Japanese Journal of Lactic Acid Bacteria

解　説

次世代シーケンサーデータの解析手法 第4回 クオリティコントロールとプログラムのインストール

孫 建強[1]、湯 敏[1]、清水 謙多郎[1,2]、門田 幸二[2*]

東京大学大学院農学生命科学研究科
[1]応用生命工学専攻
[2]アグリバイオインフォマティクス教育研究ユニット

次世代シーケンサー（以下、NGS）データの解析は、大まかに①データ取得、②クオリティコントロール（以下、QC）、③アセンブルやマッピング、④数値解析の4つのステップに分けられる。連載第4回は、Bio-Linuxにプレインストールされている FastQC（ver. 0.10.1）プログラムを用いたQCの実行および解釈の基本を述べる。また、FASTX-toolkit（ver. 0.0.14）で提供されているアダプター配列除去プログラムである fastx_clipper の挙動を例に、Linuxコマンドを駆使した動作確認の重要性を述べる。多様なインストール手段に対応すべく、apt-get, pip, cpan コマンドを利用したプログラムのインストール（FastQC ver. 0.11.3、nkf ver. 2.1.3、cutadapt ver. 1.8.1、HTSeq ver. 0.6.1、FaQCs ver. 1.34）やパスなどの各種設定についても述べる。ウェブサイト（Rで）塩基配列解析（URL: http://www.iu.a.u-tokyo.ac.jp/~kadota/r_seq.html）中に本連載をまとめた項目（URL: http://www.iu.a.u-tokyo.ac.jp/~kadota/r_seq.html#about_book_JSLAB）が存在する。ウェブ資料（以下、W）や関連ウェブサイトなどのリンク先を効率的に活用してほしい。

Key words：NGS, Bioinformatics, install, shell script, QC

はじめに

連載第3回[1]では、wgetコマンドを用いてDDBJ SRAから*L. casei* 12AのRNA-seqデータ（SRR616268）を取得した。著者らの仮想環境で数十GBに及ぶ中間ファイルの作成を行うことができたのは、ゲストOSに150GBのHDD容量を確保していたからである（第3回のW25）。ここではまず、前回最後のBio-Linux[2]の状態［**W1-1**］をおさらいし、第4回の初期状態を揃える（**図1; W1-4**）。具体的には、不要なファイルを削除し、①カレントディレクトリ（/home/iu/Documents/srp017156/）中に②2つのbzip2圧縮ファイルのみが存在する状態にしている。著者らの環境では、③カレントディレクトリのみで14GB、④全体でも23GB使われていることがわかる。この程度であれば、ゲストOSに50GB程度しかHDD容量を確保できなかった読者も、新たな気持ちで第4回をスタートさせることができるであろう。

前回同様、本稿を読み始める前に、第4回ウェブ資料に一通り目を通しておくことを勧める。多くの読者は必要最小限のスキルを身につけることを目的としているが、NGS解析においてそのような統一基準はなく、第2回原稿でも書いたように手順、手段、常識の範囲はヒトによって異なる。仮想環境がフリーズして動かなくなることもあるだろう［**W2-1**］。著者らが普段利用しないテクニック［**W3-3**］なども示しているため冗長と感じるかもしれないが、様々なやり方があることを知り、遭遇するエラーへの対処法などを経験しておくことが重要である。特に、chmodコマンドによる実行権限（Permission）の変更操作［**W3-1**］、オートマウントや絶対パス・相対パス指定［**W4**］、OS間での改行コードの違いやその対処法［**W5**］などは理解済みという前提で話を進める。

*To whom correspondence should be addressed.
Phone ：+81-3-5841-2395
Fax ：+81-3-5841-1136
E-mail ：kadota @ bi.a.u-tokyo.ac.jp

```
iu@bielinux[srp017156] pwd  <-------------------------- ①   [12:47午後]
/home/iu/Documents/srp017156
iu@bielinux[srp017156] ls -lh  <----------------------- ②   [ 1:20午後]
total 14G
-rw-rw-r-- 1 iu iu 7.2G 12月  11 14:51 SRR616268_1.fastq.bz2
-rw-rw-r-- 1 iu iu 6.6G 12月  12 14:59 SRR616268_2.fastq.bz2
iu@bielinux[srp017156] du -h  <------------------------ ③   [ 1:20午後]
14G     .
iu@bielinux[srp017156] df -h  <------------------------ ④   [ 1:20午後]
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda1       146G   23G  116G  17% /
none            4.0K     0  4.0K   0% /sys/fs/cgroup
udev            991M  4.0K  991M   1% /dev
tmpfs           201M  916K  200M   1% /run
none            5.0M     0  5.0M   0% /run/lock
none           1002M  156K 1001M   1% /run/shm
none            100M   60K  100M   1% /run/user
iu@bielinux[srp017156] ▉                                  [ 1:20午後]
```

図 1. Bio-Linux 8 のターミナル画面。PC 名は bielinux、ユーザ名は iu。① pwd でカレントディレクトリを表示。② ls コマンドでカレントディレクトリ中のファイルを表示。h オプションは human readable（ヒトが読み取りやすい）の意味。③ du コマンドでカレントディレクトリ中の総ファイルサイズを表示。「ls -lh」で大体の予想はつくが、ファイル数が多いときに便利。④ df コマンドでゲスト OS 全体のディスク利用状況を表示。

```
iu@bielinux[srp017156] ls -lh  <----------------------- ①   [ 7:51午後]
total 14G
-rwxr-xr-x 1 iu iu  360  5月   1 18:01 JSLAB4_1_Linux.sh
-rw-rw-r-- 1 iu iu 7.2G 12月  11 14:51 SRR616268_1.fastq.bz2
-rw-rw-r-- 1 iu iu 6.6G 12月  12 14:59 SRR616268_2.fastq.bz2
iu@bielinux[srp017156] more JSLAB4_1_Linux.sh  <------- ②   [ 7:51午後]
#wget -c ftp://ftp.ddbj.nig.ac.jp/ddbj_database/dra/fastq/SRA061/SRA06148
3/SRX204226/SRR616268_1.fastq.bz2
#wget -c ftp://ftp.ddbj.nig.ac.jp/ddbj_database/dra/fastq/SRA061/SRA06148
3/SRX204226/SRR616268_2.fastq.bz2
bzip2 -dc SRR616268_1.fastq.bz2 | head -n 4000000 > SRR616268sub_1.fastq
bzip2 -dc SRR616268_2.fastq.bz2 | head -n 4000000 > SRR616268sub_2.fastq
iu@bielinux[srp017156] time ./JSLAB4_1_Linux.sh  <----- ③   [ 7:51午後]
./JSLAB4_1_Linux.sh  30.34s user 3.90s system 71% cpu 47.926 total
iu@bielinux[srp017156] ls -lh  <----------------------- ④   [ 7:52午後]
total 15G
-rwxr-xr-x 1 iu iu  360  5月   1 18:01 JSLAB4_1_Linux.sh
-rw-rw-r-- 1 iu iu 7.2G 12月  11 14:51 SRR616268_1.fastq.bz2
-rw-rw-r-- 1 iu iu 6.6G 12月  12 14:59 SRR616268_2.fastq.bz2
-rw-rw-r-- 1 iu iu 306M  5月   1 19:52 SRR616268sub_1.fastq
-rw-rw-r-- 1 iu iu 278M  5月   1 19:52 SRR616268sub_2.fastq
iu@bielinux[srp017156] ▉                                  [ 7:52午後]
```

図 2. ①「ls -lh」でカレントディレクトリ中のファイルを表示。JSLAB4_1_Linux.sh に実行権限が付与されていることがわかる［W3-1］。② more でファイルの中身を表示。ターミナル画面の横幅の関係上、最初の # から始まる 2 行分が 4 行分に見える点に注意。③ time コマンドをつけて実行時間を計測しながら、./JSLAB4_1_Linux.sh を実行。この環境では 30.34 秒かかっていることがわかる。④カレントディレクトリ中のファイルを表示。*.fastq という 2 つのファイルが生成されていることがわかる。

シェルスクリプト

第 3 回の W23-2 では、テキストエディタに一連のスクリプト（コマンド群）を書き込んでおき、ターミナル画面上でそれらをコピー & ペースト（以下、C&P）して実行した。スクリプトを保存したファイルは、投稿論文の supplement としても利用可能である。必要最小限の変更を加えて別のデータセットの解析にも再利用できるだけでなく、コマンド入力ミスも防ぐことができる。**図 2** で示すように、例えば②連載第 3 回で行った乳酸菌 RNA-seq データ（SRR616268）のダウンロードから最初の 400 万行（100 万リード）分のサブセット抽出までのコマンド群を記したファイル（JSLAB4_1_Linux.sh）は、③「./JSLAB4_1_Linux.sh」で実行可能である［**W6-2**］。①では、JSLAB4_1_Linux.sh ファイルに実行権限が付与されている（-rwxr-xr-x）ことを確認している。②計 4 行分のコマンドからなることを more で確認している。最初の 2 行の wget は # でコメントアウトされているため実行

されないようにしている。理由は、既にこのコマンドは実行済み（第3回のW22-2）であり、カレントディレクトリに2つのbzip2圧縮ファイルが存在する。どこから取得したかについての完全な情報を保持しておくのが主な目的である。bzip2から始まるサブセット抽出コマンド自体は第3回のW25-2で行っている。なぜsubset_*.fastqという出力ファイルがなくなっているのか疑問に思っているヒト、上記説明が理解できていないヒトは、本連載のウェブ資料（特に第3回分全てと第4回のW6-2まで）を今一度読み返したほうがいいだろう。カレントディレクトリと入力ファイルの存在および実行権限確認（pwdとls）、JSLAB4_1_Linux.shなどの解析プログラム実行後の出力ファイルの確認は、基本中の基本である。

ここでは、1.35億リードからなるbzip2圧縮ファイルから100万リード分のサブセットを抽出している。オリジナルの100分の1程度のファイルサイズで一通りのデータ解析を行い、動作確認および全データを用いた場合に要する時間の感覚を掴むのが主な目的である。ゲストOSに割り当てているHDD容量にゆとりのあるユーザは10倍の行数（40,000,000：1,000万リードに相当）、ゆとりのないユーザは1/10の行数（400,000：10万リードに相当）からなるサブセットで以後の解析を行ってもよい。もちろん4の倍数であれば、例えば3,572,616行のサブセットでもよい。

クオリティチェック（FastQC ver. 0.10.1）

品質管理（Quality Control; 以下、QC）は、NGS解析分野でも重要な位置を占める。具体的な作業内容は、数億～数十億リードからなるNGSデータの全体的な精度チェック、クオリティの低いリードのフィルタリング、リードに含まれるアダプター配列やクオリティの低い部分配列の除去（トリミング）などである。一連のQCを経たリードのみが、NGSデータ解析の代表的なイメージであるアセンブルやマッピングの入力として用いられる。

リードのクオリティをチェックする目的で最も広く用いられているのは、おそらくFastQCである。Bio-Linuxには一般的によく用いられているプログラムの多くが予めインストール（プレインストール）されている。FastQCもそのうちの1つであり、一般的なLinuxコマンド（ls, cd, pwdなど）と同様に、fastqcというコマンドで利用可能である［**W7**］。FastQC実行の基本形［**W7-3**］、fastqcコマンドで利用可能なオプションの調べ方［**W7-5**］、-qオプションを付加した進捗状況を非表示にするやり方［**W7-6**］、シェルスクリプトでの利用例［**W7-8**］、出力ファイルの共有フォルダへのコピー［**W7-9**］の詳細についてはウェブ資料を参照されたい。**図3**では、③進捗状況非表示（-qオプション）でSRR616268sub_1.fastqファイルに対してFastQCを実行するやり方、⑤プログラムのバージョンの調べ方、⑥zip圧縮ファイルの共有フォルダ（~/Desktop/mac_share）へのコピー手順を示した。

FastQC（ver. 0.10.1）では、SRR616268sub_1_fastqcという名前のディレクトリと.zipがついた圧縮ファイルが作成される。両者は中身が同じである。ゲストOS上でそのまま実行結果を眺めても、共有フォルダ経由でファイルを移動またはコピーしてホストOS上で作業を行ってもよい［**W8**］。htmlレポート中には、入力ファイルのリード数やリード長などの基本統計値［**W8-2**］以外に様々な分析結果が含まれている。主に眺めるのは、塩基のポジションごとのクオリティスコアであろう［**W8-3**］。黄色の箱ひげ図（box plot）でポジションごとのリードのスコア分布が示

```
iu@bielinux[srp017156] pwd  <-------------------------- ①    [11:01午前]
/home/iu/Documents/srp017156
iu@bielinux[srp017156] ls -lh SRR616268sub_* <--------- ②    [11:02午前]
-rw-rw-r-- 1 iu iu 306M  5月  6 10:07 SRR616268sub_1.fastq
-rw-rw-r-- 1 iu iu 278M  5月  6 10:08 SRR616268sub_2.fastq
iu@bielinux[srp017156] fastqc -q SRR616268sub_1.fastq <--- ③    [11:02午前]
iu@bielinux[srp017156] ls -lhd SRR616268sub_* <--------- ④    [11:02午前]
-rw-rw-r-- 1 iu iu 306M  5月  6 10:07 SRR616268sub_1.fastq
drwxrwxr-x 4 iu iu 4.0K  5月  6 11:02 SRR616268sub_1_fastqc
-rw-rw-r-- 1 iu iu 293K  5月  6 11:02 SRR616268sub_1_fastqc.zip
-rw-rw-r-- 1 iu iu 278M  5月  6 10:08 SRR616268sub_2.fastq
iu@bielinux[srp017156] fastqc -v <--------------------- ⑤    [11:02午前]
FastQC v0.10.1
iu@bielinux[srp017156] cp *.zip ~/Desktop/mac_share <---- ⑥    [11:02午前]
iu@bielinux[srp017156]                                       [11:02午前]
```

図3. ①pwdでカレントディレクトリを表示。②SRR616268sub_*の条件を満たすファイルを表示。③SRR616268sub_1.fastqに対するFastQC（ver. 0.10.1）の実行。「fastqc -q *.fastq」とすることで*.fastqという条件を満たす全てのファイルに対してFastQCを実行することもできる。④SRR616268sub_*の条件を満たすファイルを表示。四角で囲ってあるものがFastQC実行結果。-dオプションをつけることで、SRR616268sub_1_fastqcディレクトリの中身を非表示にすることができる［W7-7］。⑤fastqcのバージョン情報を表示。「fastqc -h」で見ると小文字のvの利用が基本のようだが、大文字のVでもOKなようだ［W7-10］。⑥*.zipファイルの共有フォルダへのコピー。相対パスの概念についてはW4-6を参照のこと。

されている。特に注目すべきは、リードのクオリティスコアが平均的にどの程度の値になっているかという絶対的なスコア分布であろう。具体的な指針としては、ほとんどのポジションにおいてスコア分布が 20 以上になっているかどうかである。この閾値（スコア 20）は、100 回中 1 回エラーが起こる確率（p_{err}）に相当する。

この入力データの場合、ほとんどのリードがスコア 30 以上であることが黄色の箱ひげ図の分布から読み取れる。スコア 30 は、1,000 回に 1 回のエラー率（$p_{err}=0.001$）に相当する。つまり、よいデータである。スコアは $-10\times\log_{10}(p_{err})$ として計算されるため、例えばスコア 10 は、10 回中 1 回という高いエラー率（$p_{err}=0.1$）に相当する。そのため、著者らの感覚では、スコア 10 以上という条件設定は非常識である。尚、HiSeq2500 のような比較的最近の NGS 機器（プラットフォーム）から読み取られるリードデータのクオリティは非常に高く、フィルタリング条件設定の閾値として、20 ではなくスコア 30 を採用しても多くのリードが残るようである。また、旧世代シーケンサー同様、読み進めるにしたがって徐々にスコアが下がる傾向にあるが、3' 側の " 相対的な " クオリティスコアの低下はそれほど気にする必要はないだろう。

次に眺めるのは、ポジションごとの塩基の出現確率である［**W8-5**］。RNA-seq は、転写物をランダムに切断して塩基配列を決定するものであり、各塩基の出現確率はポジション間で一定と考えるのが自然である。このデータの場合、最初の 10 数塩基あたりまで出現確率がフラットになっておらず、塩基組成に偏りがあるように見える。用いたアダプター配列と頻出する部分配列情報［**W8-6**］も偏りの原因究明の一助となるであろう。アダプター配列由来であれば、高い出現確率となる塩基のロゴとして浮かび上がることもある。極端な例ではあるが、例えばカイコの small RNA-seq データ（SRR609266）をダウンロードし、ポジションごとの塩基の出現確率を眺めると、原著論文[3] には明記されていないがリードの 3' 側にアダプター配列（このデータの場合は TGGAATTCT…）のロゴが浮かび上がる。

本連載で取り扱っている乳酸菌 RNA-seq データ（SRR616268）は、原著論文がなく実験手順の詳細も不明であるため詳細に立ち入ることは困難である。一般論としては、転写物の断片化法（DNase I または超音波）や用いたプライマーの種類（oligo dT または random hexamer）次第で塩基の出現確率がフラットにならないこともある[4]。データを得た実験手順（アダプター配列情報なども含む）、利用した NGS 機器、および解析目的とこれまでの知見を照らし合わせて総合的に判断するのが正解であろう。

プログラムのインストール（FastQC ver 0.11.3）

一般に、プログラムには様々なバージョンが存在する。FastQC の場合、2014 年 7 月に公開された Bio-Linux 8 上では ver. 0.10.1 が利用可能である。FastQC ウェブサイトの Changelog または Release Notes を眺めると、2015 年 3 月 25 日にリリースされた ver. 0.11.3 が最新であることがわかる（2015 年 5 月 7 日現在）［**W9-1**］。バージョンアップの主な目的は、バグ修正（bug fix）や新機能追加である。例えば、ver. 0.10.1 から ver 0.11.1 へのバージョンアップ時の Changelog に、QC レポート中に Adapter Content という新項目を追加した（Added an adapter content plot）と書かれている。この項目は、FastQC プログラム中に登録されている既知のアダプター配列がリード中にどの程度含まれているかを概観できるものであるが、Bio-Linux 8 にプレインストールされている ver. 0.10.1 では見ることができない。これが現在利用している FastQC プログラムのバージョン情報を把握し、最新バージョンをインストールするモチベーションの 1 つであろう。

同一プログラム（FastQC）の異なるバージョン以外にも、フィルタリングやトリミングという広い意味での QC 用プログラムは数多く存在する。例えば 2014 年 11 月に発表された FaQCs という QC プログラムは、PhiX というコントロール用配列を取り除く機能をもつ[5]。また 2015 年 1 月には、バクテリア RNA-seq データに特化した *de novo* アセンブリプログラム Rockhopper 2 が論文発表されている[6]。これらの比較的新しいプログラムは Bio-Linux にプレインストールされていないため、自力でインストールする必要がある。以下では、FastQC（ver. 0.11.3）のインストール、パスの設定、およびシェル周辺の Linux の作法を解説する。

Linux 上でのプログラムのインストール作業は、「このプログラムを実行するためにはこれが必要で…」という前もって必要な事柄（prerequisite）やプログラムの依存関係（dependency）との格闘である。例えば、FastQC は Java というプログラミング言語で書かれている。そして、Java 自体にも様々なバージョンが存在する。FastQC（ver. 0.11.3）をセットアップするには、Java のバージョンが 1.5 以上であることを予め確認しておき、zip 圧縮ファイルのダウンロード・解凍、および実行権限の付与を行っておくのが基本手順である［**W9-2**］。ただし、一般に（FastQC を含む）Linux 上で動作するプログラムのマニュアルは、Linux の作法をある程度知っているという前提で記述されている。当然、「ある程度」の常識の範囲はヒトそれぞれであり、「どの程度」まで詳細にマニュアルを書くかは開発者次第である。例えば、**W9-2** までの作業を終えた Bio-Linux 8 環境では、2 つのバージョン（ver. 0.10.1 と 0.11.3）の FastQC プログラムが共存している。それゆえ、デフォルトでは、/usr/local/bin に存在する（つまりパスが通っている）ver. 0.10.1 の FastQC 実行コマンド（fastqc）が優先的に実行されることを正しく認識することが第一歩である［**W9-4**］。このマニュアルではシンボリッ

ク・リンクでパスを通す方法を記述している。/path/to/FastQC/fastqc に相当するものが、ユーザ iu のホームディレクトリ直下の Downloads にダウンロードした /home/iu/Downloads/FastQC/fastqc あるいは ~/Downloads/FastQC/fastqc であることが理解できなければ先には進めない。本稿では、コマンド fastqc が ver. 0.10.1、そして fastqc2 が ver. 0.11.3 を指すものとして 2 つのバージョンの FastQC を共存させるやり方を示している［**W9-5**］。

プログラムによっては、シンボリック・リンクではなく、パスの環境変数 $PATH に任意のディレクトリパスを追加するやり方の基本形が記載されているかもしれない［**W9-10**］。これらの知識は、インストールしたプログラムを最低限実行する上では不要な事柄ではある。しかし、知らなければ要・不要の判断すらつかないため、適切に取捨選択できる程度の知識は得ておくべきだろう。設定ファイル /etc/rc.local 中に自動マウント設定情報を書き込んでおけば心穏やかに共有フォルダを利用できるのと同様［**W4-5**］、OS がデフォルトで利用しているシェルの種類（zsh, bash, tcsh など）を把握し［**W10-3**］、そのシェルに対応した rc または profile ファイルにパスを追加するスキルを身につけて、作業効率の向上を実感してもらいたい［**W10-6**］。

アダプター配列除去（FASTX-toolkit ver. 0.0.14）

FastQC 実行結果の Overrepresented sequences という項目を眺めると、既知のアダプター配列がいくつか含まれていることがわかる［**W8-6**］。リード中に含まれるアダプター配列は人工的に付加したものであるため、多くのプログラムがアダプター配列除去に対応している［**W11-1**］。しかしフリーソフトは、自己責任での利用が基本である。「このプログラムを実行するとどういう結果が得られるか」についてある程度予想を立て、変な結果が得られていないかどうか疑いの目を持ってチェックする姿勢が大事である。ここでは、Bio-Linux 8 にプレインストールされている FASTX-toolkit（ver. 0.0.14）中の fastx_clipper というアダプター配列除去プログラムの挙動を示し、上記視点の重要性を例示する［**W11-2**］。

全部で 100 万リード、107 bp からなる SRR616268sub_1.fastq.gz の FastQC（ver. 0.10.1）実行結果として、既知のアダプター配列は多いものから 2,415 回（TruSeq Adapter, Index 3; 50 塩基一致）、1,539 回（TruSeq Adapter, Index 2; 50 塩基一致）、そして 1,318 回（TruSeq Adapter, Index 2; 49 塩基一致）出現している［**W8-6**］。この結果は、FastQC（ver. 0.11.3）においても不変であることを確認済みである。FastQC（ver. 0.11.3）出力ファイルの Adapter Content を眺めると、リードの全ポジションにおいてアダプター配列の割合が 0% 付近にあることが

```
iu@bielinux[srp017156] pwd                                    ① [ 7:51午後]
/home/iu/Documents/srp017156
iu@bielinux[srp017156] ls -lh *sub*                           ② [ 7:51午後]
-rw-rw-r-- 1 iu iu 72M  5月  6 10:07 SRR616268sub_1.fastq.gz
-rw-rw-r-- 1 iu iu 65M  5月  6 10:08 SRR616268sub_2.fastq.gz
iu@bielinux[srp017156] time gunzip SRR616268sub_1.fastq.gz    ③ [ 7:51午後]
gunzip SRR616268sub_1.fastq.gz  2.24s user 0.44s system 56% cpu 4.698 total
iu@bielinux[srp017156] wc SRR616268sub_1.fastq                ④ [ 7:51午後]
  4000000   8000000 320760716 SRR616268sub_1.fastq
iu@bielinux[srp017156] time fastx_clipper \                   ⑤ [ 7:52午後]
> -a GATCGGAAGAGCACACGTCTGAACTCCAGTCACTTAGGCATCTCGTATGC \
> -i SRR616268sub_1.fastq \
> -o hoge1.fastq
fastx_clipper -a GATCGGAAGAGCACACGTCTGAACTCCAGTCACTTAGGCATCTCGTATGC -i  -o
 126.96s user 9.97s system 85% cpu 2:39.97 total
iu@bielinux[srp017156] fastqc2 -q hoge1.fastq \               ⑥ [ 7:54午後]
> --outdir=/home/iu/Desktop/mac_share
iu@bielinux[srp017156] wc hoge1.fastq                         ⑦ [ 7:55午後]
  3915596   7831192 286011042 hoge1.fastq
iu@bielinux[srp017156] ▉                                        [ 7:58午後]
```

図 4. ① pwd でカレントディレクトリを表示。② *sub* の条件を満たすファイルを表示。③ time コマンドで処理時間を計測しつつ、gzip 圧縮ファイルを解凍。理由は fastx_clipper が圧縮ファイルを入力として受け付けないから。④ SRR616268sub_1.fastq ファイルの行数が 400 万行であることを確認。⑤ time コマンドで処理時間を計測しつつ、fastx_clipper コマンドの実行。長いコマンドなので、視認性向上のため「\（逆スラッシュ）」を用いて明示的に複数行で記述している［W9-8］。-a でアダプター配列（ここでは TruSeq Adapter Index 3）を指定、-i は入力ファイル（SRR616268sub_1.fastq）、-o は出力ファイル名（hoge1.fastq）を指定する。2 分強かかる。⑥ hoge1.fastq を入力として FastQC（ver. 0.11.3）を実行。結果は共有フォルダである /home/iu/Desktop/mac_share に出力するように指定［W9-7］。⑦ hoge1.fastq の行数を確認。3,915,596 行となっていることがわかる。

わかる［**W11-4**］。これは、多いものでも 100 万リード中 2,415 回しか出現しないことからも納得できる。これらの結果から、この 100 万リードからなるサブセットを用いた下流の解析に限っていえば、アダプター配列除去の有無はほとんど影響を及ぼさないと予想される。

図 4 は、107 bp のリードに対して、50 bp のアダプター配列（TruSeq Adapter, Index 3）を与えて fastx_clipper を実行する一連のコードを示している［**W11-6**］。④入力ファイル（SRR616268sub_1.fastq）が 4,000,000 行であり、リード長が 107bp とアダプター配列長が 50 bp であることから、アダプター配列除去後の出力ファイル（hoge1.fastq）の行数は 4,000,000 行であり、リード長の分布は最短でも（107 − 50）bp であるはずというのが著者らの動作確認時のチェックポイントである。しかし、⑦アダプター配列除去後のファイル（hoge1.fastq）に対する wc 実行結果から、このファイルは 3,915,596 行となっていることが分かる。また、hoge1.fastq のリード長分布を眺めると、最短で 5 bp のリードが存在する［**W11-7**］。これは直観的に変である。「fastx_clipper -h」でコマンドマニュアルを眺め［**W11-5**］、N を含むリードを残す -n オプション、およびアダプター配列除去後に 5 bp 未満になっても出力する -l 0 オプションを併用すると、3,965,948 行まで行数が増える［**W11-8-5**］。しかしまだ何かが足りないようである。著者らは、後にアダプター配列のみからなるリードを出力する -k オプション単体での実行結果ファイルの行数（34,052 行）を合わせるとめでたく 4,000,000 行となることに気づいた。それでもなぜ「アダプター配列のみからなるリード」が存在しうるのかはいまだに理解不能であり、理解の範囲を逸脱した結果は通常「プログラムのバグ」と断定される。

Linux コマンド習得の意義

少なくとも著者らは、フリーソフトのマニュアルなど開発者側が提供する情報のみでは、プログラムのオプションなどの挙動を完全には理解できない。それゆえ、上述の fastx_clipper プログラムの動作確認のように、実際にいくつかのオプションを駆使して得られた結果に納得がいかない場合は、Linux コマンドを駆使して同様な操作を行う。そして得られた「正解の結果」との比較を行う。例えば、**図 5** の③と④の結果から、50 bp のアダプター配列を行頭に含むリード数は 2,415 個が、そして行頭以外の全 107 bp のどこかにアダプター配列を含むリード数は（2,812 − 2,415）個が正解であるという情報を得る。そしてオプションをどう駆使しても Linux コマンド由来の正解と同数の結果が得られない場合は、そのプログラムは利用しないほうがよい。もちろんどの程度のオプションの組合せを試すかはエンドユーザの忍耐力次第ではある。著者らは、W11 で徹底的に動作確認をしたが、この労力は本稿での詳細な解説のためだけである。エンドユーザの立場では、W11-6 までの作業で fastx_clipper の利用をあきらめ、同様の機能を果たす別のプログラムの調査・インストール・利用に

```
iu@bielinux[srp017156] pwd ←――――――――――――――――――――――――― ①[ 8:48午後]
/home/iu/Documents/srp017156
iu@bielinux[srp017156] wc SRR616268sub_1.fastq ←――――――――――― ②[ 8:48午後]
  4000000   8000000 320760716 SRR616268sub_1.fastq
iu@bielinux[srp017156] grep -c GATCGGAAGAGCACACGTCTGAACTCCAGTCACTTAGGC
ATCTCGTATGC SRR616268sub_1.fastq ←――――――――――――――――――― ③
2812
iu@bielinux[srp017156] grep -c ^GATCGGAAGAGCACACGTCTGAACTCCAGTCACTTAGG
CATCTCGTATGC SRR616268sub_1.fastq ←――――――――――――――――――― ④
2415
iu@bielinux[srp017156] grep ^GATCGGAAGAGCACACGTCTGAACTCCAGTCACTTAGGCAT
CTCGTATGC SRR616268sub_1.fastq | head -n 3 ←――――――――――――― ⑤
GATCGGAAGAGCACACGTCTGAACTCCAGTCACTTAGGCATCTCGTATGCCGTCTTCTGCTTGAAAAGAA
TCTGAGCAAACCCCCTGGTTGTCGGGGCACGGAATAC
GATCGGAAGAGCACACGTCTGAACTCCAGTCACTTAGGCATCTCGTATGCCGTCTTCTGCTTGAAAAAAA
AAATAAAAGAAAAAGACAACTCAAGCATGGGGGATGG
GATCGGAAGAGCACACGTCTGAACTCCAGTCACTTAGGCATCTCGTATGCCGTCTTCTGCTTGAAAAAAA
CAAAAACCTTAGTGAACAAACATCGGTCACGGGTCCG
iu@bielinux[srp017156] grep ^GATCGGAAGAGCACACGTCTGAACTCCAGTCACTTAGGCAT
CTCGTATGC SRR616268sub_1.fastq > age.txt ←――――――――――――――― ⑥
iu@bielinux[srp017156] less age.txt ←――――――――――――――――――― ⑦[ 8:48午後]
```

図 5. ① pwd でカレントディレクトリを表示。② SRR616268sub_1.fastq ファイルの行数が 400 万行であることを確認。③指定したアダプター配列（GATCG…TATGC）をどこかに完全一致で含む行数は 2,812。④アダプター配列を行頭で完全一致で含む行数は 2,415。^ は正規表現で行頭の意味。⑤結果を行数ではなく（つまり -c オプションをつけないで）行そのものを出力させている。「| head -n 3」をつけることで最初の 3 行分のみ表示。確かに ^ をつけているから灰色下線で示すようにアダプター配列と完全一致のものが見られる。⑥ grep 結果をリダイレクト（>）して age.txt というファイルに書き出したのち、⑦ less で眺めてアダプター配列をキーワード検索するとわかりやすい。

切り替える。

Linux コマンド群とそのオプションを駆使することで、思いつく解析の多くを実現可能である。例えば**図 5** で解説したような特定の条件を満たすリード数の集合演算を手軽に行うことができる。FastQC 実行結果［**W8-6**］において、リード長よりも短い 50 bp で Overrepresented sequences が得られるが、実はこれはリードの最初の 50 塩基のみをカウントした結果であることも grep で分かる［**W11-10**］。全 107 bp のどこかにアダプター配列を含む 2,812 リードをリダイレクト（>）で任意のファイル名（uge1.txt）で保存し［**W11-11 の①**］、今度はそのファイルを入力として行頭にアダプター配列を含まない（2,812 − 2,415 = ）397 行分のリードを -v オプションを用いた grep で抽出し、別ファイル（uge2.txt）に保存するのである［**W11-11 の③**］。行頭にアダプター配列を含まないことが分かっている 397 リードの uge2.txt を head や more コマンドで眺めて目で確認（通称、目 grep）してもよいが、less コマンドでファイルを開き、50 bp のアダプター配列で検索・ハイライトすれば一目瞭然である［**W11-12**］。また、このような一連の検証を通じて、アダプター配列は 50 bp ではなくもっと長いのでは ?! という疑念を抱く。そして FastQC 実行結果［**W8-6**］から得られる「TruSeq Adapter, Index 3」でキーワード検索することで、アダプター配列の長さが 63 bp であることや Index（インデックス配列）の概念を知り、知識の幅を広げていくのである［**W11-13**］。

FastQC 実行結果から、少なくともこのデータ中には Index 2 と 3 のアダプター配列が含まれていることがわかる。また、インデックス部分の違いを考慮した一括検索を併用することで、この乳酸菌 RNA-seq データ（SRR616268）中のアダプター配列を含むリード数を大まかに推定することもできる。いくつかのポジティブコントロール（Index 2 と 3）＋ α で動作確認をしたのち、シェルスクリプトや正規表現を駆使して目的を達成しているが、オプションをある程度知っていれば数分程度の作業量である［**W12-1**］。原著論文や公共 DB 中に実験手順やアダプター配列に関する詳細な記述があれば、それらの情報をもとにした確認作業という位置づけになる。このデータの場合、TruSeq Adapter, Index 1-27 のいずれかを 100% 一致で含むリードが 5,906 個、TruSeq Adapter, Index 1-3 のいずれかを含むリードが（619 + 2,625 + 2,603）個であることがわかる。これらの結果から、差分に相当する 59 リードは Index 1-3 のいずれかのアダプター配列を含むものであり、Index 部分のクオリティスコアが低いために完全一致しなかったのだろうと推測される。結果の解釈はヒトそれぞれであろうが、Linux コマンドを駆使すれば、疑問点を迅速に解決し次の展開に速やかに移行可能であるという点については衆目の一致するところだろう。

プログラムのインストール（nkf）

OS 間（Windows ⇔ Linux）の文字コード変換手段は複数存在する。本稿では Perl で行う手順［**W5-2**］を示したが、「Linux 文字コード変換」でウェブ検索すると iconv や nkf コマンドを利用するやり方も存在することがわかる。Bio-Linux 8 には iconv はあるが nkf コマンドはないため、ここでは nkf プログラムのインストール手順を示す。結論のみを書くと、「sudo apt-get install nkf」と打てばよい［**W13-5**］。ここで、sudo は管理者権限で実行するという意味であり、apt-get が主要なコマンドである。apt-get 自体は、プログラムのインストール（install）だけでなく、アップグレード（upgrade）やアンインストール（remove）を行う機能をもっている。ここでの目的はプログラムのインストールであるため、「sudo apt-get install」までを一塊のものとみなせばよい。同様に、もし任意のプログラムのアンインストールを行いたい場合は「sudo apt-get remove *プログラム名*」を試せばよい［**W13-7**］。R パッケージのインストールを行った経験のある読者向けの説明としては、「apt-get install」は install.packages や biocLite 関数に相当するものだという理解でよい。尚、apt-get に関するウェブ上の情報は「プログラム」ではなく「パッケージ」となっている場合が多いが、表現の仕方が異なるだけで実質的に同じものである。

プログラムのインストール（Python パッケージ）

NGS 解析用プログラムの多くは、Python や Perl というインタプリタ型プログラミング言語で記述されている [7]。Python パッケージで用意されているものとしては、アダプター配列除去を行う cutadapt [8]、マップされたリードのカウントなどを行う HTSeq [9]、HTSeq が内部的に利用する PySam、ヒトゲノムの多型や変異を取扱う hgvs [10] などが挙げられる。多くの Python パッケージは Python Package Index（PyPI）上に置かれている。文献データベース PubMed の検索結果で見られる Abstract 中に「https://pypi.python.org/pypi/*パッケージ名*」という URL が記載されている場合は、ここで紹介する手順通りにやれば大抵インストールできる。プログラミング言語ごとに作成されたパッケージ（or モジュール or ライブラリ）の巨大な格納庫が存在し、その Python 版が PyPI である。R パッケージは The Comprehensive R Archive Network（CRAN）または Bioconductor 上に置かれている、Perl モジュールは The Comprehensive Perl Archive Network（CPAN）上に置かれている、などと読み替えれば PyPI の位置づけがわかるであろう。

Python パッケージをインストールする際に用いると便利なコマンドは pip である。Bio-Linux 8 には pip がプレインストールされていないが、「sudo apt-get -y

install python-pip」で pip を利用可能である［**W14-1**］。pip コマンドを用いたパッケージインストールの基本形は「sudo pip install *パッケージ名*」である。それゆえ、cutadapt パッケージをインストールする場合には「sudo pip install cutadapt」と打つだけでよい［**W14-3**］。同様に、cutadapt パッケージをアンインストールしたい場合は「sudo pip uninstall cutadapt」と打てばよい。

次に HTSeq パッケージのインストール手順を示す。開発者のウェブページでは、Linux の種類ごとに前もって必要な事柄（prerequisite）の確認や、pip コマンドを用いないインストール手順が書かれている［**W14-4**］。記述内容の解読は慣れである。Bio-Linux 8 が Ubuntu ベースのものであることを思い出し、nkf のインストールは「sudo yum …」ではなく「sudo apt-get …」のほうであったことなどの経験を積み重ねる以外にはないだろう。精神的疲労感は大きいが、作業自体は C&P で進む場合が多い。著者らは、マニュアル中の「To install HTSeq itself, …」という記述を見つけた段階で、これ以降の作業が「sudo pip install htseq」に相当すると判断して置換した。

プログラムのインストール（FaQCs ver. 1.34）

オープンソースで開発しているプログラムの多くは、GitHub という分散型バージョン管理システム上で公開されている。前述の QC 用プログラム FaQCs [5] はそのうちの 1 つである。ソースコードのダウンロード手段は 2 つある。1 つは GitHub ウェブページからの zip 圧縮ファイルのダウンロード、そしてもう 1 つは git コマンドを利用したダウンロードである。git コマンドは Bio-Linux 8 にプレインストールされていないため、「sudo apt-get install git」を行っておく［**W15-1**］。ここでは FaQCs を単純に利用したいだけなので Git や GitHub の詳細には立ち入らないが、基本的に「git clone *プログラムの URL*」で目的のプログラムのソースコードを入手できる。*URL* の部分は、*URL*.git と書くのが一般的なようであるため、そのように記述してもよい。FaQCs の場合は、「git clone https://github.com/LANL-Bioinformatics/FaQCs」と打てばよい［**W15-2**］。この作業自体は、プログラムのインストールではなくダウンロードである。そのため、git clone は wget コマンドと似たようなものという捉え方でもよい。

FaQCs のウェブサイトにも prerequisites の項目がある［**W15-3**］。第 1 項目は、「メインプログラムはプログラミング言語 Perl ver. 5.8.8 を用いて開発された」というものである。エンドユーザは、Bio-Linux 8 上でインストールされている Perl のバージョンが 5.8.8 以上であるかどうかを「perl -v」で確認してもよい。しかし、ほぼ全ての Linux には Perl などの基本的なものがプレインストールされているため、著者らはこの種の記述は特に気にしない。全 5 項目のうち、例えば第 2 項目の「Parallel::ForkManager module from CPAN」は、全てが暗号にしか見えない読者も多いだろう。これは、Parallel::ForkManager という名前の CPAN 上に置かれている Perl モジュール（というものが FaQCs.pl を実行する上で必要だということ）を意味する。よく見ると、Note に「./INSTALL.sh を実行すれば、これら 2 つの Perl モジュール（Parallel::ForkManager と String::Approx）がインストールされる」と書かれているので、マニュアルに従い実行する［**W15-4**］。これでインストールが完了である。引き続いてマニュアルを見ると、利用例（Basic usage と Full usage）が書かれている。利用例を眺めることで、著者の環境では /home/iu/Downloads/FaQCs/FaQCs.pl が FaQCs プログラムの実体であると認識する。

FastQC（ver. 0.11.1）のインストールマニュアルにも記載されているように［**W9-5**］、一般にインストールしたプログラムを簡便に利用できるようにパスを通す作業を行う。FaQCs のマニュアルには書かれていないが、FaQCs.pl の場合は、絶対パス（/home/iu/Downloads/FaQCs/FaQCs.pl）や相対パス（~/Downloads/FaQCs/FaQCs.pl）で示すことなく、どのディレクトリ上でも FaQCs.pl というプログラム名のみで利用できるようにすることを意味する。FastQC では、/usr/local/bin にシンボリック・リンクを張る手段［**W9-5**］と .zshrc ファイル中の $PATH に /home/iu/Downloads/FastQC を追加する手段［**W10-3**］の両方を示した。著者らは、前者の方法で FaQCs.pl のパスを通し［**W15-5**］、「FaQCs.pl -h」での動作確認時にエラーに遭遇した［**W15-6**］。

FaQCs に限らず、手順通りのインストール作業中や作業後の動作確認時にエラーに遭遇することはよくある。その場合、著者らは、まず最低限動作する手段を探す。①「./INSTALL.sh」実行前後の状態を思い返し、この作業自体に問題はなかったと判断した［**W15-7**］。次に、②シンボリック・リンクを張ったことに起因する可能性を考えた。具体的な検証手段は、FaQCs.pl が存在するディレクトリ（~/Downloads/FaQCs/）上で「./FaQCs.pl -h」がうまく動くことを確認した［**W15-7**］。この段階で最低限の実行手段を得たことになるので、基本的にこれ以降は FaQCs に限って言えばプラスアルファの作業となる。「FaQCs.pl -h」でのエラーの内容は「Perl モジュールがインストールされていないのでは ?!」というものであった。しかし、①でインストール自体は問題なさそうであることから、Perl モジュールのパスがうまく通っていないだけだろうと考える［**W15-8**］。具体的な対処法としては、③「sudo cpan *Perl モジュール名*」として独立に Perl モジュールをインストール［**W16-2**］することで「FaQCs.pl -h」に成功した［**W16-3**］。

このプラスアルファの努力は無駄ではない。理由は、この努力のおかげで “FaQCs の利用例どおりに ” アダプター

配列除去やクオリティフィルタリングなどのQCを行えるようになるからである［**W17**］。利用例の（パスが通っているという前提で記述されている）FaQCs.pl部分を~/Downloads/FaQCs/FaQCs.plなどに置き換えてアドホックに対応するユーザは結果としてほとんどいない。理由は、多くの中上級者はどうにかして利用例どおりにFaQCs.plのみで動かせるようにする。そして、多くの初心者を含むそれ以外のユーザは、自力で必要な箇所を置き換える発想を持っていないからである。

おわりに

多くのプログラムのインストールマニュアルは、必要最低限の情報しか書かれていない。Linuxに関しては、著者らはその方針に賛同する。理由は、初心者を意識してあまり丁寧に場合分けして記載しても、それ自体が理解できない場合が多いからである。豊富な経験と知識でプログラム開発者の思考回路や意図を汲み、様々な迂回路を駆使して最新のNGS解析用プログラムを利用する姿勢が、特に乳酸菌を含む非モデル生物を解析するエンドユーザにとっては重要であろう。本連載で特に力を入れたウェブ資料こそが、豊富な経験と知識に相当するものである。連載第5回では、QC適用前後のマッピングやアセンブルの違いについて述べる予定である。

謝　辞

本連載の一部は、国立研究開発法人科学技術振興機構　バイオサイエンスデータベースセンター（NBDC）との共同研究の成果によるものです。

参 考 文 献

1） 孫建強，三浦文，清水謙多郎，門田幸二（2015）次世代シーケンサーデータの解析手法：第3回Linux環境構築からNGSデータ取得まで．日本乳酸菌学会誌 **26**：32-41.
2） Field D, Tiwari B, Booth T, Houten S, Swan D, et al. (2006) Open software for biologists: from famine to feast. Nat Biotechnol **24**: 801-803.
3） Nie Z, Zhou F, Li D, Lv Z, Chen J, et al. (2013) RIP-seq of BmAgo2-associated small RNAs reveal various types of small non-coding RNAs in the silkworm, Bombyx mori. BMC Genomics **14**: 661.
4） Hansen KD, Brenner SE, Dudoit S. (2010) Biases in Illumina transcriptome sequencing caused by random hexamer priming. Nucleic Acids Res **36**: e131.
5） Lo CC, Chain PS (2014) Rapid evaluation and quality control of next generation sequencing data with FaQCs. BMC Bioinformatics **15**: 366.
6） Tjaden B (2015) De novo assembly of bacterial transcriptomes from RNA-seq data. Genome Biol. **16**: 1.
7） 門田幸二，孫建強，湯敏，西岡輔，清水謙多郎（2014）次世代シーケンサーデータの解析手法：第1回イントロダクション．日本乳酸菌学会誌 **25**：87-94.
8） Martin M (2011) Cutadapt removes adapter sequences from high-throughput sequencing reads. EMBnet. journal. **17**: 10-12.
9） Anders S, Pyl PT, Huber W (2015) HTSeq—a Python framework to work with high-throughput sequencing data. Bioinformatics **31**: 166-169.
10） Hart RK, Rico R, Hare E, Garcia J, Westbrook J, et al. (2015) A Python package for parsing, validating, mapping and formatting sequence variants using HGVS nomenclature. Bioinformatics **31**: 268-270.

Methods for analyzing next-generation sequencing data IV. FASTQ quality control and program installation

Jianqiang Sun[1], Min Tang[1], Kentaro Shimizu[1, 2], and Koji Kadota[2]

[1]Department of Biotechnology, [2]Agricultural Bioinformatics Research Unit, Graduate School of Agricultural and Life Sciences, The University of Tokyo.

Abstract

RNA-seq differential expression analysis workflow generally consists of four steps: (i) retrieving data, (ii) quality control (QC), (iii) de novo assembling and/or read mapping, and (iv) statistical analysis. We here focus on the second step QC and tools for removing adapter sequences. We explain the two programs (FastQC ver. 0.10.1 and FASTX-toolkit ver. 0.0.14) that are pre-installed in the Bio-Linux 8. We also exemplify how to install a total of five programs (FastQC ver. 0.11.3, nkf ver. 2.1.3, cutadapt ver. 1.8.1, HTSeq ver. 0.6.1, FaQCs ver. 1.34) using apt-get, pip, and cpan.