Linux入門 (2)

正田 備也

masada@rikkyo.ac.jp

参考資料

Linux入門講座 （エンジニアの入り口）

https://eng-entrance.com/category/linux

Linux基本コマンドTips （atmarkIT）

https://www.atmarkit.co.jp/ait/series/3065/

Linuxコマンド(4/4)

テキスト処理

正規表現

演習

- wgetコマンドで、以下のWebページをWikipediaから取得する。
  - https://en.wikipedia.org/wiki/Category:21st-century_American_male_actors
    - これは、アメリカの男性俳優一覧を取得するときに、起点とすることができるページ。
- このページの上のほうに、family nameの最初の2文字に従って俳優を振り分けて記録してある多数のWebページへのリンクがある。
- これらのリンクのURLを、Linuxコマンドだけで、すべて抽出するには、どうすればいいか？

```
$ cat Category¥:21st-century_American_male_actors | grep "https"
```

printf

- データを整形して表示するコマンド
  - C言語のprintfを知っていると分かりやすいかも
- 「printf フォーマット 引数1 引数2 ・・・」と使う
  - 文字列や数値など表示したいものを引数としてじかに書く

```
$ printf "%s¥n" apple orange banana
$ printf "%s " apple orange banana
$ printf "%05d¥n" 321 1026 6445555
$ printf "%9.3f¥n" 1.4 102.2362 3141592.65359
```

xargs

- 標準入力やファイルからリストを読み込んでコマンドラインを作成し、実行する
- 使い方は「コマンドA | xargs コマンドB」
  - xargsは、コマンドAの出力を受け取り、行ごとに分ける
  - そして、各行に入っている文字列を、そのまま、コマンドBの引数として渡す

```
$ ls /etc/*.conf | xargs head -1
$ ls /etc/*.conf | xargs printf "%40s¥n"
```

grep

- grepコマンドはファイルの中の文字列検索に使われる
- 特定の文字列を含む行をすべて標準出力に出力する
- 例：ファイルsample.txtからappleという文字列を含む行を出力

```
$ cat sample.txt | grep apple
```

- 正規表現を使うこともある
  - grepは正規表現にマッチする文字列を含む行をすべて出力する

正規表現 (regular expression)

- 特定のパターンの文字列を検索するために使う
  - 「文字列の集合を一つの文字列で表現する方法の一つ」（Wikipedia）
- 以下の特殊文字を使う
  - これらを「メタ文字」と呼ぶこともある

. ^ $ [ ] * + ? | ( )

メタ文字「.」

- 「.」なんでもいい1文字にマッチする
  - 「1.3」という正規表現を使ってgrepで文字列検索すると・・・
  - 「123」「1z3」「133」などを含めばマッチする

- 「1.3」のようなピリオドを含む文字列を検索したいときは？
  - 「1¥.3」という正規表現を使う
  - このような「¥」（バックスラッシュ）を、エスケープ文字と呼ぶ
  - 「¥」の直後にくる文字は、そのままその文字として検索される

メタ文字「^」と「$」

- 「^」は行の先頭、「$」は行の最後とマッチする

- 「^1.3」という正規表現を使ってgrepで文字列検索すると…
  - 先頭の文字が「1」で、次の文字は何でもよく、
  - さらに次の文字が「3」であるような行が検索される

- 「1.3$」という正規表現で文字列検索すると…
  - 末尾の文字が「3」で、その手前の文字は何でもよく、
  - さらに手前の文字が「1」であるような行が検索される

メタ文字「*」

- 直前の文字（or 表現）が0個以上続いているときマッチする
  - 「12*3」は「13」「123」「1222223」などを含めばマッチする

メタ文字「+」「?」

- 「+」と「?」はegrepコマンド（あるいはgrep -E）で使える
  - man re_formatを参照

- 「+」は、直前の文字（or 表現）が1個以上続いているときマッチ
  - 「12+3」は「123」「1222223」などを含めばマッチする
  - 「12+3」は「13」にはマッチしない

- 「?」は、直前の文字（or 表現）が0個または1個あるときマッチ
  - 「12?3」は「123」を含めばマッチする

メタ文字「[」「]」

- 「[abc]」は、abcのどれかひとつを含めばマッチする
- 「[^abc]」は、abcのどれも含まなければマッチする
- 「[a-z]」は、ASCIIコード順でaからzの間の文字を含めばマッチする
- 「[^0-9]」は、ASCIIコード順で0から9の間の文字をどれも含まなければマッチする

メタ文字「(」「)」「|」

- egrepコマンド（あるいはgrep -E）で使える
- 「(abc)」は、abcを含めばマッチする（「abc」と同じ）
- 「(abc|xyz)」は、abcまたはxyzを含めばマッチする
  - 「(gray|grey)」は「gr(a|e)y」と同じ文字列にマッチする
- 「(abc)+」は、abcの1回以上の繰り返しを含めばマッチする
  - このように「+」は、1文字以外の表現の繰り返しを表すためにも使える

sed

- stream editorの意
- 「sed スクリプトコマンド ファイル名」と使う
- 与えられたテキストを、コマンドに従って処理し、標準出力へ出力する
  - テキストは、ファイル名を指定することによってか…
  - パイプやリダイレクトによって与える
- コマンドの部分に正規表現が使える

```
$ echo "abracadabra" | sed "s/ab/AB/g"
```

sedはとても多機能

- マニュアルを見よう
  - https://www.gnu.org/software/sed/manual/sed.html
  - 特に「5.3 Overview of basic regular expression syntax」のあたり

sedによる置き換え

- sコマンドは置換処理をおこなう

```
$ echo "abracadabra" | sed 's/a/x/g'
$ echo "abracadabra" | sed 's/a/x/'
$ echo "abracadabra" | sed 's/ab/x/g'
$ echo "abracadabra" | sed 's/^ab/x/g'
$ echo "abracadabra" | sed 's/[b|d]/x/g'
$ echo "abracadabra" | sed 's/a[b|d]/x/g'
$ echo "abracadabra" | sed 's/ra$/x/g'
$ echo "abracadabra" | sed 's/[rb]/x/'
$ echo "abracadabra" | sed 's/[^a]/x/g'
$ echo "abracadabra" | sed 's/[^a]¥+/x/'
```

awk

- 名称は開発者であるAho, Weinberger, Kernighanの頭文字
- プログラミング言語なので複雑なことができるが・・・
- ここでは、ワンライナーでの利用に限って説明する

```
$ ps axu | awk '{print $1}'
```

awkを使ったワンライナー

- 標準入力からテキストを受け取って、各行に対して同じ処理を適用する
- 基本的な構造は以下のとおり

```
awk 'BEGIN{テキストを読む前の処理}{各行に対する処理}END{テキストを読んだ後の処理}'
```

awkの使い方の例（1/4）

```
$ ps axu | awk '{print $1}'
```

- 各行の最初のフィールドを出力する
  - デフォルトのフィールド・セパレータは、空白またはタブ
- `$0`は行全体を表す

```
$ ps axu | awk '{print NF}'
```

- 各行のフィールドの個数を表示する
  - `NF`は特殊な変数で、各行のフィールドの個数を保持している

awkの使い方の例（2/4）

```
$ ps axu | awk 'BEGIN{s=0}{if($1=="root"){s+=$4}}END{print s}'
```

- 第1フィールドが"root"なら、第4フィールドを数値として解釈し、変数sに加える
  - awkでは変数も使える
- テキストを読み終わった後で、その総和を表示
  - BEGIN{s=0}は無くても大丈夫。変数は適切に初期化される。

awkの使い方の例（3/4）

```
$ ps axu | tail -n +2 | awk '{a[$2]=$1}END{for(k in a){print k,a[k]}}'
```

- awlでは、連想配列も使える
- 第2フィールドをキー、第1フィールドを値としている
  - tailコマンドでps axuの出力の2行目からだけを取り出している
- テキストを読んだ後で、すべてのキーについて、その値を表示している

awkの使い方の例（4/4）

```
$ ls -l /etc | awk '{n=split($NF,a,".");if(n>1){print a[1]}else{print $NF}}'
```

- 最後のフィールドを"."で分割してできる部分文字列が２個以上なら１個目の部分文字列を表示

- そうでなければ最後のフィールドそのものを表示
  - split()のように、awkコマンドの中で使える関数がいろいろとある

bashスクリプト

コマンドを組み合わせて"プログラム"を書く

変数、式の評価

配列

分岐、ループ

bashスクリプト

- この授業ではシェルとしてbashを使っていると想定
- bashではコマンドを組み合わせてスクリプト（プログラムのようなもの）を書ける
- 以下の内容をエディタで作成、適当なファイル名（例えば`hello.sh`）で保存

```
#!/bin/bash
echo 'Hello World!'
```

- `echo`コマンドで文字列を表示し、`exit 0`で正常終了（`exit 1`が異常終了）

bashスクリプトの実行

- hello.shをbashスクリプトとして実行するときは、以下のように入力

```
$ bash hello.sh
```

同じ行に複数の命令を書く

- 同じ行に複数の命令を書くときは「;」で区切る
- 以下の２つのスクリプトは同じことをする

```
#!/bin/bash
cd ~/ ; ls | head -1 ; ls | tail -1 ; cd -
```

```
#!/bin/bash
cd ~/
ls | head -1
ls | tail -1
cd -
```

- 「cd -」は、直前にいたディレクトリに戻る

変数 (1/2)

- 例えば、いま使っているシェルがどのシェルかを確認するには、次のようにする

```
$ echo $SHELL
```

  - 「$」の後にある「SHELL」は変数の名前
  - 「$」を前に付けることで、「$SHELL」の部分が変数の値で置き換えられる
    - この例にある「SHELL」は環境変数で、システムの設定ですでに値が代入されている変数。

- 以下のように自分で新しい変数を用意することもできる

```
$ a=1000
$ echo $a
```

  - 変数の後の「=」の前後には空白を入れないこと

変数 (2/2)

- 変数には文字列も代入できる

```
$ s="hello world"
$ echo $s
$ echo "$s"
$ echo "${s}"
```

  - 上の３通りのechoコマンドは同じものを表示する
  - ただし、最後の２つが、安全

```
$ x=10
$ echo "$x+2"
$ echo "`expr $x + 2`"
```

  - 「`echo $x+2`」の「`$x+2`」の部分は、ただの文字列になってしまう
  - 「`」でくくったところは、その中を実行した結果でその部分が置き換えられる

backticks(``)によるcommand substitution

- backticksで囲んだ部分は、その中にあるコマンドを実行した結果で置き換えられる

```
$ a="`date`"
$ echo "$a"
```

- 同じことは、backticksを使わなくても、$()を使えば、実現できる

```
$ a="$(date)"
$ echo "$a"
```

* bashの特殊な変数「@」

- 以下のスクリプトを、例えば**my-ls.sh**というファイル名で保存

```
#!/bin/bash
ls -al "$@"
```

- そして、以下のように実行してみる

```
$ bash my-ls.sh `ls /etc/*.conf`
```

- 「@」は、引数すべてを半角スペースで区切ってつなげたものに置き換えられる

  - https://www.gnu.org/software/bash/manual/html_node/Special-Parameters.html

exprコマンド

- 与えられた式の評価をするコマンド

```
$ expr 1 + 2
$ expr 1+2
$ expr 10 - 13
$ expr 5 ¥* 3
$ expr 10 / 3
$ expr 10.0 / 3
$ x=21
$ expr 10 - ¥( $x / 3 ¥)
```

  - 演算子の前後には空白が必要
  - 掛け算の記号「*」や丸括弧「(」「)」はバックスラッシュ「¥」でエスケープする
  - 整数でない値はエラーになる
    - 他の使い方は、ググって調べてください

* 配列

```
$ a=("apple" "banana" "orange")
$ echo $a
$ echo $a[1]
$ echo ${a[1]}
$ a[2]="mikan"
$ echo ${a[@]}
```

- 値を使うときは、添え字を指定する部分も含めて、「{」「}」でくくる必要がある

```
$ a=(`date +%Y%m%d%H%M%S`)
$ echo ${a[@]}
$ a+=(`date +%Y%m%d%H%M%S`)
$ echo ${a[@]}
```

- dateコマンドは時刻を表す文字列を得るコマンド（次のスライド参照）
- 「+=」を使うと配列の末尾に要素を追加できる

dateコマンド

- 時刻を表す文字列を得るコマンド
- オプション「--date="文字列"」で表示したい時刻を指定
  - 指定がなければ現在の時刻を表示
- システムの時刻をセットすることもできるが、やらないほうがいい
  - システムの時刻の管理には普通はNTPを使う
- 表示するときのフォーマットを指定できる

```
$ date +%<format-option>
```

if文

- 条件式は「[」と「]」で囲う（前後は空白を入れる）

```
$ x=10 ; if [ $x -gt 3 ] ; then echo "ok" ; else echo "no" ; fi
```

  - 「fi」で閉じること
  - 「if」で始まる部分、「then」で始まる部分、「else」で始まる部分は、「;」で区切る
  - 「-gt」はgreater thanの意（他に「-ge」「-lt」「-le」「-eq」「-ne」などがある）

  - else ifは「elif」と書く

whileループ

- 条件式は「[」と「]」で囲う
  - 「[」と「]」の前後は空白を入れる

```
$ i=10 ; while [ $i -ge 1 ] ; do echo $i ; i=`expr $i - 1` ; done
```

  - whileループの中身は、doで始め、doneで閉じる
  - 以下の内容テキストファイルを作成し、コマンドラインから「bash ファイル名」としても同じことができる

```
#!/bin/bash
i=10
while [ $i -ge 1 ]
do
    echo $i
    i=`expr $i - 1`
done
```

while readループ

- 標準入力から受け取ったテキストの各行についてループを回して処理をするとき使う
  - パイプやリダイレクトと組み合わせて使う

```
$ ls -al | while read x ; do echo $x ; done
$ while read x ; do echo $x ; done < <(ls -al)
```

  - １つ目の「<」はリダイレクト、２つ目の「<」はプロセス置換
- readの後に複数の変数を書くと挙動が変わる

```
$ ls -al | while read x y z ; do echo $x ; done
$ ls -al | while read x y z ; do echo $y ; done
$ ls -al | while read x y z ; do echo $z ; done
```

forループ

- Pythonのように値のリストを使う場合は、以下のように書く

```
$ for x in "apple" "banana" "orange"; do echo $x ; done
```

- C言語のように整数の値を一定の範囲で走らせるときは、以下のように書く

```
$ for ((i=1;i<=10;i++)); do echo `expr $i ¥* 2` ; done
```

    - 上のワンライナーは、スクリプトファイルとして書くと、以下のようになる

```
#!/bin/bash
for ((i=1;i<=10;i++))
do
    echo `expr $i ¥* 2`
done
```

シェルスクリプトの安全性

- シェルスクリプトはできるだけ使わないほうがいいという意見もある
  - 自分の意図と違う動作をするスクリプトを書いても、エラーが出ず気づかないかも
  - Pythonなどhigher-levelな言語で同じことができる（エラーや警告も親切）

- 特に管理者権限で処理を実行するスクリプトを書くときは注意！

例）「`sudo -u nobody "$@"`」という行がスクリプトに入っているとする

- このスクリプトに引数「`-u root reboot`」を渡すと、リブートされてしまう
- なぜなら、`sudo`コマンドは最後に現れた「`-u`」の情報を拾うため

- https://sipb.mit.edu/doc/safe-shell/

課題2 (1/2)

- 下のWebページの問題をそのまま使用または一部改変しました

https://eng-entrance.com/linux-plactice-shellscript

1. 「var-hello_world.sh」というファイルを作成して、スクリプト内で変数STRに「STR="hello world"」と代入、それをechoコマンドで表示してみよう
2. 「if-gt.sh」を作成し、 if文を使用して、変数NUMの値が0より大きければ「True」と表示してみよう
   - 変数NUMをどのような値で初期化するかを変えてみつつ、動かす。

課題2 (2/2)

3. 「`for-list.sh`」というファイルを作成し、`for`文を使用して、文字列「`one`」「`two`」「`three`」を別々の行にこの順に表示してみよう
4. 「`while-line.sh`」というファイルを作成し、`while`文と`read`文を組み合わせて、「`/etc/hosts`」の中身を一行づつ表示してみよう

次回のための準備

- Overleafにアカウントがなければ、作っておきましょう

https://www.overleaf.com/