1　基本的な環境設定

1.1　VMware Tools のインストール

```
$ cd ~/Downloads
$ cp /run/media/rsk0315/VMware\ Tools/VMwareTools-*.tar.gz ./
$ tar xvf VMwareTools-*.tar.gz
$ cd vmware-tools-distrib/
$ sudo ./vmware-install.pl
```

1.2　sudo の設定

以下のコマンドを用いて /etc/sudoers を編集し，いちいちパスワードを求められないようにする．

```
$ sudo visudo
```

差分は次の通り．

```
$ sudo diff -up ~/tmp/sudoers /etc/sudoers
--- /home/rsk0315/tmp/sudoers
+++ /etc/sudoers
@@ -99,7 +99,7 @@ root        ALL=(ALL)       ALL
 %wheel       ALL=(ALL)       ALL

 ## Same thing without a password
-# %wheel     ALL=(ALL)       NOPASSWD: ALL
+%wheel       ALL=(ALL)       NOPASSWD: ALL

 ## Allows members of the users group to mount and unmount the
 ## cdrom as root
```

1.3　git のアップグレード

ビルドするのに必要なライブラリ群をインストールする．

```
$ sudo yum install openssl-devel curl-devel expat-devel
```

gettext-devel，perl-devel，zlib-devel，perl-ExtUtils-MakeMaker も依存している？ 最初から入っているかも？

古い git が入っていることを期待し，以下を実行．

```
$ git clone https://github.com/git/git.git
$ cd git
$ make && make install
```

prefix は $(HOME) になっているので，$PATH を見てちゃんと新しいのが実行されるかを確認する．

1.4　vim のアップグレード

```
$ sudo yum install libX11-devel libXt-devel gtk2-devel ncurses-devel
```

atk-devel も依存している？

```
$ git clone https://github.com/vim/vim.git
$ cd vim
$ ./configure --prefix=$HOME --build=x86_64-redhat-linux --with-x \
>     CFLAGS=-I/usr/include/X11
$ make && make install
```

1.5 emacs のインストール

http://ftp.jaist.ac.jp/pub/GNU/emacs/ などから最新のものをダウンロード.

```
$ sudo yum install gnutls-devel
$ cd ~/Downloads
$ curl http://ftp.jaist.ac.jp/pub/GNU/emacs/emacs-26.1.tar.xz -o emacs-26.1.tar.xz
$ tar xvf emacs-26.1.tar.xz
$ cd emacs-26.1/
$ ./configure --prefix=$HOME --build=x86_64-redhat-linux --without-x
$ make && make install
```

1.6 python3 のインストール

```
$ sudo yum install https://centos7.iuscommunity.org/ius-release.rpm
$ sudo yum install python36u{,-{libs,devel,pip}}
```

1.7 bash のアップグレード

```
$ curl http://ftp.gnu.org/gnu/bash/bash-5.0-alpha.tar.gz -o bash-5.0-alpha.tar.gz
$ tar xvf bash-5.0-alpha.tar.gz
$ cd bash-5.0-alpha/
$ ./configure --prefix=$HOME --build=x86_64-redhat-linux
$ make && make install
$ cp doc/bash.1 ~/usr/share/man/man1/
```

1.8 GCC のアップグレード

```
$ curl http://ftp.tsukuba.wide.ad.jp/software/gcc/releases/gcc-8.2.0/gcc-8.2.0.tar.xz -o gcc-8.2.0.tar.xz
$ tar xvf gcc-8.2.0.tar.xz
$ cd gcc-8.2.0/
$ slack-dog ./contrib/download_prerequisites
$ ./configure --prefix=$HOME --build=x86_64-redhat-linux --program-suffix=-8.2 --disable-multilib --enable-languages=c,c++
$ make -j4 BOOT_CFLAGS='-march=native -O3'
$ make install
```

1.9 その他有用なものたちのインストール

```
$ sudo yum install php
```

LaTeX も早いうちに入れよう．

1.10 LaTeX のインストール

TeX Live を使う．

```
$ sudo yum install perl-Digest-MD5
```

2 発展的な環境構築

趣味の領域に含まれると思われるもの．

2.1 bashrc の編集

最初の状態では以下の内容しかなく，そっけなさすぎる（インストールの手順にもよるとは思うが）．

```
# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
      . /etc/bashrc
fi

# Uncomment the following line if you don't like systemctl's auto-paging feature:
# export SYSTEMD_PAGER=

# User specific aliases and functions
```

2.1.1 エイリアス

ファイルの移動時に上書き確認をするのは基本中の基本．

```
alias rm='rm -iv' cp='cp -iv' mv='mv -iv'
```

2.1.2 シェルオプション

リダイレクトの上書きを防ぐのも基本．

```
# Shell options
set -o noclobber       # same as `set -C'
shopt -s histverify autocd
```

noclobber によって > による上書きを防ぐことができる．強制的に上書きしたいときは >| を使う．histverify は履歴展開が起こったときコマンドが即座に実行されないようにする．autocd はコマンドとして見つからなかった文字列を cd への引数と解釈させる．~ や .. などのみでディレクトリを移動できるようになるが，暴発には注意．

2.1.3 ページャのオプション

```
# Pager configurations
export LESS=Fr
```

- `less -F`：一画面で収まるならそのまま表示
- `less -r`：エスケープシーケンスを解釈

2.1.4 キーバインドの補助設定

`C-s` などが期待通りに動作するようにする.

```
# Key bindings
stty rprnt undef stop undef werase undef kill undef
```

デフォルトに戻す場合は以下の通り.

```
stty rprnt \^r stop \^s werase \^w kill \^u
```

履歴展開が有効な場合に `^` が暴発しないように注意.

2.1.5 `inputrc` の編集

```
# emacs-like key bindings
"\C-u": universal-argument
"\ew": copy-region-as-kill
"\C-w": kill-region
"\e\C-w": kill-whole-line

# variables
set enable-bracketed-paste On
set mark-symlinked-directories On
```

`enable-bracketed-paste` が有効のとき，改行文字が `accept-line` として暴発するのを防ぐ（タブ文字の `complete` なども同様）. `mark-symlinked-directories` が有効のとき，ディレクトリを指すリンクが Tab 補完されたときに `/` が付加される.

2.2 エディタの初期化ファイル

2.2.1 `init.el` の編集

emacs の初期化ファイル.

```
;; テーマの設定
(load-theme 'tango-dark t)

;; 行数を表示
(global-linum-mode t)
(setq linum-format "%4d ")
```

```
;; タブ文字を展開する
(setq-default indent-tabs-mode nil)

;; 括弧の対応付けを表示
(show-paren-mode t)

;; モードラインの整形
(setq
 mode-line-position
 '("%p "
   (line-number-mode " L%l")
   (column-number-mode
    (column-number-indicator-zero-based " c%c" " C%C")))
 column-number-mode t
 column-number-indicator-zero-based nil)

;; その他モジュールの読み込み
(add-to-list 'load-path "~/.emacs.d/elisp")
(require 'markdown)
```

2.2.2 vimrc の編集

```
" 色の設定
:colorscheme koehler
:syntax on

" ステータスバーの設定
:set showcmd
:set ruler

" ショートカット
:nnoremap ZX <Esc>:w<CR><C-z>

" バックスペースの挙動
:set backspace=eol,start,indent
```

2.3 gitconfig の編集

パラメータなどについては以下を参照.

```
$ git help config
```

color.diff の項目を見るとよい. 数値を指定したときに <Esc>38;5;##m の形式になってくれるのはたまたまなのかも？

```
[color.diff]
        new = green bold
        old = red bold
        frag = 43 bold
        commit = 106
```

2.4 フォントの設定

MigMix はいいぞ．/etc/fonts/conf.d/65-nonlatin.conf を編集する．とりあえず一番上にしよう．

```
$ diff -up ~/tmp/65-nonlatin.conf /etc/fonts/conf.d/65-nonlatin.conf
--- /home/rsk0315/tmp/65-nonlatin.conf
+++ /etc/fonts/conf.d/65-nonlatin.conf
@@ -4,6 +4,7 @@
      <alias>
              <family>serif</family>
              <prefer>
+                     <family>MigMix 1M</family>
                      <family>Artsounk</family> <!-- armenian -->
                      <family>BPG UTF8 M</family> <!-- georgian -->
                      <family>Kinnari</family> <!-- thai -->
@@ -69,6 +70,7 @@
      <alias>
              <family>sans-serif</family>
              <prefer>
+                     <family>MigMix 1M</family>
                      <family>Nachlieli</family> <!-- hebrew -->
                      <family>Lucida Sans Unicode</family>
                      <family>Yudit Unicode</family>
@@ -144,6 +146,7 @@
      <alias>
              <family>monospace</family>
              <prefer>
+                     <family>MigMix 1M</family>
                      <family>Miriam Mono</family> <!-- hebrew -->
                      <family>VL Gothic</family>
                      <family>IPAMonaGothic</family>
```

3 便利コマンドの定義

3.1 プロンプト文字列

\s-\v\$ ではそっけないので変える．~/.bashrc に追記．

```
# Prompt strings
PS0=$'\x1b[0m'
PS1="\n\$(. ~rsk0315/.bashrc.d/ps1.sh)"
PS1+=$'\n\[\x1b[0m\]\$ \[\x1b[1m\]'
```

ps1.sh は以下の通りで，

- カレントディレクトリ
- シェルの階層
- バージョン情報

を表示する．

```
# -*- mode: sh; sh-shell: bash -*-
```

```
. ~rsk0315/.bashrc.d/color-seq.sh

print_wd () {
    wd="$(pwd)"
    if [[ "$wd" =~ ^"$HOME" ]]; then
        wd="${wd/#$HOME/$'\176'}"
        [[ "$wd" == \~ ]] || wd+=/
    elif [[ "$wd" == / ]]; then
        wd=/
    else
        wd+=/
    fi
    echo -n "$(color_seq 45 nr)$wd$(color_seq '' n)"
}

print_wd
echo : bash-"$BASH_VERSION" "[$((SHLVL-1))]"
```

color-seq.sh は色のエスケープシーケンスを生成する（関数を定義する）スクリプト．別に関数にする必要はなくて，普通にスクリプトとして置いておいてもいい気がする．

```
# -*- mode: sh; sh-shell: bash -*-

color_seq () {
    color="$1"
    attr="$2"

    res=
    [[ "$attr" =~ n ]] && res+=$'\01'
    res+=$'\x1b['

    if [[ -z "$color" ]]; then
        res+=0
        [[ "$attr" =~ b ]] && res+=\;1
        [[ "$attr" =~ r ]] && res+=\;0
        res+=m
    else
        [[ "$attr" =~ b ]] && res+=1\;
        [[ "$attr" =~ r ]] && res+=0\;
        res+="38;5;${color}m"
    fi
    [[ "$attr" =~ n ]] && res+=$'\02'
    echo "$res"
}
```

3.2 終了ステータス

echo $? を叩かないとわからないのは不便なので，勝手に出してくれるようにする．典型的な値についてはコメントつき．

以下は ~/.bashrc.d/exit-status.sh.

```
# -*- mode: sh; sh-shell: bash -*-
```

```
declare -a signals=(
    [1]='Program received signal SIGHUP, Hangup.'
    [2]='Program received signal SIGINT, Interrupt.'
    [3]='Program received signal SIGQUIT, Quit.'
    [4]='Program received signal SIGILL, Illegal instruction.'
    [5]='Program received signal SIGTRAP, Trace/breakpoint trap.'
    [6]='Program received signal SIGABRT, Aborted.'
    [7]='Program received signal SIGBUS, Bus error.'
    [8]='Program received signal SIGFPE, Arithmetic exception.'
    [9]='Program terminated with signal SIGKILL, Killed.'
    [10]='Program received signal SIGUSR1, User defined signal 1.'
    [11]='Program received signal SIGSEGV, Segmentation fault.'
    [12]='Program received signal SIGUSR2, User defined signal 2.'
    [13]='Program received signal SIGPIPE, Broken pipe.'
    [14]='Program terminated with signal SIGALRM, Alarm clock.'
    [15]='Program received signal SIGTERM, Terminated.'
    [16]='Program received signal ?, Unknown signal.'
    [17]=''
    [18]='Program received signal SIGCONT, Continued.'
    [19]='Program received signal SIGSTOP, Stopped (signal).'
    [20]='Program received signal SIGTSTP, Stopped (user).'
    [21]='Program received signal SIGTTIN, Stopped (tty input).'
    [22]='Program received signal SIGTTOU, Stopped (tty output).'
    [23]=''
    [24]='Program received signal SIGXCPU, CPU time limit exceeded.'
    [25]='Program received signal SIGXFSZ, File size limit exceeded.'
    [26]='Program terminated with signal SIGVTALRM, Virtual timer expired.'
    [27]=''
    [28]=''
    [29]='Program terminated with signal SIGIO, I/O possible.'
    [30]='Program received signal SIGPWR, Power fail/restart.'
    [31]='Program received signal SIGSYS, Bad system call.'
    #[32]='Program received signal SIG32, Real-time event 32.'
)

. ~rsk0315/.bashrc.d/color-seq.sh

declare -gi status=0
on_debug () {
    status=$1
    declare -gi executed+=1
    if (( executed == 0 )); then
        declare -gi failcount=0
        return
    fi

    #if (( status != 0 )); then
    #    declare -gi failcount+=1
    #fi
}

on_err () {
    declare -gi failcount+=1
}

on_prompt () {
```

```
    declare -gi status=$1

    if (( failcount <= 0 )); then
        if (( executed > 1 )); then
            echo -e "$(color_seq 10 r)(*'-')b < All commands exited successfully$(color_seq '')" >&2
        elif (( executed == 1 )); then
            echo -e "$(color_seq 10 r)(*'-')b < Exited successfully$(color_seq '')" >&2
        fi
    elif (( status == 0 )); then
        if (( executed > 1 )); then
            echo -ne "$(color_seq 10 r)(*'-')b < Last command exited successfully, " >&2
            echo -ne "$(color_seq 9 r)but $failcount command" >&2
            if (( failcount > 1 )); then echo -n s >&2; fi
            echo -e " failed$(color_seq '')" >&2
        else
            :;
        fi
    elif (( executed > 0 )); then
        echo -e "$(color_seq 9 r)exited with code $status" >&2

        if (( status < 128 )); then
            case $status in
                1)
                    echo "(*'~')? < Something went wrong?" >&2;;
                2)
                    echo "(*'~')? < Incorrect usage or some error occurred?" >&2;;
                127)
                    echo "(*'~')? < Command not found? Check \$PATH, \$PWD, permissions, and spelling" >&2;;
                *)
                    echo "(*'~')..." >&2;;
            esac
        elif (( ${#signals[status-128]} > 0 )); then
            echo "(*'~')/ < ${signals[status-128]}" >&2
        else
            echo "(*'~')..." >&2
        fi

        echo -ne "$(color_seq '')" >&2
    fi

    declare -gi failcount=-1
    declare -gi executed=-1

    declare -gi status=0
}

trap 'on_debug "$?" "$_"' DEBUG
trap 'on_err "$_"' ERR
PROMPT_COMMAND='on_prompt "$?" "$_"'
```

バックグラウンドで実行したり！で論理反転したりするとこわれるけど仕方ない.

以下を ~/.bashrc に追記して反映させる.

```
# Display previous exit status
. ~rsk0315/.bashrc.d/exit-status.sh
```

3.3 最新ファイルの出力

ls を利用して最新のファイルを選ぶスクリプト．GNU 拡張の ls は や ' などをエスケープできるオプションがあって素敵なんだけど，それを復元するのが厄介なので諦める．そもそもそんなファイル名にする方がどうかしている．

```
# -*- mode: sh; sh-shell: bash -*-

while getopts :p:s: foo; do
    case $foo in
        p ) perm="$OPTARG";;
        s ) suffix="$OPTARG";;
    esac
done

latest=$(ls -ltA | grep -E "${suffix:+\\.(}$suffix${suffix:+)}\$" \
             | grep "^-$perm" | awk '$0=$9' | sed q)
[[ -z "$latest" ]] && exit 1
echo $latest
```

パーミッションと拡張子を指定可能．見つからなければ 1 を返す．

それを利用して最新のソースを make する．俗にいう g に対応するスクリプト．

```
# -*- mode: sh; sh-shell: bash -*-

CC="${CC:-gcc-8.2}"
CXX="${CXX:-g++-8.2}"

cflags="-Wall -O3 -fsanitize=undefined"
CFLAGS="$cflags $CFLAGS $@"
CXXFLAGS="$cflags $CXXFLAGS $@"

src=$(latest -s'c|cxx|cc|C|cpp') \
    || { echo no source files found. >&2; exit; }
make CC="$CC" CXX="$CXX" CFLAGS="$CFLAGS" CXXFLAGS="$CXXFLAGS" ${src%.*}
```

最新の実行ファイルを実行する．俗にいう a に対応するスクリプト．必ずしも ./a.out とは限らないファイルを実行できる．

```
# -*- mode: sh; sh-shell: bash -*-

src=$(latest -p ..x) || { echo no executables found. >&2; exit; }
set -x
"./$src" "$@"
```