管理者権限がない場合の快適な Linux(Mac,WSL) 開発環境のすゝめ

M. Kanega

2019 年 8 月 19 日

概要

最近友人の PC やらに私の PC の設定を移植する機会が多くあるため、これを機に情報を一元化しようと考えた次第である。ゆえに本稿では 2019 年 6 月現在で個人的に最も快適と思われる開発環境・環境整備について箇条書き的にまとめる。内部的な仕組みをよく理解していなくとも、本稿で提示されたコマンドをターミナルに打ち込んでいけば開発環境が整うようにという意識で書いた。想定する読者としては、ターミナルを少し使ったことはあるが詳しいことはわからない人から、バリバリ設定いじるけど、コマンド忘れがちだから少しまとまっている記事が欲しいという玄人まで、幅広いところを標的としている。「ターミナル？コンソール？コマンド？なにそれ？おいしいの？」という人は少しターミナルを触ってみてから読んでほしい。必須ではないが、知っていると便利な場面があるアドバンストな項目については † をつけて区別している。また、主に管理者権限がないような場合について述べる（管理者権限があるような場合は付録に簡単にまとめた。）。

目次

1　準備

これから様々なソフトをインストールする前に、インストールするためのディレクトリを整備しておく。

bash

```
$ mkdir ~/usr
$ mkdir ~/usr/bin
$ mkdir ~/opt
```

また、必要に応じて path を通しておく。(赤文字が追記・変更した箇所である。)[*1]

~/.bash_profile

```
......

# User specific environment and startup programs

PATH=$HOME/bin:$HOME/usr/bin:$PATH

......
```

bash

```
$ source ~/.bash_profile
```

2　vim

vim はエディターであり、簡単なソースの編集やファイルの編集に重宝される。vim と対をなす emacs や nano というエディターも存在するが、ここでは割愛する。

初期の段階から使えるようになっている事が多いが、蓋を開けてみると最小パッケージの状態でインストールされており、後述するシンタックスハイライトなど多くの機能が使えないようになっていることが多い。

ここでは、”huge 版”と呼ばれる、シンタックスなどが利用できるようになっている vim を使える状態にすることを目標とする。

2.1　インストール

以下のコマンドを一行ずつ入力していけばインストールすることができる ($ は入力しなくて良い。)。

[*1] ファイルを編集するときは、以下のようにコンソールにコマンドを打ち込む必要がある。
`$ vi {file_name}`
以上のようにすると vim 上でテキストエディタが立ち上がる。
なお、その状態で `a` と入力するとテキストを入力できる状態になり、`esc` キーを入力するとコマンド受付モードに戻る。コマンドモードで大文字の `ZZ` と打ち込むと上書き保存してエディタを閉じる。更に詳細な操作法が知りたい場合は [12] を参照。

bash

```
$ cd ~/opt
$ git clone https://github.com/vim/vim.git
$ cd vim
$ ./configure --prefix=$HOME/usr --with-features=huge --enable-multibyte --enable-cscope --enable-fail-if-missing
$ make
$ make install
$ cd ~/usr/bin
$ ln -s vim vi
```

2.2 シンタックスハイライト

シンタックスハイライトを有効にするための手順を紹介する。

シンタックスハイライトとは、コマンドやソースコードを書き込むと自動的に色付けして見やすくできる機能の事である。

vim の各種設定を記述しておくファイルはデフォルトで ~/.vimrc あるいは ~/.gvimrc である。これらのファイルを編集することにより vim の設定を改変する。なお、gvimrc についてはシンタックスハイライトに直接関係するところではなく、記述は必須ではない（エディターの見た目の問題である。）。

~/.vimrc

```
syntax on

let OSTYPE = system('uname')
if OSTYPE == "Darwin\n"
:set term=xterm-256color
:syntax on
endif
```

~/.gvimrc

```
set lines=55
set columns=180
colorscheme koehler
set background=dark
```

3 zsh + prezto [1–3]

zsh とは shell の一つである。

bash の系統であるが bash よりもカスタマイズに対して柔軟に対応できる shell であり、使いやすい（著者談）。

最近では有志による zsh カスタマイズパッケージ prezto がオープンソースで配布されているため、初学者にもとっつきやすくなっている。

ここでは zsh と prezto を組み合わせて bash よりも視認性の高いコンソール画面を目指すこととする。

3.1 インストール

以下のコマンドを一行ずつ入力していけば zsh をインストールすることができる ($ は入力しなくて良い。)。

bash

```
$ cd ~/opt
$ wget "http://nchc.dl.sourceforge.net/sourceforge/zsh/5.7.1/zsh-5.7.1.tar.xz"
$ tar -Jxvf zsh-5.7.1.tar.xz
$ cd zsh-5.7.1
$ ./configure --prefix=$HOME/usr --enable-multibyte --enable-locale --disable-gdbm
$ make
$ make install
```

次に bash に対して、ログイン時に zsh を起動するよう命令を書いておく。
chsh コマンドを用いてログインシェルを変更してもよいが、今回はこの方法を取る。

~/.bash_profile

```
・・・・・・

# User specific environment and startup programs

PATH=$HOME/bin:$HOME/usr/bin:$PATH

export PATH

exec $HOME/usr/bin/zsh

・・・・・・
```

さらに以下のコマンドにより、prezto をインストールする。

bash

```
$ zsh
$ git clone --recursive https://github.com/sorin-ionescu/prezto.git "${ZDOTDIR:-$HOME}/.zprezto"

// 既存の設定ファイルを退避（必要な場合）
$ mkdir zsh_orig && mv zshmv .zlogin .zlogout .zprofile .zshenv .zshrc zsh_orig

$ setopt EXTENDED_GLOB
for rcfile in "${ZDOTDIR:-$HOME}"/.zprezto/runcoms/^README.md(.N); do
ln -s "$rcfile" "${ZDOTDIR:-$HOME}/.${rcfile:t}"
done
```

これで zsh をインストールすることができた。
一度 exit してからもう一度 zsh すると表示が変わったことに気づくだろう。

3.2 各種オプション

zsh の強みとして、カスタマイズの容易さがある。どういうカスタマイズが使いやすいかは人によるので深くは触れないが、以下に筆者オススメのカスタマイズを載せておく。(赤文字が追記・変更した箇所である。)

~/.zpreztorc

```
.....

zstyle ':prezto:load' pmodule \
'environment' \
'terminal' \
'editor' \
'history' \
'directory' \
'spectrum' \
'utility' \
'completion' \
'syntax-highlighting' \
'autosuggestions' \
'git' \
'prompt' \
'command-not-found'

.....

# Prompt

# Set the prompt theme to load.
# Setting it to 'random' loads a random theme.
# Auto set to 'off' on dumb terminals.
zstyle ':prezto:module:prompt' theme 'paradox'

.....

# Set syntax highlighting styles.
# zstyle ':prezto:module:syntax-highlighting' styles \
#   'builtin' 'bg=blue' \
#   'command' 'bg=blue' \
#   'function' 'bg=blue'
#
# Set syntax pattern styles.
# zstyle ':prezto:module:syntax-highlighting' pattern \
#   'rm*-rf*' 'fg=white,bold,bg=red'

zstyle ':prezto:module:syntax-highlighting' color 'yes'

.....
```

~/.zshrc

```
......

# Customize to your needs...

autoload -U compinit
compinit -u
```

4 ssh[4–7]

ssh は離れた場所にある計算機などにアクセスする際に有用な機能であり、このコマンドを習得することは必須である場合が多い。

なお、ssh は最初から標準装備で入っていることが多いため、ここではインストール方法については述べない。

4.1 GUI 環境で ssh する

リモート側から GUI の画面を引っ張ってきたいときは次のように-YC オプションを付ける。

zsh

```
$ ssh -YC {remote_user}@{remote_host}
```

4.2 ssh 鍵による認証方法

ssh でリモートにログインするとき、いちいちパスワードを入力するのはめんどくさい。しかし、通信はセキュアにしたい。そんなときに便利なのが公開鍵と秘密鍵の利用である。

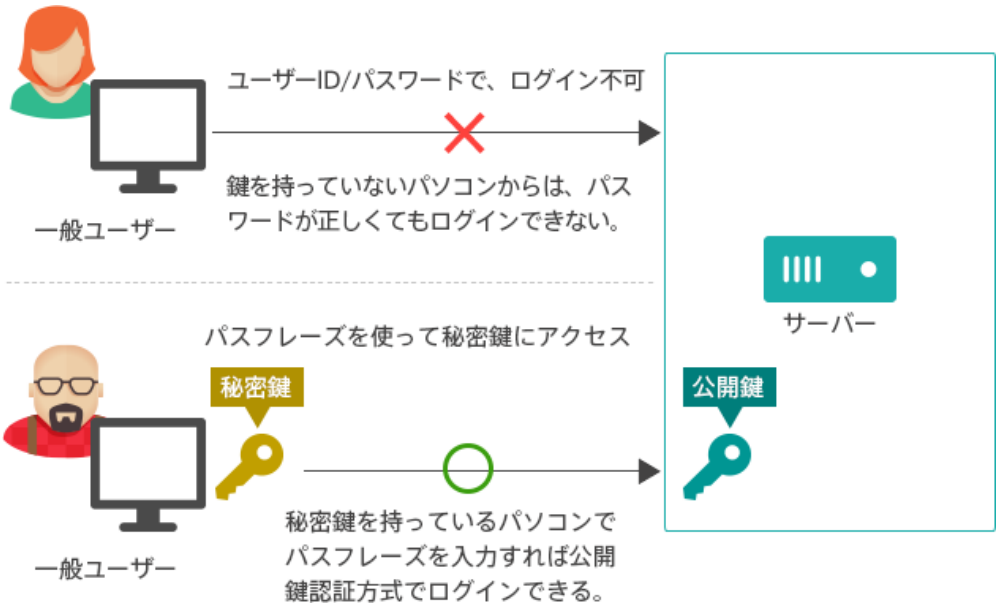

図 1: ssh 鍵認証の仕組み[*2]

[*2] https://knowledge.sakura.ad.jp/3543/ より引用

サーバー側に公開鍵を置き、ローカルに秘密鍵を持っておく。ssh コマンドでリモートからローカルの秘密鍵を参照したとき、対応した秘密鍵ならば接続ができるという仕組みである。

ssh 鍵の実装において一番注意しなければいけないのは、**秘密鍵は絶対に流出させてはいけない**ということである。

早速実装していくこととしよう。

zsh

```
$ cd ~/.ssh
\\ いろいろ聞かれるけどここでは Enter キーを 3 回押してもらえば OK
\\ .ssh フォルダに id_rsa（秘密鍵）,id_rsa.pub（公開鍵）が生成される。
$ ssh-keygen -t rsa -b 4096
\\ サーバーにログイン
$ ssh hoge@hoge.com
\\ フォルダとファイルを作る（既に存在している場合は実行しなくても可）
$ mkdir .ssh
$ touch .ssh/authorized_keys
\\ 権限の設定。ここを間違えると、公開鍵認証でログイン出来ないので注意
$ chmod 700 .ssh
$ chmod 600 .ssh/authorized_keys
$ exit
\\ 以下、サーバーに鍵を登録する。
$ ssh-copy-id -i id_rsa.pub hoge@hoge.com
```

これで鍵認証によるログインが可能になっているはずである。[*3][*4]

4.3 ～/.ssh/config の設定

ssh の設定は ～/.ssh/config に事細かく記述することができる。特に、ssh ログイン時にアドレスを省略できる点が便利である。ここでは多彩な設定の中で最も簡単なところについて解説する。

目標としては、

zsh

```
$ ssh {remote_user}@{remote_host}
```

ではなく、

zsh

```
$ ssh {Host}
```

でログインできるように設定する。

覚えておくべきオプションが複数あるため、箇条書きで説明する。

[*3] ここでは rsa 鍵を用いたが、最近は rsa 鍵では少々古いようである。ecdsa などの鍵を使うのが推奨されている。詳しくは [7] を参照。

[*4] 公開鍵認証とパスワードによる認証方式を併用することはあまりよろしくない。複数の手段でログインできてしまうのはセキュリティホールを広げることになるからである。ゆえに、パスワードによる認証方式を無効化しておきたい。しかし、無効化するためには管理者権限が必要なので、管理者権限がない場合は管理者権限がある人にお願いしてみよう。パスワード無効化の手法についても [7] に記載がある。

オプション

Host　任意の名前を指定できる。実際に ssh するときにはこの名前で指定することになる。*でワイルドカードを指定することもできる。
HostName　リモートのドメイン名を書く。
User　リモートのユーザー名を書く。
Port　デフォルトでは 22 番が指定される。変更したい場合は開いているポート番号を指定する。
IdentityFile　対応する ssh 秘密鍵の場所を指定する。
ProxyCommand　多段 ssh（後述）に必要。
LocalForward　VNC 画面共有する場合に必要。
RemoteForward　リモートのファイルをローカル側から編集したいときに使える。
ServerAliveInterval　一定時間操作しないときに接続が死なないようにする設定。指定時間ごとにリモートにシグナルを送る。

設定の一例を以下に示しておく。

~/.ssh/config

```
Host *
    ServerAliveInterval     60
    ForwardX11Timeout       24h

Host github github.com
    HostName                github.com
    IdentityFile            ~/.ssh/id_git_rsa
    User                    git

#hoge (hoge@hogehoge.com)
Host hoge
    HostName                hogehoge.com
    Port                    22
    User                    hoge
    LocalForward            8888 localhost:5901
    RemoteForward           52698 localhost:52698
```

4.4　多段 ssh をする場合 †

場合によっては、大学側のサーバーが隔離されたスペースにあり、複数のサーバーを経由しないと到達不可能な場合がある。そのような場合に便利なのが多段 ssh の設定であり、ProxyCommand の活躍するところである。

目的のサーバーにアクセスしたいというコマンドを受け取ったら、踏み台サーバーに接続するコマンドを実行するというスクリプトを組んでおいて、それを連鎖させるというイメージを持っておくと素直に理解できるのではないかと思う。

以下にサンプルとして
localhost
↓
hoge@hoge.com(踏み台サーバー 1)
↓
hogehoge@hogehoge.com(踏み台サーバー 2)
↓
fuga@fuga.com(接続したいサーバー)
というように多段 ssh するようにできた config を記載しておく。

~/.ssh/config

```
# fuga@fuga.com
Host fuga
    HostName        fuga.com
    Port            22
    User            fuga
    IdentityFile    ~/.ssh/id_rsa
    ProxyCommand    ssh -YC -W %h:%p hogehoge
    LocalForward    8888 fuga.com:5901
    RemoteForward   52698 localhost:52698

#hogehoge@hogehoge.com
Host hogehoge
    HostName        hogehoge.com
    Port            22
    User            hogehoge
    IdentityFile    ~/.ssh/id_rsa
    ProxyCommand    ssh -YC -W %h:%p hoge
    LocalForward    8888 hogehoge.com:8888

#hoge@hoge.com
Host hoge
    HostName        hoge.com
    Port            22
    User            hoge
    IdentityFile    ~/.ssh/id_rsa
    LocalForward    8888 localhost:8888
```

このように記述し、fuga@fuga.com に ssh コマンドを、

zsh

```
$ ssh -YC fuga
```

と打てば X11Forwarding がオンになり、localhost の 52698 ポートに RemoteForward が通り、8888 ポートに VNC 画面共有用の接続が生成される。[*5]

5 git[8, 9]

5.1 インストール

以下のコマンドを一行ずつ入力していけばインストールすることができる ($ は入力しなくて良い。)。

[*5] 公開鍵は事前に各サーバーにそれぞれ § 4.2の内容を実行して登録している。

zsh

```
$ cd ~/opt
$ wget https://github.com/git/git/archive/v2.17.1.tar.gz
$ tar -zxvf v2.17.1.tar.gz
$ cd git-2.17.1/
$ ./configure --prefix=$HOME/usr
$ make
$ make install
```

5.2 使用方法

ここで説明するには少々文章量が多くなりそうである。幸いにもわかりやすく解説されているウェブサイトが多数存在しているため、詳しくは [8] や [9] などを参照していただきたい。(後のバージョンで気が向いたら記載するかもしれない。)

とはいえ、多数の機能を持つ git を全て網羅する必要はなく、あくまで研究活動に必要な範囲で習得すれば良い。以下では最低限使えるべきコマンドと、git に push (アップロード) または pull (ダウンロード) する一連の流れを示しておく。(各コマンドのイメージについては述べないので、ウェブサイトを参照してほしい。)

カレントディレクトリの追加・更新ファイルを一括でアップロードする方法

zsh

```
$ git add .
$ git commit .
\\ここでエディターが立ち上がるので、なにかコメントを記入して保存する。
\\(記入しないと abort される。)
$ git push
```

クラウド上の更新ファイルを一括でダウンロードする方法

zsh

```
$ git pull
```

すでに存在するプライベートリポジトリを clone して push できる状態にする方法

zsh

```
$ git clone https://{user_name}@github.com/{user_name}/{repository_name}.git
\\パスワード聞かれるので入力してあげる。

\\この状態でいきなり push することはできない。お前はどこのどいつだ！と怒られる。
\\つまり、どこのどいつか教えてあげれば push できる。
\\自分のメールアドレスと名前を教えてあげよう。(本名じゃなくても OK)
$ git config --global user.email "hoge@hoge.com"
$ git config --global user.name "hoge"
```

新規にリポジトリを立ち上げる場合（git で管理したいフォルダにいる状態から始める。）

zsh

```
$ git init
$ git commit .
$ git remote add origin https://github.com/{user_name}/{repository_name}.git
$ git push
```

ちなみに、いずれの場合も github.com にアカウントとリポジトリが存在しなくてはならない（当然だが）。
アカウントを作りたい場合は、https://github.com/ にアクセスし、Sign up する。

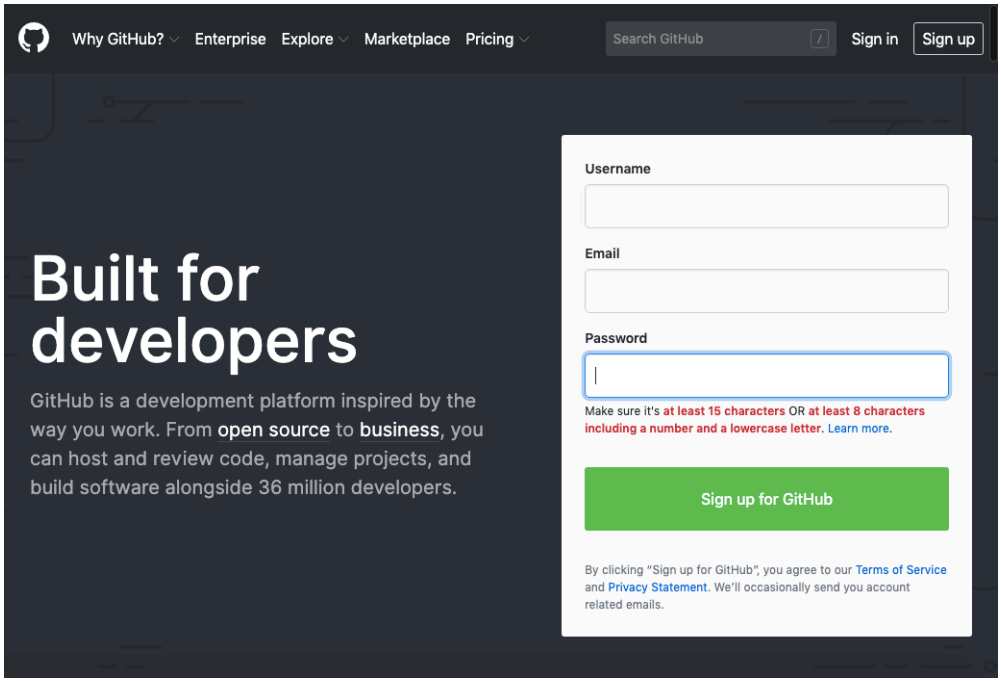

図 2: github のトップページ

そして、飛ばされたユーザーページから新しいリポジトリを作れば OK である。

5.3 ssh の利用

プライベートリポジトリを利用している場合、push や pull のたびにパスワードを聞かれて面倒である。そこで、ssh の鍵認証を利用する。§ 4.2で述べた方法により、秘密鍵と公開鍵を作り、そのうち公開鍵の中身をまるっとコピーし、図 3にペーストする。タイトルは公開鍵の名前とする。さらに、git で管理しているフォルダに移動し、.ssh/config の編集及び、次のコマンドを実行すると通信が http から ssh になりパスワードレスで push&pull できるようになる。

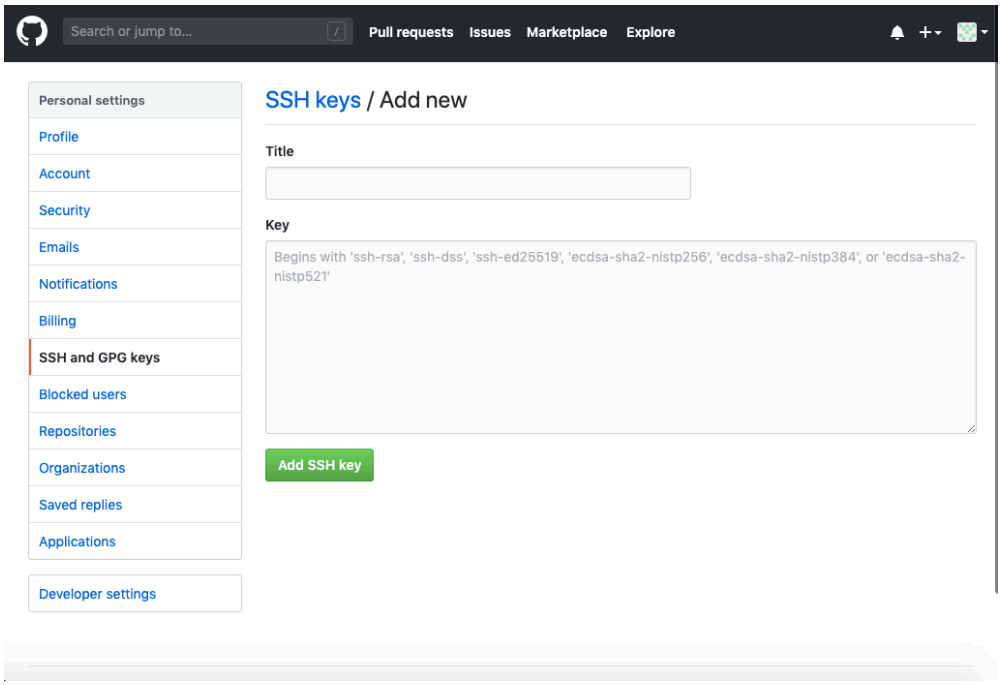

図 3: ssh 公開鍵の登録

~/.ssh/config

```
Host github github.com
 HostName github.com
 IdentityFile ~/.ssh/{git 用に生成した秘密鍵 }
 User git

. . . . . .
```

zsh

```
$ git remote set-url origin github:{user_name}/{repository_name}.git
```

6 screen[10]

長時間かかるプログラムを実行しているとき、ジョブをバックグラウンドで流しているだけでは進行状況がわからずもやもやすることがある。そんなときはプログラム実行と同時にプログレスバーを表示し、何%ほど計算が終わったのか確認できるようにするのがスマートだろう。しかし、例えば ssh でリモートに計算を飛ばしているときにプログレスバーを表示し続けるにはフォアグラウンドで処理を続けなければいけないので、接続を切断することができない。これはとても不便である。そこで登場するのが screen である。

screen は仮想的な画面を新たに生成するコマンドである。さらに、フォアグラウンドで仮想的に開いている画面のデータを保持したままバックグラウンドに回すことができる。すなわち、バックグラウンドでジョブを回しているにもかかわらず、プログレスバーや定数の情報は失われない。また、バックグラウンドに回したジョブはいつでもフォアグラウンドに引っ張ってくることができる。

このように便利なコマンドなのである。

6.1 インストール

以下のコマンドを一行ずつ入力していけばインストールすることができる ($ は入力しなくて良い。)。

zsh

```
$ cd ~/opt
$ wget https://ftp.gnu.org/gnu/screen/screen-4.6.2.tar.gz
$ tar -zxvf screen-4.6.2.tar.gz
$ cd screen-4.6.2
$ ./configure --prefix=$HOME/usr --enable-colors256 --enable-locale
$ make
$ make install
```

6.2 使用方法

仮想コンソールを出すためには、頭に `screen` とつけてコマンドを打ってやれば良い。例えば、実行ファイル a.out を実行したいときは以下のようにする。

zsh

```
$ screen ./a.out
```

screen の仮想コンソールが立ち上がった状態からコンソールを保持したままバックグラウンドプロセスに回すためには `Ctrl+A` のあとに `D` を押す必要がある。
今、バックグラウンドにどんなプロセスがあるか確認するためには、以下のようにする。

zsh

```
$ screen -ls
```

こうすることで、現在実行中のプロセスの番号一覧が確認できる。
さらに、バックグラウンドの特定のプロセスをアクティブに呼び戻すためには、以下のようにする。

zsh

```
$ screen -r {process_number}
```

他にも様々な機能があるが、詳しくは [10] などを参照すると良い。

7 gnuplot[11]

gnuplot はオープンソースのグラフ描画ソフトである。Fortran や C 言語のソースコード上から呼び出すことができるなど、動作が非常に軽いことが特徴で、一部の分野で非常に重宝されているツールである。

7.1 インストール

以下のコマンドを一行ずつ入力していけばインストールすることができる ($ は入力しなくて良い。)。

zsh

```
$ cd ~/opt
$ wget https://sourceforge.net/projects/gnuplot/files/gnuplot/5.2.7/gnuplot-5.2.7.tar.gz
$ tar -zxvf gnuplot-5.2.7.tar.gz
$ cd gnuplot-5.2.7
$ ./configure --prefix=$HOME/usr
$ make
$ make check
$ make install
```

ここまで実行したあとに、

zsh

```
$ which gnuplot
```

と打って、

zsh

```
/home/{user_name}/usr/bin
```

あるいは、

zsh

```
~/usr/bin
```

と返ってくればインストール成功である。

7.2 ssh との連携

リモートの gnuplot はローカルから ssh するときに X11Forwarding を許可しているとグラフをインタラクティブに飛ばしてくることができる。

具体的には、§ 4.1で述べた通り、ローカルから ssh するときに

zsh

```
$ ssh -YC {remote_user}@{remote_host}
```

というように`-YC` オプションを付けてやると、X11Forwarding がオンの状態になる。

そこで gnuplot に `test` と打ち込んでやると、ローカルにグラフが転送されてきたことがわかるだろう。

7.3 Fortran との連携 †

Fortran から直接 gnuplot を動かすためには、一度 gnuplot の設定ファイル (.plt,.gp など) を作成して、そこにコマンドを書き込み、system call で gnuplot を呼び出すという手順を取る。

以下にサンプルプログラムを添付するので参考にしてほしい。

Code 1: Fortran

```fortran
PROGRAM sample

  ! ・・・・・・
  ! ここまでで fileName=hoge.dat にデータを格納したと仮定。

  !----GNUPLOT-------------------------------
  fileName = "'"//TRIM(ADJUSTL(fileName))//"'"
  OPEN(13,file = "gnupdummy.plt")
  WRITE(13,*) "se te po eps enhanced color"
  WRITE(13,*) "se ou ", TRIM(ADJUSTL(figName))
  WRITE(13,*) "se xl 'x'"
  WRITE(13,*) "se yl 'y'"
  WRITE(13,*) "plot", TRIM(ADJUSTL(fileName)), " u 1:2 w p pt 7 lt 1"
  CLOSE(13)
  CALL system("gnuplot gnupdummy.plt")


END PROGRAM sample
```

7.4 C++ との連携 †

C++ から直接 gnuplot を動かすことは Fortran よりも単純である。

ファイルポインタから呼び出すことができるのはテキストファイルだけではない。コマンドを指定することで任意のプログラムを呼び出すことができる。

一度ファイルポインタでプログラムを呼び出してしまえば、あとはテキストファイルに書き込むのと同じ感覚でコマンドを書き込んでいくことができる。

以下にサンプルプログラムを添付するので参考にしてほしい。

このサンプルプログラムは Overdamped Langevin equation を Euler 法で解き、その軌跡のアニメーションと、軌跡の時間発展のグラフを同時出力するプログラムである。

Code 2: C++

```cpp
/*  ヘッダファイルの宣言  */
#include <cassert>
#include <time.h>

#include <cmath>
#include <cstdio>
#include <cstdlib>
#include <fstream>
#include <iostream>
#include <random>

#ifdef _OPENMP /*  parallelize  */
#include <omp.h>
#endif

/*  std:: の省略・namespaceの宣言  */
using namespace std;

/*  関数の宣言  */
double Apot(double, double);
```

```
21
22  double Hamiltonian(double, double);
23
24  /*  変数の宣言  */
25  long i, n, N;
26
27  int error;
28
29  double dt, runtime, TRANSIENT_TIME;
30  long TRANSIENT_STEP;
31
32  double k;
33
34  double dx, x0, x1;
35
36  double Gamma, kB, Tbath, coef;
37
38  double fext, fnoise;
39
40  double E;
41
42  double xhistMAX;
43
44  string logFile, figFile, gifFile, dataFile, origin;
45  int prog;
46
47  int main() {
48
49    /*  ファイルポインタ  */
50    FILE *gp = popen("gnuplot", "w");
51    assert(gp);
52
53    /*  設定の入力  */
54    cout << "What␣is␣name␣of␣output␣file?(Except␣for␣file␣extension.)␣:␣";
55    cin >> origin;
56
57    dataFile = origin + ".dat";
58    figFile = origin + ".eps";
59    gifFile = origin + ".gif";
60
61    logFile = "NumericalExperiment.log";
62
63    cout << "Tbath␣=␣";
64    cin >> Tbath;
65
66    cout << "Gamma␣=␣";
67    cin >> Gamma;
68
69    cout << "k␣=␣";
70    cin >> k;
71
72    cout << "dt␣=␣";
73    cin >> dt;
74
75    cout << "TRANSIENT_TIME␣=␣";
76    cin >> TRANSIENT_TIME;
77
78    cout << "\n";
79
```

```
80    TRANSIENT_STEP = (long)TRANSIENT_TIME / dt;
81
82    dx = 1.0e-8;
83
84    /*  アニメーション用  空間サイズの設定  */
85    xhistMAX = 10;
86
87    kB = 1.0;
88    coef = sqrt(2.0 * Gamma * kB * Tbath / dt);
89
90    prog = 1;
91
92    /*  Mersenne twisterのセッティング（周期2^19937-1の乱数）  */
93    random_device rd;
94
95    mt19937_64 mt(rd());
96
97    normal_distribution<double> gasdev(0, 1); /*  標準正規分布  */
98
99    /*  現在時刻  */
100   char currTime[100];
101
102   time_t current;
103   struct tm *local;
104
105   time(&current);              /*  現在の時刻を取得  */
106   local = localtime(&current); /*  地方時の構造体に変換  */
107
108   sprintf(currTime,
109           "[␣%04d/%02d/%02d␣ー␣%02d:%02d:%02d␣]:␣", local->tm_year + 1900,
110           local->tm_mon + 1, local->tm_mday, local->tm_hour, local->tm_min,
111           local->tm_sec);
112
113   /*  ログファイル生成  */
114   ofstream logfile(logFile, ios::app);
115
116   logfile << "This␣is␣a␣c++␣datalog␣of␣executable␣file␣:␣'" << __FILE__ << "'."
117           << "\n"
118           << "Source␣file␣is␣compiled␣at␣" << __TIME__ << "␣-␣" << __DATE__
119           << "␣.\n"
120           << "The␣result␣is␣outputted␣to␣'" << dataFile << "'.\n"
121           << "The␣figure␣of␣result␣is␣outputted␣to␣'" << figFile << "'.\n"
122           << "The␣animation␣of␣result␣is␣outputted␣to␣'" << gifFile << "'.\n\n"
123           << currTime << "Program␣is␣started."
124           << "\n"
125           << "CONSTANT␣:␣"
126           << "\n"
127           << "Tbath␣=␣" << Tbath << "\n"
128           << "Gamma␣=␣" << Gamma << "\n"
129           << "dt␣=␣" << dt << "\n"
130           << "TRANSIENT_TIME␣=␣" << TRANSIENT_TIME << "\n\n";
131
132   /*  プログレスバー  */
133   fprintf(stderr, "0%%␣␣␣␣␣␣␣50%%␣␣␣␣␣␣␣100%%\n");
134   fprintf(stderr, "+---------+---------+\n");
135   fprintf(stderr, "#");
136
137 START:
138
```

```
139   /*  Animation configure (gnuplot)  */
140   fprintf(gp, "set␣terminal␣gif␣animate␣optimize␣delay␣5␣size␣800,400\n");
141   fprintf(gp, "set␣output␣'%s'\n", gifFile.c_str());
142   fprintf(gp, "set␣nokey\n");
143   fprintf(gp, "set␣size␣square\n");
144   fprintf(gp, "set␣xr␣[-%f:%f]\n", xhistMAX, xhistMAX);
145   fprintf(gp, "set␣yr␣[-1:1]\n");
146
147   clock_t start = clock(); /*  開始時刻取得  */
148
149   ofstream outputfile(dataFile); /*  ファイルポインタ  */
150
151   x0 = 0.0; /*  変数初期化  */
152
153   /*  Animation  */
154   fprintf(gp, "set␣title␣\"Brownian motion, t = %.3f\"\n", runtime);
155   fprintf(gp, "plot␣'-'␣pt␣7␣lt␣3␣ps␣5\n");
156
157   fprintf(gp, "%f,␣0\n", x0);
158   fprintf(gp, "e\n");
159
160   /*  時間発展  */
161   for (i = 1; i <= TRANSIENT_STEP; i++) {
162
163     /*  プログレス(5%刻み)  */
164     if (1.0 / TRANSIENT_STEP * i * 100 >= 5 * prog) {
165       fprintf(stderr, "#");
166       prog += 1;
167     }
168
169     /*  Euler法  */
170     runtime = i * dt;
171
172     fext = -(Apot(x0 + dx, k) - Apot(x0 - dx, k)) / 2.0 / dx;
173
174     fnoise = coef * gasdev(mt);
175
176     x1 = x0 + dt * (fext + fnoise) / Gamma;
177
178     /*  発散のチェック  */
179     if (x1 < -1000 || 1000 < x1) {
180
181       error = error + 1;
182
183       if (error == 1) {
184
185         cout << "don't␣panic!␣Initializing.....␣Calculation␣RESTART." << '\n';
186       }
187       if (error == 100) {
188
189         cout << "quit..." << '\n';
190
191         goto END;
192       }
193
194       goto START;
195     }
196
197     E = Hamiltonian(x1, k); /*  現在のエネルギー  */
```

```
198
199     x0 = x1; /*  変数の更新  */
200
201     outputfile << runtime << "␣" << x1 << '\n'; /*  ファイルへの記録  */
202
203     /*  Animation  */
204     fprintf(gp, "set␣title␣\"Brownian motion, t = %.3f\"\n", runtime);
205     fprintf(gp, "plot␣'-'␣pt␣7␣lt␣3␣ps␣5\n");
206
207     fprintf(gp, "%f,␣0\n", x1);
208     fprintf(gp, "e\n");
209   }
210
211 END:
212
213   clock_t end = clock(); /*  終了時刻取得  */
214
215   currTime[0] = '\0'; /*  時刻初期化  */
216
217   time(&current);               /*  現在の時刻を取得  */
218   local = localtime(&current); /*  地方時の構造体に変換  */
219
220   /*  現在時刻の出力  */
221   sprintf(currTime,
222           "[␣%04d/%02d/%02d␣ー␣%02d:%02d:%02d␣]:␣", local->tm_year + 1900,
223           local->tm_mon + 1, local->tm_mday, local->tm_hour, local->tm_min,
224           local->tm_sec);
225
226   /*  ログ記録（実行時間）  */
227   logfile << currTime << "Program␣is␣ended."
228           << "␣␣|␣␣"
229           << "Duration␣=␣" << (double)(end - start) / CLOCKS_PER_SEC
230           << "sec.\n\n\n";
231
232   outputfile.close();
233   logfile.close();
234
235   /*  gnuplot 設定初期化  */
236   fprintf(gp, "reset\n");
237
238   /*  サンプルグラフの出力  */
239   fprintf(gp, "set␣terminal␣postscript␣eps␣enhanced␣color\n");
240   fprintf(gp, "set␣output␣'%s'\n", figFile.c_str());
241   fprintf(gp, "plot␣'%s'␣u␣1:2␣w␣l\n", dataFile.c_str());
242
243   pclose(gp);
244
245   fprintf(stderr, "\n\nfinish!\n");
246
247   return 0;
248 }
249
250 /*  関数の設定  */
251 double Apot(double x, double k) {
252
253   double apot;
254
255   apot = 0.5 * k * x * x;
256
```

```
257   return apot;
258 }
259
260 double Hamiltonian(double x, double k) {
261
262   double hamiltonian;
263
264   hamiltonian = Apot(x, k);
265
266   return hamiltonian;
267 }
```

付録　管理者権限がある場合

ここでは管理者権限があるような場合の各パッケージのインストール方法を簡単に述べる。

管理者権限がある場合はパッケージ管理ソフトが使えるので、各ソフトウェアの導入は非常に楽になる。ただし、ものによっては古いソフトウェアが提供されている場合があるため注意が必要である。どうしても最新のものが使いたいという場合は、やはりソースから自分でビルドするしかない。

OS によってパッケージ管理用プラットフォームが異なるため、場合分けを行いながら解説していくこととする。

A　apt(Ubuntu)

apt は Ubuntu 環境において使われているメジャーなパッケージ管理システムである。これは Ubuntu にデフォルトで装備されているので、指定のコマンドを打ち込めば任意のソフトウェアをインストールすることができる。

具体的には、以下のようにする。[*6]

zsh

```
$ sudo apt update
$ sudo apt upgrade
$ sudo apt install build-essential vim zsh ssh git screen gnuplot
```

これでインストール完了である。インストール後の手順に関しては管理者権限がない場合と同様である。

B　yum(Redhat 系,CentOS)

yum は主に CentOS で利用されているパッケージ管理システムである。apt に似ているが、yum は少々取り扱いが厄介である点に気をつけたい。

また、デフォルトでインストールされるパッケージが古いものが多く、自分でビルドしないといけない状況に陥ることがよくある。

各パッケージのインストールは、以下のようにする。[*6]

zsh

```
$ yum -y update --exclude=kernel*
$ yum groupinstall "Development Tools"
$ yum install vim zsh ssh git screen gnuplot
```

これでインストール完了である。インストール後の手順に関しては管理者権限がない場合と同様である。

[*6] prezto に関しては § 3と同じ手順でインストールする。

C　Homebrew(Mac or Linux)

Homebrew は Mac にて主に活躍するパッケージ管理システムである。MacPorts という競合も存在するが、その便利さから Homebrew のほうが主流である。最近では Linux で開発されていた Linuxbrew が本家 Homebrew にフォークされたため、実質 Linux 環境でも導入することができ、運用の幅が広い。

homebrew はデフォルトでインストールされているパッケージではないため、ここではインストール方法から解説する。

C.1　Homebrew のインストール

Homebrew は以下のコマンドを入力することでインストールできる。

Mac の場合

zsh

```
$ xcode-select --install
$ /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
\\ちゃんと入ったかチェック
$ brew doctor
```

Ubuntu の場合

zsh

```
$ sudo apt update
$ sudo apt upgrade
$ sudo apt install build-essential curl file git
$ sh -c "$(curl -fsSL https://raw.githubusercontent.com/Linuxbrew/install/master/install.sh)"
$ test -d ~/.linuxbrew && eval $(~/.linuxbrew/bin/brew shellenv)
$ test -d /home/linuxbrew/.linuxbrew && eval $(/home/linuxbrew/.linuxbrew/bin/brew shellenv)
$ test -r ~/.zshrc && echo "eval \$($(brew --prefix)/bin/brew shellenv)" >>~/.zshrc
$ echo "eval \$($(brew --prefix)/bin/brew shellenv)" >>~/.profile
\\ちゃんと入ったかチェック
$ brew doctor
```

CentOS の場合

zsh

```
$ yum groupinstall 'Development Tools'
$ sudo yum install curl file git
$ sudo yum install libxcrypt-compat
$ sh -c "$(curl -fsSL https://raw.githubusercontent.com/Linuxbrew/install/master/install.sh)"
$ test -d ~/.linuxbrew && eval $(~/.linuxbrew/bin/brew shellenv)
$ test -d /home/linuxbrew/.linuxbrew && eval $(/home/linuxbrew/.linuxbrew/bin/brew shellenv)
$ test -r ~/.zshrc && echo "eval \$($(brew --prefix)/bin/brew shellenv)" >>~/.zshrc
$ echo "eval \$($(brew --prefix)/bin/brew shellenv)" >>~/.profile
\\ちゃんと入ったかチェック
$ brew doctor
```

C.2 各パッケージのインストール

Homebrew がインストールできたらあとは各パッケージのインストールコマンドを打ち込んでやればいいだけである。[*7]

zsh

```
$ brew update
$ brew upgrade
$ brew install vim zsh ssh git screen
$ brew tap ie-developers/ie
$ brew install ie-developers/ie/gnuplot  --with-cairo  --with-aquaterm  --with-x11
```

参考文献

[1]「Zsh + Prezto で快適コマンド環境を構築する」, URL：https://dev.classmethod.jp/tool/zsh-prezto/.

[2]「お前らのターミナルはダサい」, URL：https://qiita.com/kinchiki/items/57e9391128d07819c321.

[3]「Zsh 初心者が zshrc を色々調べたの晒してみる」, URL：https://qiita.com/ryuichi1208/items/2eef96debebb15f5b402.

[4]「SSH 公開鍵認証で接続するまで」, URL：https://qiita.com/kazokmr/items/754169cfa996b24fcbf5.

[5]「多段 ssh を行うときに、ローカルの秘密鍵を参照し続ける」, URL：https://qiita.com/ymd_/items/5eb833ad757bd8b3e6c3.

[6]「多段 SSH の設定を.ssh/config にまとめる」, URL：https://qiita.com/ik-fib/items/12e4fab4478e360a82a1.

[7]「ssh 公開鍵認証設定まとめ」, URL：https://qiita.com/ir-yk/items/af8550fea92b5c5f7fca.

[8]「Git の基礎勉強 ～Git によるバージョン管理を使う～」, URL：https://tracpath.com/bootcamp/learning_git_firststep.html.

[9]「Git の基本的な使い方メモ」, URL：https://qiita.com/opengl-8080/items/451c5967cbbc262f4f0d.

[10]「Linux screen コマンド使い方」, URL：https://qiita.com/hnishi/items/3190f2901f88e2594a5f.

[11]「centos gnuplot ビルド」, URL：https://qiita.com/hnishi/items/4e047969df6f9c2f7828.

[12]「Vim 初心者のための基本的な操作方法のまとめ」, URL：https://qiita.com/tomoya1993/items/7b7dfffd3553e55601ef.

[*7] 現在確認されている issue として、gnuplot のインストール時にオプションが指定できないというものがある。ゆえにここではオプションを指定できるようにするための回避策を実施している。