2024年5月10日

シェルとカーネル（教科書 3章 P77の図を参照）
計算機（OS）の中核（カーネル）と人間との間にあってコマンド履歴の保存などをしてくれるプログラムをシェルとよぶ。sh（シェル）, csh（シーシェル）,zsh（ズィーシェル）などいくつもの種類が有るがUbuntu上での標準はbash（バッシュ）

環境変数　（教科書 10章 10.4~10.6）
シェル中で使える変数をシェル変数と呼ぶ、プログラム中からも参照できるシェル変数を特に環境変数と呼ぶ。
変数名は SHELL など大文字だけで表記する様にする。値の参照時にははじめに$記号を付けて $SHELLの様に記す。

env コマンド （処理系によってはprintenv）　：　現在登録されている環境変数を全て列挙・表示する

echo コマンド　：　特定の環境変数を表示する時に使う
echo $変数名　（例は下記よく使う環境変数の右）

よく使う環境変数

$SHELL	使っているシェル	echo $SHELL -> /bin/bash
$USER	ユーザー名（アカウント名）	echo $USER -> m1600##
$HOME	ホームディレクトリ	echo $USER -> /home/linhome/m1600##
$PATH	コマンドファイルを探す場所のリスト	echo $PATH -> /bin:/sbin:/user/bin:..... （後述）

export コマンド　：　環境変数の設定
export　変数名=新しい値　例：　**export NEWVAL=value**　: NEWVALと言う環境変数に value と言う値が入る
ここでNEWVALの前に $ はつかないことに注意（$が付くのは環境変数の値を参照する時）

which コマンド　：　lsやwc, catなど指定したコマンドの実行ファイル（実体）がある場所を表示
Linuxのコマンドは C等で記述されたプログラムをコンパイルした実行ファイルであり、上記のPATH環境変数で記述されたいくつかの場所に分散して置かれている。which コマンドを使えばそれぞれのコマンド実行ファイルがどこにあるかを知ることができる。
例：　**which ls** -> /bin/ls　　**which wc** -> /usr/bin/wc

先に記述した PATH と言う環境変数には、コマンド実行ファイルを置くディレクトリのリスト（：で区切られている）になっている。人間が ls と打ち込むと、シェルは /bin ついで /sbin とPATH変数にリストされたディレクトリの中に ls というファイルを順次探し、最初に見つかった ls を実行する。
例えば今日作成する gc_count1 という実行形式ファイルもLinux コマンドと同様に扱えるが、このプログラムを作成するディレクトリ（~/C_PROT/03）は PATH のリストの中にはいっていない為、実行する際に、カレントディレクトリにある gc_count1であることを明示する為に、　./gc_count1　とタイプする必要がある。自分で作成した実行形式の置き場所を決めて、そのディレクトリをPATH環境変数のリストに追加しておけば、どのディレクトリからでも絶対パスを指定しなくてもコマンド名だけで実行できるようになる。（すでにあるコマンドと同じ名前にならないような注意は必要）
この講義では、実行形式の置き場所としてホームディレクトリの下に、binというディレクトリを作ることにする。
mkdir bin
export PATH=$PATH:~/bin
このexport文により、既に定義されたPATH変数に ~/bin を追加できる
ただし、環境変数はログアウトするとリセットされてしまうので、ホームディレクトリにあって、ログイン時に実行されるコマンドを記述した .bashrc と言うファイルの最後の行に、このexport 文を加えておけば、ログインするごとに export をしなくてすむ。

vi ~/.bashrc　　->　最終行に export PATH=$PATH:~/bin を追加

シェルの履歴 （3章 p82）

history コマンド　　：　　これまでに入力したコマンドの履歴が表示される

例：　**history**

```
  100    cat  .bashrc
  101    ls
  102    cd  C_PROG
  103    cd  04
  104    pwd
  105    history
```

! 記号：　履歴にあるコマンドを簡単に再実行

上記の例の状態で
!p としてenterキーを押せば、最後に実行した p で始まるコマンド pwd が実行される
!ca としてenterキーを押せば、最後に実行したcaで始まるコマンド　cat .bashrc が実行される
!102としてenterキーを押せば、102番目のコマンド cd C_PROG が実行される
!! としてenterキーを押せば、直前に実行した history が実行される

履歴については、上向きカーソルキー　↑　で順次さかのぼりながら実行したコマンドが表示され、Enterキーで実行される。　行き過ぎた場合下向きカーソルキー　↓　で戻れるほか、任意のコマンドが表示されている状態で、Backspaceキーまたは、左右のカーソルキー　←　→　で、表示されたコマンドを修正してから実行することができる。

標準入出力とリダイレクト・パイプ　（7章・p221, p228の図を参照）
それぞれのコマンドは通常、キーボードから入力を受け取り、ディスプレイ上に出力を表示する。この入力を標準入力(STDIN)とよぶ。出力には2種類あり　標準出力（STDOUT）とエラーメッセージなどを出力する標準エラー出力（STDERR）となっている。

リダイレクト　>　<　>>
これらの、標準入出力をファイルにつなぐのがリダイレクトである。date の出力を画面ではなく date.txt というファイルに書き出すには　**date　>　date.txt**　とする。もう一度 **date > date.txt** とすると、1度目の結果は上書きされ、date.txtの中身は1行のままである。すでにあるファイルの末尾に、書き足していくためには >> を使って、**date >> date.txt** とする。以上は標準出力をファイルにつなぐ例で、逆に標準入力をファイルからつなぐ例として、たとえば、　**wc　<　date.txt**　とすれば、date.txt の中身を ワードカウント wcにつないで 行数・語数・文字数をカウントできる。 wc についてはより単純に、 **wc　date.txt**　とすることもできるが、標準入力のみを受け付ける仕様のコマンドではリダイレクトが有用な使い方になる。

パイプ　|
リダイレクトとは異なり、パイプを使うと、あるコマンドの標準出力を、直接他のコマンドの標準入力とすることができる
たとえば、リダイレクトで、　　**ls　>　list.txt**　として、　**wc　<　list.txt**　とすれば、lsの出力行数などをカウントできるが、list.txt というファイルを介さなくても、　　**ls　|　wc**　と直接 ls コマンドの出力をwc コマンドの入力とできる。

フィルタコマンド　（8章　p234）
標準入力を受けて、標準出力を出すコマンドをフィルタコマンドとよび、これらとパイプの機能をうまく組み合わせて活用することで、さまざまな処理が可能になる。
フィルタコマンドの例：
cat　：入力（ファイルの内容）をそのまま出力
head　：入力の先頭部分のみを出力　（ オプションで行数指定　-10 で先頭10行）
tail　：入力の末尾部分のみを出力　（オプションで行数指定　head と同様）
grep　：検索パターンに一致する行のみ出力　（後述）
sort　：順番に並べ替え
uniq　：重複行を除く
tac　：逆順に出力
wc　：行数・語数・文字数を出力

du -b /bin/*　：/binにあるファイルのサイズ（バイト数）を列挙
du -b /bin/* | sort -n　：数値順・小さい方から列挙
du -b /bin/* | sort -rn　：数値順・大きい方から列挙
du -b /bin/* | sort -rn | head -5　：数値順・大きい方から５個だけ表示

↑　最後のコマンドの出力例
1984584　/bin/busibox
1037528　/bin/bash
678496　/bin/networkctl
663952　/bin/systemctl
498912　/bin/jounalctl

grep コマンド　：　特定の文字列を含む行のみを出力
grep ABC foo.txt で foo.txtというファイルからABCを含む行のみを出力
-v オプションを付ければ、逆に 文字列を含まない行のみを出力できる
フィルタとしても機能するので、 **grep ABC foo.txt | grep DEF** とすれば、foo.txtから、ABCとDEF両方を含む行のみを出力できる。（２番めのgrepがフィルタ機能）
探したい文字列がスペースなどを含む場合　シングルクォートで検索文字列を囲む
grep ' gene ' Ecoli.gb
このようなエントリの行数をカウントしたければ
grep ' gene ' Ecoli.gb | wc

vi の機能（簡易版）

文字列の修正にバックスペースキーを用いない
カーソルの移動にマウス（スクロールホィール）を用いない

以下に述べるのは
基本的にノーマルモードでのコマンド　（<Esc>を押してからならok）

i	カーソルの前からインサートモードに
a	カーソルの後からインサートモードに
o	カーソルの下の行からインサートモードに
O	カーソルの上の業からインサートモードに
r	次に打つ１文字でカーソル位置を置き換える
R	カーソル位置から上書きのインサートモード
dw	単語の削除
cw	単語の変更　（インサートモードに入る）

インサートモードから抜けるときはいつでも <Esc>

/	検索	ノーマルモードで　/　の後に探したい文字列を入れる	**/CDS**
?	上方検索	カーソル位置より前の検索文字を探す	**?CDS**
n	繰返し検索	/や?による検索を繰り返す	

.　　前の操作の繰り返し　例えば特定の文字列を検索してcwする操作は一度行えばあとは n.n.n. とすることで次々と置換を繰り返せる

:set number	行番号を表示する　　(.vimrc に書いておくこともできる)
:set nonumber	行番号を非表示にする
dd	一行削除（その行はバッファーに入る）
yy	一行コピー（バッファーに入る）（ヤンク）
p	バッファーの中身をカーソルの下に貼り付け
P	バッファーの中身をカーソルの上に貼り付け

ビジュアルモード
V　　カーソルの移動により範囲を指定（行単位）
v　　カーソルの移動により範囲を指定（文字単位）
カーソル移動して選択範囲を決定したら、続けて
d　　選択範囲を削除（バッファに入る）
y　　選択範囲をコピー（バッファに入る）
続けてカーソルを移動して場所を選んでバッファの中身をペースト

置換　文全体で特定の文字列を別の文字列に書き換えたい場合　（たとえば CDS を NCDS に）

:1,$s/CDS/NCDS/g

１行目から、最後の行（$)まで、CDS を NCDS に置き換える 最後の g が無いと特定の１行にCDSが複数ある場合一つ目だけを置換

編集結果を上書きしないで（ファイルをもとのままにして）終わる場合には　<ESC> してから

:q!

書き換えた結果を保存する場合には

:wq

チュートリアルがまだ終わってない人は
しておいてくださいね