オープンCAE勉強会@富山（第46回）講習会資料
2016年8月20日
2016年8月23日改訂

OpenFOAM®ユーザーのための シェルスクリプト入門

中山 勝之 （オープンCAE勉強会@富山）

講習の概要

OpenFOAM®を使用していると、シェルスクリプトを使用していることがあります。

シェルスクリプトは、複数の処理（コマンド）をまとめて行うことができる便利な機能ですが、コマンドに慣れていない初心者には取っつきにくい部分があるかもしれません。

今回の講習会は、OpenFOAM®のチュートリアルの1ケースを例にとり、OpenFOAM®を使用する際に良く見かけるシェルスクリプト（Allrun, Allclean等）について解説を行い、シェルスクリプトを自作して実行する演習を行います。

講習の構成と目標

構成

1. 説明：シェル・シェルスクリプトについて （10分）
2. 実習：シェルの操作・シェルスクリプト作成 （40分）
3. 実習：OpenFOAM®で使用されるシェルスクリプト（50分）

目標

1. Allrun, Allcleanの内容を解読できるようになる
2. 講習を通じてシェルスクリプトを読む機会を得ることで、プログラムが何をしているのかということを考えるようになって欲しい

講習内容のPC環境について

OS : Lubuntu MATE 16.04 (64bit)

OpenFOAM® Version : 3.0.x, 4.0, v3.0+, v1606+

講習用仮想マシンURL

https://drive.google.com/file/d/0B-srXSXryn85MlNWSVc5akxfRVE/view?usp=sharing

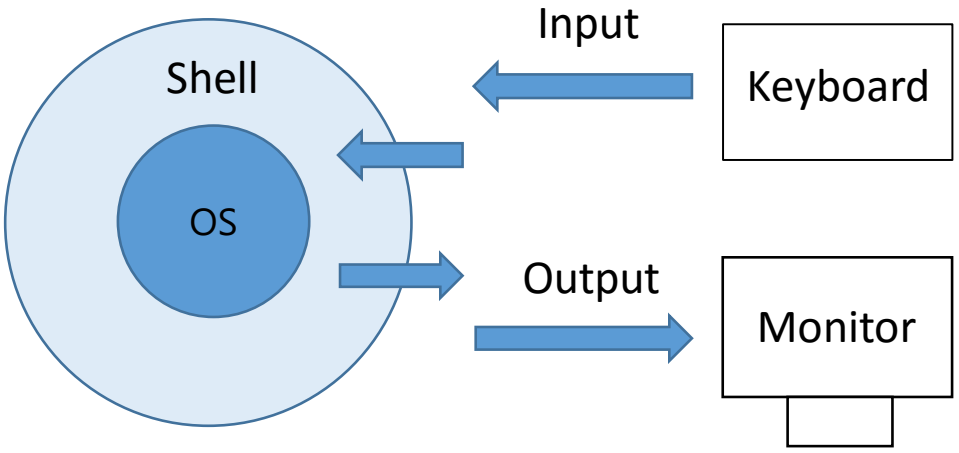
その１

シェル・シェルスクリプトについて

シェルについて

シェルとは、コンピュータのOS（オペレーティングシステム）を構成するソフトウェアの一つで、利用者からの操作の受け付けや、利用者への情報の提示などを担当するもの。転じて、OS以外のソフトウェアについても、その操作や表示を担当する機能やソフトウェア部品などのことをシェルと呼ぶことがある。

http://e-words.jp/w/%E3%82%B7%E3%82%A7%E3%83%AB.html



Ubuntuの標準シェルはbashが使用される

bash（バッシュ）は UNIX で使用するシェルのひとつで、GNUプロジェクトによるプロダクトのひとつ

https://ja.wikipedia.org/wiki/Bash

シェルスクリプト

シェルスクリプトとは, シェルの動作をまとめて記述したスクリプトのこと

決められた文法にしたがって処理を記述することによって, シェルでの処理をまとめて行う, 作業を自動化するということができる

基本的な文法

変数

制御構文 （if, case, whileなど）

演算・比較 （四則演算, 数値・文字列比較, ファイルチェック, 論理演算）

関数定義

本講習では、実習を通じて基本的な文法の使い方を学びます

ただし、時間に限りがあることもありすべての項目に触れることはできないため、文法・コマンドの詳細は参考文献を参照してください

Linuxのコマンドについて（1）

ファイル、ディレクトリ操作

コマンド	説明	使用例
ls	今いるディレクトリのファイルを表示する	ls -la システムファイルを含むすべてのファイルを表示 ls -ltr ファイルを更新日順に表示
cd	ディレクトリを移動する	cd ../ 相対パスで移動 cd /var/tmp 絶対パスで移動
pwd	今いるディレクトリのパスを表示する	
mv	ファイルを移動する	mv -f 移動先を上書き
cp	ファイルをコピーする	cp -rf 強制的に上書きコピー
rm	ファイルを削除する	rm -rf 強制的に削除
chmod	ファイルの権限を変更する	chmod 666 a.txt 単体で変更 chmod -R 777 logs/ フォルダの中身ごと変更
chown	ファイルの所有者、グループを変更する	chown user:user a.txt userユーザに変更

Linuxのコマンドについて（2）

ファイル内容表示・編集

コマンド	説明	使用例
less	ファイルの中身を確認する	less -N aaa.txt 行数を表示して確認する
more	ファイルの中身をページ単位で確認する	more -N aaa.txt 行数を表示して確認する
cat	ファイルを標準出力に出力する	cat aaa.txt bbb.txt 複数のファイルを連結して出力 cat aaa.txt |grep abc 標準出力をgrepする
tail	ファイルの末尾を表示する	tail -f log.txt 追加された分も表示する
vi	ファイルを編集する	vi aaa.txt編集モード a 追加する x 1文字削除する d 1行削除する :set number 行番号を表示 :wq　保存して終了 :q!　保存しないで終了

Linuxのコマンドについて（3）

その他

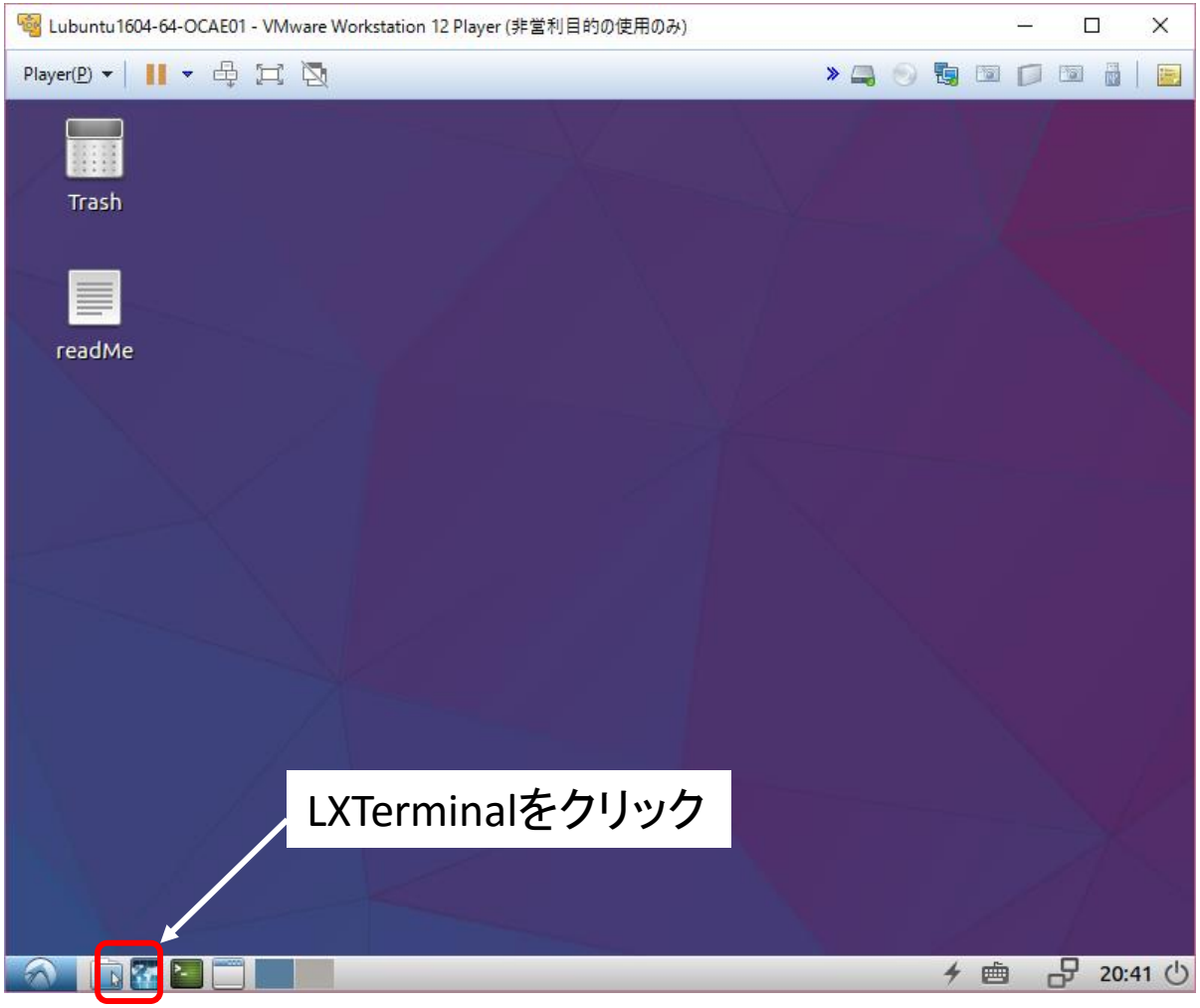
コマンド	説明	使用例
su	ユーザを変更する	su - rootユーザに変更
sudo	指定ユーザーでコマンドを実行する	sudo service smbd restart rootユーザでサービスを再起動
which	コマンドのフルパスを表示する	which php
tar	ファイルの圧縮、解凍をする	tar xzf buckup.tar.gz 解凍tar cvzf buckup.tar.gz buckup/ tar.gzで圧縮
diff	ファイルの差分を表示	diff a.txt b.txt

その 2

実習：シェルの操作・シェルスクリプト作成

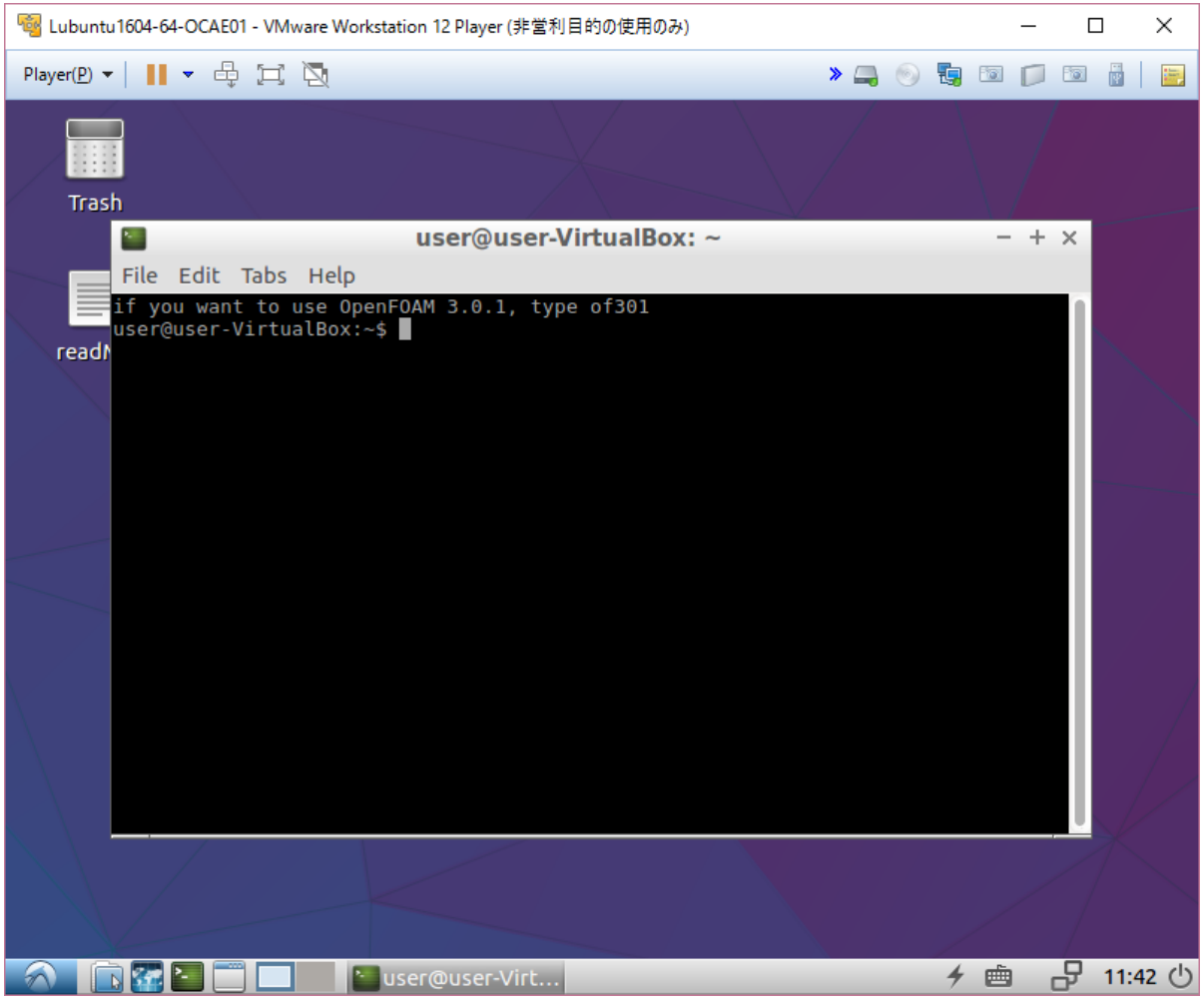
操作：ターミナル（端末）起動（1）

ターミナルを起動



操作：ターミナル（端末）起動（2）

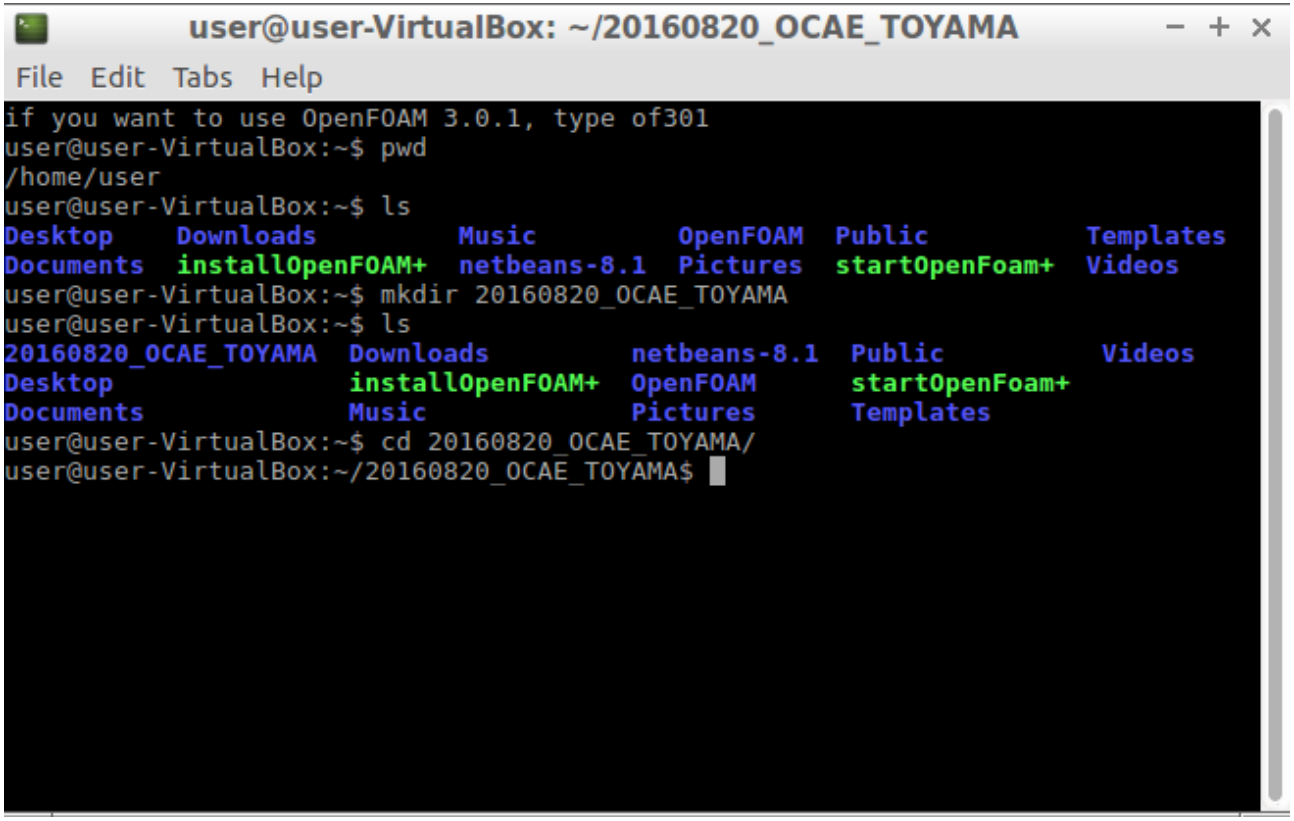
ターミナルを起動



操作：コマンド実行

コマンド[ls, pwd, mkdir, cd]を実行してみる

① $ pwd ⏎　　今いるディレクトリのパスを表示
② $ ls ⏎　　今いるディレクトリのファイルを表示
③ $ mkdir 20160820_OCAE_TOYAMA ⏎　20160820_OCAE_TOYAMA という名前のディレクトリを作成
④ $ cd 20160820_OCAE_TOYAMA ⏎　　ディレクトリ20160820_OCAE_TOYAMAに移動



操作：シェルスクリプトを作成・実行（test0.sh, 1）

コマンドlsとpwdを続けて行うシェルスクリプトを作成

① geditを起動し、test0.shというファイルを作成する

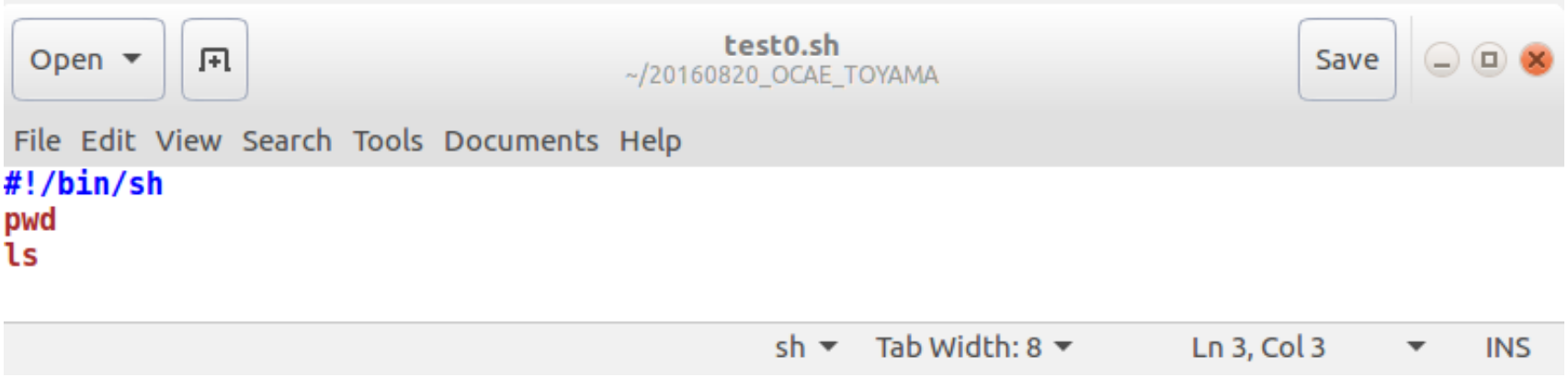
```
$ gedit test0.sh ⏎
```

② test0.shに以下の内容を記述する

test0.shの内容

```
1  #!/bin/sh
2  pwd
3  ls
```

← シェルスクリプトを書くときのお約束．1行目に記述



操作：シェルスクリプトを作成・実行（test0.sh, 2）

コマンドlsとpwdを続けて行うシェルスクリプトを作成

③ lsを実行し、ファイルができていることを確認する

```
$ ls ⏎
test0.sh
```

④ test0.shを実行する

```
$ ./test0.sh ⏎
bash: ./test0.sh: Permission denied
```

計算が実行できないことを確認。。。。。

原因はファイルに実行権限がない

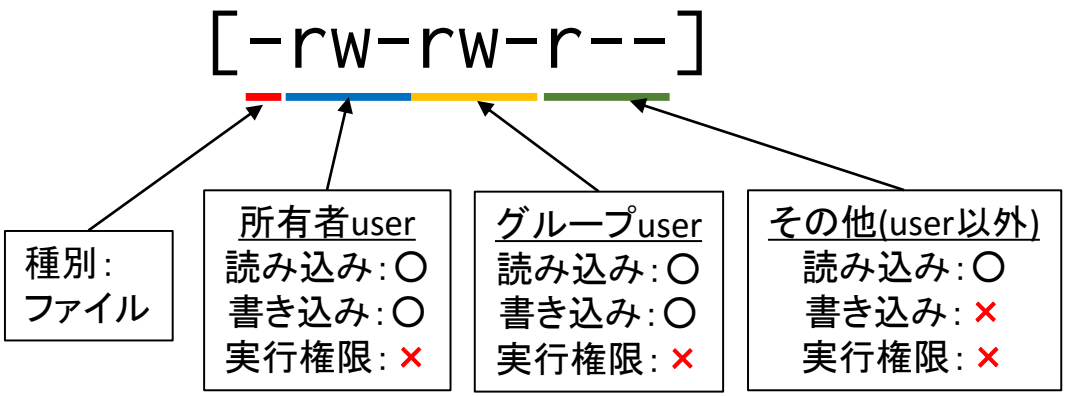
操作：シェルスクリプトを作成・実行（test0.sh, 3）

コマンドlsとpwdを続けて行うシェルスクリプトを作成

⑤ ls -lを実行し、ファイルの詳細を確認する

```
$ ls  -l ⏎
-rw-rw-r--  1 user user   17    X月  X    XX:XX  test0.sh
```

パーミッション（ファイルのアクセス権）の読み方



1文字目はファイル種別
[-] ファイル
[d] ディレクトリ
[l] シンボリックリンク

2文字目から4文字目はファイルの所有者に対する権限
5文字目から7文字目はファイルの所有グループに対する権限
8文字目から10文字目はその他に対する権限
[r] 読み取り
[w] 書き込み
[x] 実行
[-] 権限なし

操作：シェルスクリプトを作成・実行（test0.sh, 4）

コマンドlsとpwdを続けて行うシェルスクリプトを作成

⑥ コマンドchmodを実行し、ファイルに実行権限付加する

```
$ chmod +x test0.sh ⏎
$ ls -l ⏎
-rwxrw-r--  1 user user   17    X月  X    XX:XX  test0.sh
```

⑦ test0.shを実行する

```
$ ./test0.sh ⏎
/home/user/20160820_OCAE_TOYAMA
test0.sh
$
```

まとめ：シェルスクリプトを作成した際にはファイルの実行権限の有無に注意する

操作：シェルスクリプトを作成・実行（test1.sh）

Hello World!を出力させるtest1.shというファイルを作成する

test1.sh

```
1  #!/bin/sh
2  echo "Hello World!"
```

```
$ chmod +x test1.sh ⏎
$ ./test1.sh ⏎
Hello World!
```

echoコマンド
引数に指定された文字列や変数の内容を表示する
文字列の場合は、表示したい文字列を[“（ダブルクォート）]で囲む

操作：シェルスクリプトを作成・実行（test2.sh）

コメントについて

test2.sh

```
1  #!/bin/sh
2  #echo "Hello World!"
3
4  <<COMMENT
5  echo "Hello world!"
6  echo "Hello world!"
7  echo "Hello world!"
8  COMMENT
```

コメントは, #を使用し、
#の後ろに書かれた文字はコメントとされ、読まれない
#は一行のみコメント化される

複数行コメントしたい場合は,
[<< + 任意の文字列]を使用する

```
$ chmod +x test2.sh ⏎
$ ./test2.sh ⏎
$
```

操作：シェルスクリプトを作成・実行（test3.sh）

変数について

test3.sh

```
1   #!/bin/sh
2
3   string="Hello world!"
4
5   echo string
6   echo 'string'
7   echo "string"
8   echo
9   echo $string
10  echo '$string'
11  echo "$string"
12  echo
13  echo ${string}
14  echo '${string}'
15  echo "${string}"
```

```
$ chmod +x test3.sh ⏎
$ ./test3.sh ⏎
string
string
string

Hello world!
$string
Hello world!

Hello world!
${string}
Hello world!
```

基本的に、変数の内容を使いたいときは、$の後に変数名を書くことで使うことができる.
“(ダブルクォート)で囲むときは変数が展開され、‘(シングルクォート)で囲むときは変数が展開されない.

操作：シェルスクリプトを作成・実行（test4.sh）

変数について

test4.sh

```
1   #!/bin/sh
2
3   num=4
4
5   echo num
6   echo 'num'
7   echo "num"
8   echo
9   echo $num
10  echo '$num'
11  echo "$num"
12  echo
13  echo ${num}
14  echo '${num}'
15  echo "${num}"
```

```
$ chmod +x test4.sh ⏎
$ ./test4.sh ⏎
num
num
num

4
$num
4

4
${num}
4
```

操作：シェルスクリプトを作成・実行（test5.sh）

変数について

test5.sh

```
1   #!/bin/sh
2
3   name=Ichiro
4   age=18
5
6   echo '$name is $age years old.'
7   echo "$name is $age years old."
8   echo
9   printf '%s is %d years old.¥n' $name $age
10  printf "%s is %d years old.¥n" $name $age
```

```
$ chmod +x test5.sh ⏎
$ ./test5.sh ⏎
$name is $age years old.
Ichiro is 18 years old.

Ichiro is 18 years old.
Ichiro is 18 years old.
```

printfコマンド

printfコマンドはデータを整形して出力する
echoコマンドより詳細を指定できる

補足：文字列操作のいろいろ

ファイルのパスを操作するときに利用される

PATHNAME=/home/user/20160820_OCAE_TOAYAMA/long.file.name.txt

記述例	出力
${PATHNAME##/*/}	long.file.name.txt
${PATHNAME#/*/}	user/20160820_OCAE_TOAYAMA/long.file.name.txt
$PATHNAME	/home/user/20160820_OCAE_TOAYAMA/long.file.name.txt
${PATHNAME%.*}	/home/user/20160820_OCAE_TOAYAMA/long.file.name
${PATHNAME%%.*}	/home/user/20160820_OCAE_TOAYAMA/long
${PATHNAME%.txt}.dat	/home/user/20160820_OCAE_TOAYAMA/long.file.name.dat

数値・変数の計算 ―演算―

計算したい式を `$((` と `))` で囲う

test6.sh

```
1  #!/bin/sh
2
3  x=2
4  y=12
5  z=$(( x + y ))
6
7  echo "x = $x"
8  echo "y = $y"
9  echo "x + y = $x + $y = $z"
```

```
$ chmod +x test6.sh ⏎
$ ./test6.sh ⏎
x = 2
y = 12
x + y = 2 + 12 = 14
```

引数を利用したシェルスクリプト・引数処理に使用する変数

シェルスクリプト実行時に指定された引数を扱うための変数が用意されている

変数名	自動的に設定される値
$#	実行時に指定された引数の数を表す変数 「$ ./test.sh abc def ghi」と実行された場合、シェルスクリプト test.sh 内で変数 $# を参照するとその値は 3 となる
$@	シェルスクリプト実行時に指定された全パラメータが設定される変数 変数 $* と基本的に同じだが、 “” で囲んだときの動作が異なる
$*	シェルスクリプト実行時に指定された全パラメータが設定される変数 変数 $@ と基本的に同じだが、 “” で囲んだときの動作が異なる
$0	実行時のコマンド名が設定される変数 「./test.sh」と実行した場合には「./test.sh」が、 「/home/user/test.sh」と実行した場合には「/home/user/test.sh」が設定される
$1 – $n	シェルスクリプト実行時に指定した引数の値がそれぞれ設定される変数 1番目に指定した引数は $1 に、2番目に指定した引数は $2 に、n 番目に指定した引数は $n に設定される。10番目以降の引数参照時は ${10} のように {} を使用する必要がある。これは $10 を $1 "0" のように、シェルに誤った解釈をされることを防ぐためである

引用URL：http://shellscript.sunone.me/parameter.html

操作：シェルスクリプトを作成・実行（test7.sh）

引数について

test7.sh

```
1   #!/bin/sh
2
3   echo "script name: $0"
4   echo "arg num : $#"
5   echo "arg 1   : $1"
6   echo "arg 2   : $2"
7   echo "arg 3   : $3"
8   echo "arg 4   : $4"
9   echo "arg 5   : $5"
10  echo "arg 6   : $6"
11  echo "arg 7   : $7"
12  echo "arg 8   : $8"
13  echo "arg 9   : $9"
14  echo "arg 10? : $10"
15  echo "all arg : $*"
16  echo "all arg2 : $@"
```

```
$ ./test7.sh a b c d e f g h i j k l
script name: ./cmd.sh
arg num : 12
arg 1   : a
arg 2   : b
arg 3   : c
arg 4   : d
arg 5   : e
arg 6   : f
arg 7   : g
arg 8   : h
arg 9   : i
arg 10? : a0
all arg : a b c d e f g h i j k l
all arg2:  a b c d e f g h i j k l
$
```

関数の定義

たびたび繰り返されるコマンド群は、関数として定義することで見やすく小さなスクリプトを作ることができる

```
name ()
{
    コマンド
}
```

if文の定義（1） ―制御構文―

条件によって処理を分岐させる

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

if文の定義(2) ―制御構文―

条件の部分は[ ]で記述する
[ ]内の記述例は下表

記述例	説　　明
-e filename	ファイルが存在していれば真
-d filename	ディレクトリであれば真
-f filename	通常ファイルであれば真
-h filename	シンボリックリンクであれば真
-r filename	読むことが可能なファイルであれば真
-w filename	書き込み可能なファイルであれば真
-n string	文字列の長さが 0 でなければ真
-z string	文字列の長さが 0 であれば真
string	文字列で空でなければ真
s1 = s2	文字列s1 と文字列s2 が同じであれば真
n1 -eq n2	数値n1 と数値n2 が同じであれば真
n1 -ne n2	数値n1 と数値n2 が異なれば真
! expression	評価した式が偽であれば真
expr1 -a expr2	式expr1 と式expr2 が真であれば真
expr1 -o expr2	式expr1 と式expr2 のいずれかが真であれば真

case文の定義　ー制御構文ー

変数の値によって複数の処理に分岐させる

```
case $変数 in
    パターン1)
        命令1
        ;;
    パターン2)
        命令2
        ;;
    パターン3|パターン4)
        命令3
        ;;
    *)
        命令4
        ;;
esac
```

while文の定義　ー制御構文ー

ある条件が成立する間処理を繰り返す

```
while [ 条件 ]
do
    処理
done
```

コマンドをつなげる演算子 ー演算子ー

演算子	記述	説明	使用例
;	コマンド１; コマンド２	コマンドを連続して書くことができる コマンドは連続して実行される	pwd ; ls
&	コマンド &	コマンドをバックグラウンドジョブ（裏ジョブ）として実行	
&&	コマンド１&& コマンド２	コマンド１を実行し、正常に終了した場合、コマンド２を実行する	pwd && ls pwdは正常に終了するためlsは実行される
||	コマンド１|| コマンド２	コマンド１を実行し、異常終了した場合、コマンド２を実行する	pwd && ls pwdは正常に終了するためlsは実行されない
|	コマンド１| コマンド２	コマンド１の標準出力をコマンド２に渡す	cat file名 | grep

補足：括弧( )の意味について

```
$ cd /home/user/20160820_OCAE_TOYAMA⏎
$gedit test8.sh ⏎
```

test8.sh

```
#!/bin/sh

( cd ~/  ; pwd )
pwd
```

```
$ chmod +x test8.sh⏎
$./test8.sh ⏎
/home/user
/home/user/20160820_OCAE_TOYAMA
$
```

(　)の中で実行したコマンドは別プロセスで実行されるため、
起動したスクリプト内には何も影響を与えない

引用URL:http://qiita.com/yohm13/items/3527d517768402efbcb6

その３

OpenFOAMで使用されるシェルスクリプトを見てみる

OpenFOAMで使用されるシェルスクリプト

Tutorialで使用されるものとしては

Allrun	計算に必要な手続きを記述
Allclean	ケースファイルを計算前の状態に（初期化）する手続きを記述

したがって、Allrunがあるケースファイルは、取りあえずAllrunを実行することで、どんな計算が行われているかを知ることができる。

Allrunの内容をみることで、計算に必要な手続きを知ることができる

解読できるようになれば、やっていることの理解が進むはず！？

Allrun, Allcleanの例（１）- Solver:rhoCentralFoam, Tutorial:shockTube-

準備

⓪ OpenFOAM®の環境を端末に読み込む

```
$ of301⏎
```

① $FOAM_TUTORIALS/compressible/rhoCentralFoam/shockTubeを
/home/user/20160820_OCAE_TOYAMAにコピーする

```
$ cp -rf $FOAM_TUTORIALS/compressible/rhoCentralFoam/shockTube␣
/home/user/20160820_OCAE_TOYAMA ⏎
```

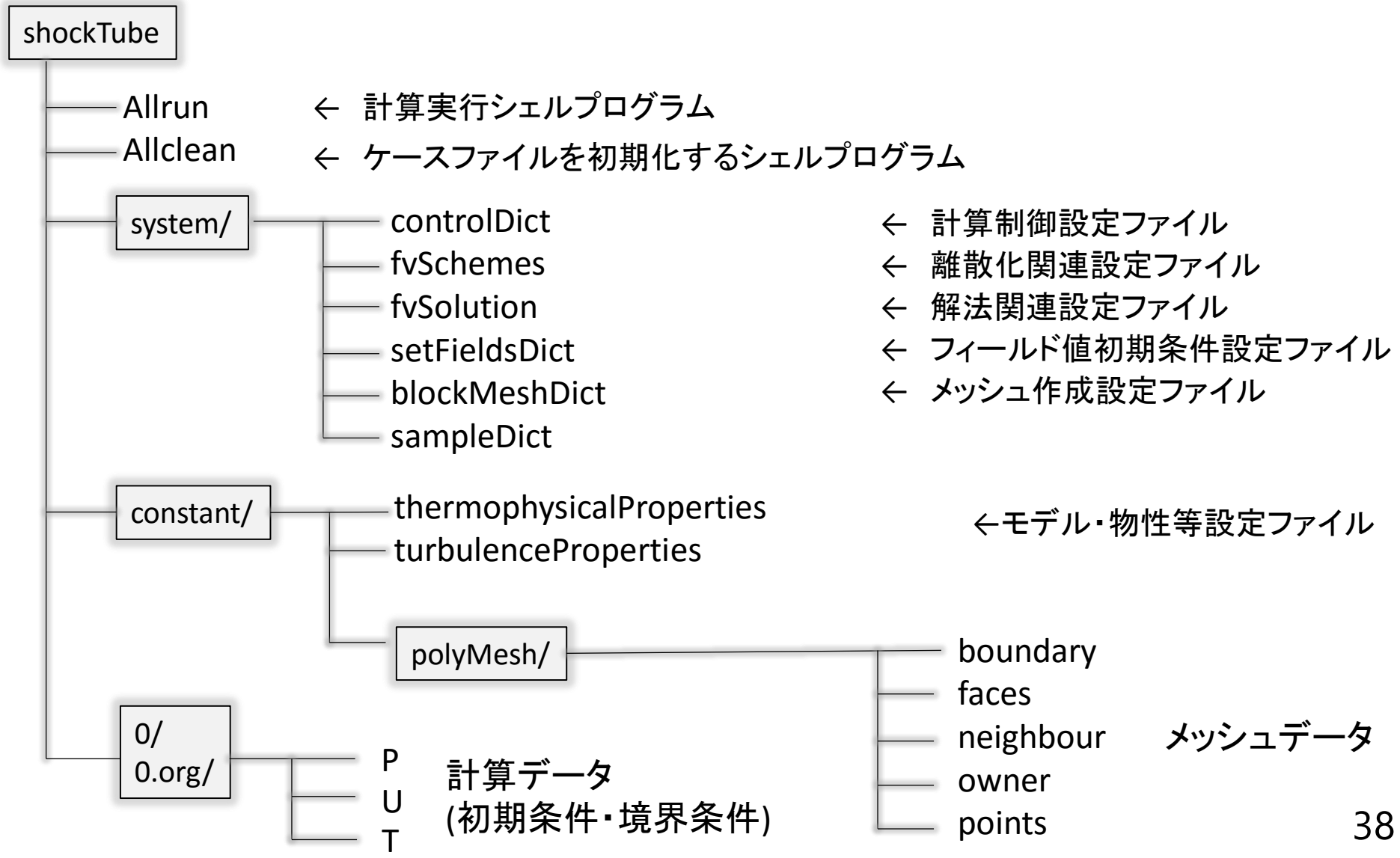
注意：２行で書かれているが１行で続けて入力

② ディレクトリを移動する

```
$ cd    /home/user/20160820_OCAE_TOYAMA/shckTube ⏎
```

Allrun, Allcleanの例（2）- Solver:rhoCentralFoam, Tutorial:shockTube-

ファイル構成 (ケースファイル)



Allrun, Allcleanの例（3）- Solver:rhoCentralFoam, Tutorial:shockTube-

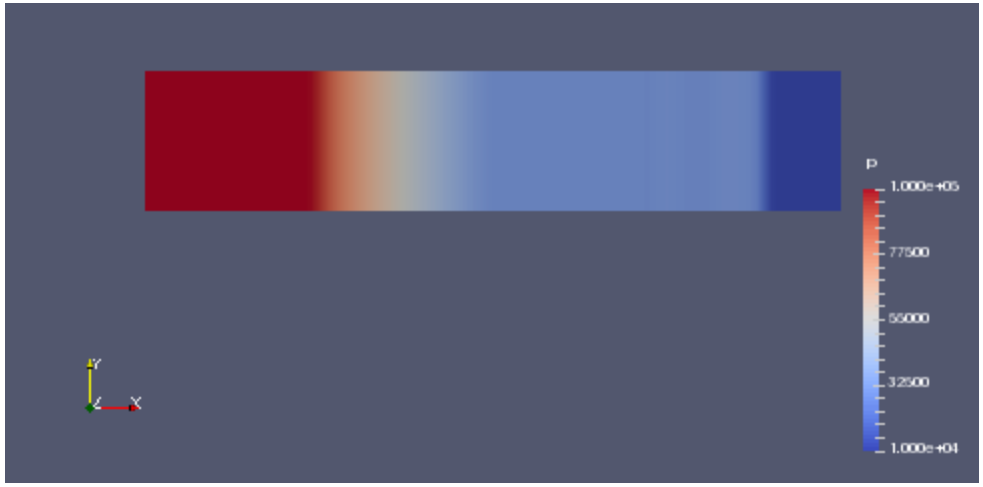
取りあえず計算を実行してみる

① Allrunを実行

```
$ ./Allrun⏎
Running blockMesh on /home/user/20160820_OCAE_TOYAMA/shockTube
Running setFields on /home/user/20160820_OCAE_TOYAMA/shockTube
Running rhoCentralFoam on /home/user/20160820_OCAE_TOYAMA/shockTube
$
```

② paraFoamを実行して結果を表示してみる

```
$ paraFoam ⏎
```



Allrun, Allcleanの例（4）- Solver:rhoCentralFoam, Tutorial:shockTube-

取りあえず計算を実行してみる

③ lsコマンドでファイルの確認

```
$ ls⏎
0       0.003  0.006   Allclean   log.blockMesh        system
0.001   0.004  0.007   Allrun     log.rhoCentralFoam
0.002   0.005  0.org   constant   log.setFields
```

計算データの追加

ログファイルの追加

④ Allcleanの実行

```
$ ./Allclean⏎
Cleaning /home/user/20160820_OCAE_TOYAMA/shockTube case
$ ls⏎
0       0.org   Allclean  Allrun  constant  system
```

初期化されたことを確認

ここまでのまとめ

Allrunを実行すると
１. blockMesh→setFields →rhoCentralFoamの順にコマンドが実行される
２. コマンドごとにログファイルが生成される

Allcleanを実行すると
計算結果とログファイルが削除され、初期化される（現段階ではこの認識でいきます）

演習：myAllrunを作成（１）

実際にAllrunを見る前に。。。。

myAllrunを作成し、同じ結果を出す

１. blockMesh→setFields →rhoCentralFoamの順にコマンドが実行される
２. コマンドごとにログファイルが生成される
３.端末に現在行っているコマンド名を表示する
　ex. Running blockMesh on /home/user/20160820_OCAE_TOYAMA/shockTube

演習：myAllrunを作成（2）

1. blockMesh→setFields →rhoCentralFoamの順にコマンドが実行される
2. コマンドごとにログファイルが生成される
3. 端末に現在行っているコマンド名を表示する
   ex. Running blockMesh on /home/user/20160820_OCAE_TOYAMA/shockTube

myAllrun

```
#!/bin/sh
echo “Running blockMesh on /home/user/20160820_OCAE_TOYAMA/shockTube”
blockMesh > log.blockMesh
echo “Running setFields on /home/user/20160820_OCAE_TOYAMA/shockTube”
setFields > log. setFields
echo “Running rhoCentralFoam on /home/user/20160820_OCAE_TOYAMA/shockTube”
rhoCentralFoam > log.rhoCentralFoam
```

作成したらmyAllrunを実行して結果を確認する

＃実行権限の付加を忘れずに

Allrun（1）

Allrunはとてもスマートに記述されている

Allrun

```
#!/bin/sh
cd ${0%/*} || exit 1    # Run from this directory

# Source tutorial run functions
. $WM_PROJECT_DIR/bin/tools/RunFunctions

runApplication blockMesh
runApplication setFields
runApplication `getApplication`
```

myAllrun

```
#!/bin/sh
echo “Running blockMesh on /home/user/20160820_OCAE_TOYAMA/shockTube”
blockMesh > log.blockMesh
echo “Running setFields on /home/user/20160820_OCAE_TOYAMA/shockTube”
setFields > log. setFields
echo “Running rhoCentralFoam on /home/user/20160820_OCAE_TOYAMA/shockTube”
rhoCentralFoam > log.rhoCentralFoam
```

解説は次ページから

Allrun（2）

Allrun

```
2    cd ${0%/*} || exit 1    # Run from this directory
```

解説

ケースファイルがあるディレクトリに移動する
cdコマンドが失敗した場合exitコマンドを実行しエラーコード1（異常終了）とする

Allrun（3）

Q. 何故このように記述するのか　　A. どの場所であっても計算を実行できるようにするため

/home/user/20160820_OCAE_TOYAMA/shockTubeで実行する場合

```
$ pwd⏎
/home/user/20160820_OCAE_TOYAMA/shockTube
$ ./Allrun⏎
```

環境変数$0は　[./Allrun]　が入る

環境変数${0%/*} は　[.]　が入る

cd ${0%/*}はcd .　となり、場所は移動しない

/home/user/20160820_OCAE_TOYAMA/で実行する場合

```
$ pwd⏎
/home/user/20160820_OCAE_TOYAMA
$ ./shockTube/Allrun⏎
```

環境変数$0は　[./shockTube/Allrun]　が入る

環境変数${0%/*} は　[./shockTube]　が入る

cd ${0%/*}はcd ./shockTube　となり、shockTubeディレクトリに移動する

Allrun（4）

Allrun

```
4   # Source tutorial run functions
5   . $WM_PROJECT_DIR/bin/tools/RunFunctions
6
7   runApplication blockMesh
8   runApplication setFields
9   runApplication `getApplication`
```

5行目

カレントディレクトリに$WM_PROJECT_DIR/bin/tools/RunFunctionsを読み込む

目的

関数runApplication, getApplicationを使用するため

Allrun（5）

Allrun

```
7   runApplication blockMesh
```

$WM_PROJECT_DIR/bin/tools/RunFunctions（OF301の場合）

```
41  runApplication()
42  {
43      APP_RUN=$1                  #APP_RUN=$1=blockMesh
44      APP_NAME=${1##*/}           #APP_NAME=${1##*/}=blockMesh
45      shift                       #引数を１つシフトする($1=" ")
46
47      if [ -f log.$APP_NAME ] #条件分岐：log.blockMeshが存在すると真
48      then
49          echo "$APP_NAME already run on $PWD: remove log file to re-run"
50      else
51          echo "Running $APP_RUN on $PWD"
52          $APP_RUN "$@" > log.$APP_NAME 2>&1 #blockMesh > log.blockMesh  2>&1
53      fi
54  }
```

*フルパスでコマンドを記述する場合に意味が出てくる

解説

関数runApplication コマンド名は

コマンド名 > log.コマンド名 2>&1

のコマンドを実行する

Allrun（6）

Allrun

```
9   runApplication `getApplication`
```

$WM_PROJECT_DIR/bin/tools/RunFunctions（OF301の場合）

```
36  getApplication()
37  {
38      sed -ne 's/^ *application¥s*¥([a-zA-Z]*¥)¥s*;.*$/¥1/p' system/controlDict
39  }
```

system/controlDict

```
18  application     rhoCentralFoam;
```

解説

- 関数getApplicationは、system/controlDictに記述されるソルバー名（rhoCentralFoam）を出力する
- `getApplication`とすることで変数として扱っている

Allrun（7）

```
sed -ne 's/^ *application¥s*¥([a-zA-Z]*¥)¥s*;.*$/¥1/p' system/controlDict
```

sed [コマンドオプション] ‘[範囲指定] p’ [入力ファイル]

$ sed -ne ‘/hoge/ p’ inputFile # inputFile内のhogeが含まれる行だけを標準出力

青色下線部分について（1）

```
's/^ *application¥s*¥([a-zA-Z]*¥)¥s*;.*$/¥1/p'
```

置換コマンド

sed -e “s/置換条件/置換文字/g”

が基本．つまり

＊最後にgを付けた場合は置換条件に当てはまるすべての文字列が置換される

```
's/^ *application¥s*¥([a-zA-Z]*¥)¥s*;.*$/¥1/p'
```

置換条件　　置換文字

Allrun（8）

```
sed -ne 's/^ *application¥s*¥([a-zA-Z]*¥)¥s*;.*$/¥1/p' system/controlDict
```

青色下線部分について(2)　置換条件

```
's/^ *application¥s*¥([a-zA-Z]*¥)¥s*;.*$/¥1/p'
```

先頭　　後尾

application　rhoCentralFoam　;

^ *	application	¥s*	¥([a-zA-Z]*¥)	¥s*	;	.*$
先頭から0個以上のスペース	文字列 application	0個以上のスペース	大文字小文字の任意の文字列		文字 ;	任意の文字（後尾まで）

Allrun（9）

```
sed -ne 's/^ *application¥s*¥([a-zA-Z]*¥)¥s*;.*$/¥1/p' system/controlDict
```

青色下線部分について(3)

```
's/^ *application¥s*¥([a-zA-Z]*¥)¥s*;.*$/¥1/p'
```

(　)でくくれば、置換後文字列で¥1,¥2,¥3・・・のように部分的に再利用できる
ただし(　)はそれぞれ¥(, ¥) とエスケープ文字を付加する必要がある

置換前	application	rhoCentralFoam	;
置換後	rhoCentralFoam		

Allclean（1）

Allclean

```
1  #!/bin/sh
2  cd ${0%/*} || exit 1    # Run from this directory
3
4  # Source tutorial clean functions
5  . $WM_PROJECT_DIR/bin/tools/CleanFunctions
6
7  rm -rf 0
8  cp -r 0.org 0
9  cleanCase
```

解説

5行目
カレントディレクトリに$WM_PROJECT_DIR/bin/tools/CleanFunctionsを読み込む

9行目
関数cleanCaseにより計算データ、形状データ、後処理データ、ログデータを消去する

Allclean（2）

$WM_PROJECT_DIR/bin/tools/cleanFunctions（OF301の場合,一部省略）

```
71    cleanCase()
72    {
73        cleanTimeDirectories          #時刻データの削除
74        cleanDynamicCode              #functionObject使用で生成されるDynamicCodeを削除
75
76        rm -rf processor* > /dev/null 2>&1
77        rm -rf postProcessing > /dev/null 2>&1
78        rm -rf probes* > /dev/null 2>&1
79        rm -rf forces* > /dev/null 2>&1
80        rm -rf graphs* > /dev/null 2>&1
81        rm -rf sets > /dev/null 2>&1
82        rm -rf surfaceSampling > /dev/null 2>&1
83        rm -rf cuttingPlane > /dev/null 2>&1
84        rm -rf system/machines > /dev/null 2>&1
85
86        if [ -d constant/polyMesh ]               #メッシュデータを削除する部分
87        then
88            (cd constant/polyMesh && ¥
89                rm -rf ¥
90                allOwner* cell* face* meshModifiers* ¥
91                owner* neighbour* point* edge* ¥
92                cellLevel* pointLevel* refinementHistory* level0Edge* surfaceIndex* sets ¥
93                > /dev/null 2>&1 ¥
94            )
95        fi
...   . . . . . .
111   }
```

Allclean（3）

$WM_PROJECT_DIR/bin/tools/cleanFunctions（OF301の場合,一部省略）

```
45  cleanTimeDirectories()
46  {
47      echo "Cleaning $PWD case"
48      zeros=""
49      while [ ${#zeros} -lt 8 ]
50      do                                                          #時刻データの削除
51          timeDir="0.${zeros}[1-9]*"
52          rm -rf ./${timeDir} ./-${timeDir} > /dev/null 2>&1
53          zeros="0$zeros"
54      done
55      rm -rf ./[1-9]* ./-[1-9]* ./log ./log.* ./log-* ./logSummary.* ./.fxLock ./*.xml ./ParaView*
    ./paraFoam* ./*.OpenFOAM ./*.blockMesh ./.setSet > /dev/null 2>&1          #関連データの削除
56  }
57
58
59  #
60  # Remove dynamicCode subdirectory if it looks appropriate
61  #
62  cleanDynamicCode()
63  {
64      if [ -d system -a -d dynamicCode ]
65      then
66          rm -rf dynamicCode > /dev/null 2>&1                 #dynamicCodeでディレクトリの削除
67      fi
68  }
```

演習

$FOAM_TUTORIALS/incompressible/icoFoam/cavityにAllrunファイルを作成し実行する

① $FOAM_TUTORIALS/incompressible/icoFoam/cavityを
/home/user/20160820_OCAE_TOYAMAにコピーする

```
$ cp -rf $FOAM_TUTORIALS/incompressible/icoFoam/cavity
/home/user/20160820_OCAE_TOYAMA ⏎
```

注意：2行で書かれているが1行で続けて入力

② ディレクトリを移動する

```
$ cd    /home/user/20160820_OCAE_TOYAMA/cavity ⏎
```

③ スライド44の内容のAllrunファイルを作成し実行する
＃ただし、8行目は削除する

まとめ

OpenFOAM®で使用されるシェルスクリプト
Allrun, Allcleanを解説した

講習を通じて、OpenFOAM®で使用するコマンドや、シェルスクリプトを解読する作業を知ることで、プログラムが何をしているのかということを自力で調べるようになるきっかけになれば幸いです

みなさんお疲れ様でした！！

参考文献

シェルスクリプト入門 書き方のまとめ
http://motw.mods.jp/shellscript/tutorial.html

UNIX & Linux コマンド・シェルスクリプト リファレンス
http://shellscript.sunone.me