したり、一部を変更したりするために使います。
- 変数を修正するのに非常に便利です。
- STRING1 を STRING2 に変更する構文は:
  `sed s/STRING1/STRING2/g`
- 例: `$ v="im*.nii.gz"`
  ```
  $ v=‘echo $v | sed s/im/Subject/g‘
  $ echo $v
    Subject1.nii.gz Subject2.nii.gz Subject3.nii.gz
  ```
- 警告: . * [ ] /はバックスラッシュがない限り、第 1 の文字列で特別の意味を持ちます。
- Tip: /の代わりにどんな文字でも使うことができます。
  例 `sed s@STRING1@STRING2@g`
  これはディレクトリやファイルを扱う時にとても便利です。

6 関数

- 他のプログラミング言語と同様に、シェルスクリプトの中で関数を定義することができます。
- スクリプトをひとつひとつが理解可能で再利用可能な部品に小さく分けるときに便利です。
- 関数は独立したスクリプトのように呼び出すことができます。

- 関数を作成する構文は:
  `function` **関数名** { **コマンド** ; }
  もっと短くすると
  関数名 () { **コマンド** ; }

- 例:
  ```
  function hi { echo "Hi! $1" ; }
  $ hi
    Hi!
  $ hi There
    Hi!  There
  ```

7 正規表現

- **正規表現**はパターンマッチングで使う構文で、たくさんのコマンドが正規表現を利用しています。(例: grep, sed)
- 非常に柔軟性が高いが故にすぐに使いこなせるるものではありません。

- しかし、いくつかのパターンは覚えやすくとても便利です。
- 類似点があるものの、ワイルドカードとは**異なります**。
- 正規表現で使われる特殊記号としては以下が挙げられます。

  `.` 任意の一文字に合致します
  `*` 直前の文字が 0 個以上あることに合致します
  `.*` 任意の文字列に合致します
  `[ ]` その範囲内にある任意の文字に合致します
  `^` 行頭を意味します
  `$` 行末を意味します
  `[^ ]` その範囲内に**ない**文字列に合致します

- 例:

```
$ echo "Hello world" | sed 's/w.*/X/g'
  Hello X（w の後に任意の文字列が来る文字列を X に置換）
$ echo "Hello world" | sed 's/w*/X/g'
  XHXeXlXlXoX XXoXrXlXdX（よくない例:直前に 0 個以上ある w を X に置換，ワイルドカードとの違いに注意）
$ echo "Hello world" | sed 's/^.* /X/g'
  Xworld（行頭の任意の文字列 + 半角スペースを X に置換）
$ echo "Hello world" | sed 's/.$/X/g'
  Hello worlX（行末の任意の一文字を X に置換）
$ echo "Hello world" | sed 's/[wo]/X/g'
  HellX XXrld（w と o を X に置換）
$ echo "Hello world" | sed 's/[^wo]/X/g'
  XXXXoXwoXXX（w と o 以外を X に置換）
```

演習

ここで**演習: ファイル内容とファイル名の変更**をやってみましょう。

8 FSL コマンド

- fsl のコマンドラインユーティリティはたくさんありますが、スクリプトを使いやすくするために特化したユーティリティがいくつかあります。

  1. `remove_ext`
     これは画像に特化した拡張子のみ取り除きます: .nii.gz .nii .hdr .img .hdr.gz .img.gz
     例 `remove_ext /Volumes/MJ/img1.nii.gz` の結果は`/Volumes/MJ/img1` です。
  2. `imtest`
     特定のファイルが存在し、それが画像ファイルであれば 1 を、そうでなければ 0 を返します
     例 `imtest ../img1` はもし`../img1.nii.gz` (もしくは`../img1.hdr`) が存在すれば 1 を返し

ます。

3. imglob これを使うと画像ファイルのみのリストを得ることができます。
   例 imglob *は*に合致するファイルから画像ファイルだけを並べます

4. imcp, immv, imrm, imln
   これらは画像の拡張子を指定することなく、画像ファイルに対して cp, mv, rm, ln を行うものです。(Analyze や nifti ファイルを扱っていて、どちらの形式かわからないときなどに便利です)
   例 imcp im1 im1_orig は nifti ファイルに対しては cp im1.nii.gz im1_orig.nii.gz と同じですが、Analyze ファイルに対しては、cp im1.hdr im1_orig.hdr と cp im1.img im1_orig.img のように動きます。

9 コマンドをもう一歩

- その他にも便利なコマンドがたくさんあります。例を挙げると:
  - basename
    ファイルの前についているディレクトリ情報と指定した拡張子を取り除きます
    例 basename /tmp/epi.nii.gz .nii.gz の出力は epi となります
  - dirname
    指定したファイルのディレクトリのみを返します
    例 basename /tmp/epi.nii.gz の出力は/tmp となります
  - sort
    アルファベット順もしくは数字順にファイル (の行) を並べ替えます
  - which
    実行ファイルがどこにあるかを教えてくれます
    例 which flirt の出力は/usr/local/fsl/bin/flirt となります

- 便利なコマンドの続きです
  - head
    ファイルの最初の n 行を表示します
  - tail
    ファイルの最後の n 行を表示します
  - touch
    空のファイルを作ります
  - paste
    ファイルを (水平方向に) 結合します

演習

ここで**演習: 動画の作成**をやってみましょう (ただしこれは上級者向けです)。

10 さらなる学習のために

- 画像解析のさらなるコマンドラインツールに関しては、下記をごらんください。
  http://fsl.fmrib.ox.ac.uk/fsl/fslwiki/Fslutils
- 特定のコマンドについて詳細を知りたい場合は:
  - そのコマンドの man もしくは help を実行してください
- 一般的なコマンドおよびスクリプトについて学びたかったら:
  - ウェブを検索しましょう
  - 他のスクリプトを見てみましょう (例 $FSLDIR/bin の中のスクリプト)
    メモ: それらがスクリプトかどうかはそのファイルに対して file を実行すればわかります
  - apropos はコマンドの説明の中に合致するキーワードがあるコマンドを検索します
    例 apropos merge はマージに関係するコマンドをすべて表示します
  - UNIX やシェルスクリプトに関する本 (ただし、大半はかなり専門的です)