応用編

がんの全ゲノム解析

令和5年度

目次

第 Ⅰ 章
Linux 入門
（基本操作、コマンド、ファイル操作 等）

弘前大学大学院医学研究科附属

健康・医療データサイエンス研究センター　センター長

玉田嘉紀

概要

本テキストでは応用編第Ⅰ章として、ゲノム解析に必須となる Linux（リナックス）の基本的な操作方法を学びます。

ゲノム解析などのバイオインフォマティクス解析では Linux という OS（Operating System）で動作するソフトウェアを多く利用します。つまり、ゲノム解析ツールを使いこなすためには Linux の使い方に習熟しておく必要がある、ということになります。また大規模なゲノム解析は、遠隔にある高性能なコンピュータ（サーバ）やスーパーコンピュータ（大型計算機）上で行うことも多く、それらの OS としてほとんどの場合 Linux が採用されています。従って、そういったサーバを利用するためにも Linux の習得は必須です。

Linux の操作は、絵で表現された画面上の部品をマウスやトラックパッドなどで操作するのではなく、文字による入出力が中心となります。直感的な使い方ができず難しいとされますが、習得すると非常に柔軟で効率的なコンピュータの利用ができるようになります。

第 Ⅱ 章以降では、本章で説明されている知識を前提としていますので必要に応じて適宜本章に戻って操作方法を確認するようにしてください。

OS と Linux

Linux は OS と呼ばれる、コンピュータを動作させるための基本となるソフトウェアの1つです。
まず OS と Linux について簡単に説明します。

OS（Operating System） とは、コンピュータで様々なソフトウェア（プログラム）を動作させるための基盤となるソフトウェアです。日本語では「基本ソフトウェア」と呼ばれたりもします。OS はキーボードなどの入力装置からの入力、ファイルの入出力や操作、画面表示などの基本的な機能を提供します。また CPU やメモリ、ネットワークやストレージなどを管理したり、それらへのアクセス方法を提供し、ソフトウェアから統一的に利用できるようにしています。一方、Word や Excel などのオフィス・ソフトウェアやこれから勉強するさまざまなバイオインフォマティクス解析ツール・ソフトウェアは OS 上で動作することから「アプリケーション（応用）・ソフトウェア」と呼ばれます。アプリケーション・ソフトウェアは、OS の提供している機能を利用することで、コンピュータというハードウェア上で動作しています。

OS にはさまざまな種類のものが存在します。代表的なものは Microsoft 社の Windows や Apple 社の MacOS です。またスマートフォンで利用されている Android や iOS も OS の1つです。Linux もそのような OS の1つです。

Linux はさまざまな人々が共同で開発している OS で、無償で自由に利用することができます。設計図（ソース・コード）を自由に見ることができ、改変も自由にできるのが特徴の1つです。現在、ネットワークを介して利用する大型・高性能コンピュータ（サーバ）用の OS として広く普及しています。前述の通り、バイオインフォマティクス用アプリケーションも Linux 向けに開発されているものがほとんどです。サーバ用 OS に Unix（ユニックス）というものがありますが、Unix の標準規格が定められており、Linux もこの標準に準拠しています。従って Linux は Unix の一種とも言えます。

Linux ではアプリケーション・ソフトウェアの開発環境も無償で自由に利用できます。また複数のソフトウェアを組み合わせて利用することが、比較的容易にできるため、さまざまな処理を組み合わせて解析を行うバイオインフォマティクスのアプリケーションととても相性が良いことも、それらが Linux で開発されている理由と言えるでしょう。従って、Linux の操作を習得することが、バイオインフォマティクスツールを使いこなす最初の一歩となります。

Linux とディストリビューション

Linux と一言で言っても、Red Hat や Ubuntu、CentOS など、さまざまな「ディストリビューション」と呼ばれる Linux のバリエーションがあります。Linux 自体は、OS の核となる部分（＝カーネル）のみを指します（→ P.16）。そのカーネルを動作させるために付随する基本的なプログラム群やアプリケーションは、Linux とは独立して開発されているものもあり、Linux のディストリビューションごとに、採用されているソフトウェアや OS 周辺の仕組み、アプリケーションのパッケージ管理システムなどに差があります。従って同じ Linux でも必要な操作が違う場合がありますが、基本的な考え方や操作は共通ですので安心してください。

サーバの遠隔利用とCUI

Linux は、ネットワークを介して遠隔利用する場合が一般的です。通常のパソコンと利用形態の違いについて説明します。

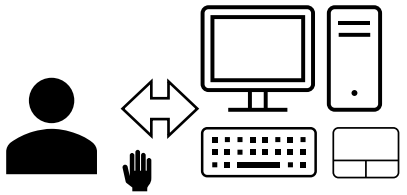
Linux で動作するコンピュータ（サーバ）は、ほどんどの場合直接操作せず、ネットワークを介して手元にある別の PC から遠隔で利用します。通常の PC では、画面に絵で表現されたファイルやプログラムなどをマウスで操作しますが、遠隔利用では、コンピュータの操作は文字入力による操作が中心になります。絵で表現された操作を GUI (Graphical User Interface) といい、文字による操作を **CUI (Character User Interface)** といいます。

現在では、遠隔利用でも GUI でコンピュータの操作を行うことは可能ですが、CUI を利用することがまだ一般的です。CUI は習得に時間がかかりますが、操作の自動化や効率、操作記録の保存・再利用などの点で利点が多くあります。バイオインフォマティクスのツールも多くは CUI を前提としています。

Linux のコンピュータを遠隔で利用するには手元にある PC からネットワークを介してそのコンピュータにまず「ログイン」します。手元にある PC を「ローカル端末」や「ローカル PC」と言ったりします。ローカル PC では、遠隔利用のためのソフトウェアが必要です。一般にターミナル（端末）・ソフトウェア（アプリ）と呼ばれます。また Linux サーバに接続するには通常 **SSH (Secure Shell)** と呼ばれる暗号化された安全な通信規格を利用します。したがって、Linux サーバを遠隔利用するためには SSH に対応したターミナル・ソフトウェアが必要です。SSH ターミナル・ソフトウェアは SSH サーバに接続するためのアプリですので、SSH クライアントとも呼ばれます。

利用形態の違い

デスクトップ・パソコン (PC) の利用



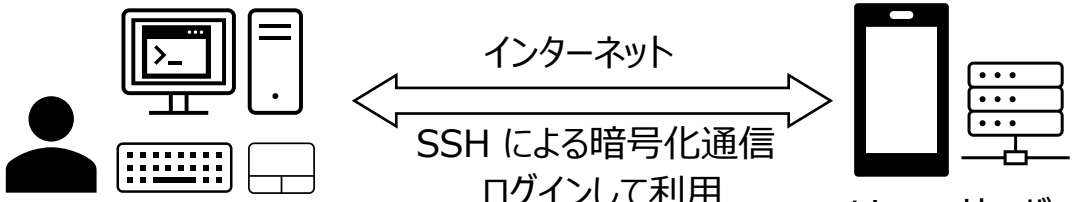
PC の直接利用

操作は直接 PC にマウスやキーボードから行い、実行したソフトウェアはその PC 内で動作する。

GUI: 画面上の絵に対して操作を行う。

遠隔の Linux サーバの利用

ソフトウェアはサーバで動作する。



ローカル PC 端末　　Linux サーバ

CUI: 主に文字による操作と表示

```
$ mkdir directory …
```

CUI のメリット・デメリット

CUI のメリット・デメリットをまとめます。

CUI (Linux) のメリットからまず以下に挙げます。

- **操作が高速** GUI では画面を見ながら正確にマウスなどでポインタを操作してソフトウェアを実行しなければいけませんが、CUI では、全てキーボードからの入力で完結し、極端な場合、画面を見ずに操作することも可能です。
- **操作の記録が容易** CUI での操作は文字によりコマンド入力で行いますので、それをそのままコピペで記録することができますし、自動で実行履歴を記録することもできます。これは研究や解析を行うこと上で非常に重要なことで、CUI では、何をどうしたのか、全てを記録することが可能です。
- **操作方法の伝達が容易** 何か操作方法を教える時に、GUI の場合は相手の画面を見て状況を把握する必要があります。CUI ではコマンドを伝えるだけですし、エラーも文字で表示されるので、状況を伝えることが容易です。
- **操作の再現が容易** 上述の通り、操作を記録することを簡単にできますし、もう一度同じことをしたい場合、それをそのままコマンド入力として貼り付ければ、以前の実行を簡単に再現できます。
- **操作の組み合わせが容易** CUI のソフトウェアは、テキストで入出力をします。つまりあるアプリケーションの出力をそのまま別のアプリケーションの入力として簡単に利用することができます。これは全く独立したアプリケーションを簡単に組み合わせて使用できることを意味します。
- **操作の自動化が容易** 操作の記録と再現が容易であることからもわかるように、自動化も比較的容易にできます。解析対象となるデータ（ファイル）も文字で伝えるため、簡単に入れ替えたり、書き換えたりできますので処理の自動化が容易です。
- **大型計算機の利用が容易** 文字だけで情報をやり取りするため、高性能な遠隔の大型計算機を用いることが容易です。通信環境が遅くても、あるいは、ローカル端末の PC の性能が悪くても、ソフトウェアは遠隔の高性能なサーバ上で動くため、それほど大きな支障はありません。

もちろんデメリットもあります。

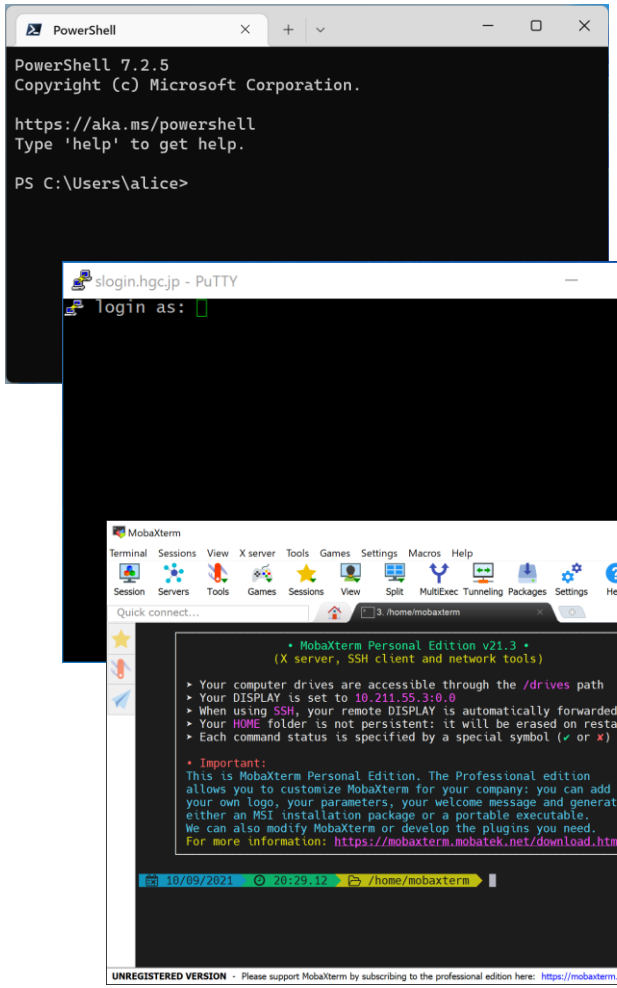
- **直感的に使えない** コマンドの使い方を覚えなければ画面を見ただけでは何ができるのか見当もつきません。
- **正確に入力する必要がある** コマンドやファイル名を1文字ミスタイプするだけで、正しく動かなくなってしまいます。

ただし、これらのデメリットを補うための仕組みもちゃんと用意されています。安心してください。

Windows用 SSH クライアント

無償で利用できる代表的な Windows用 SSH ターミナル・ソフトウェア（SSH クライアント）を紹介します。

- **Windows 10/11 搭載 SSH クライアント**
  - ✓ Windows 10 バージョン 1803 以降には標準で「OpenSSH for Windows」という SSH クライアントが標準搭載されています。
  - ✓ Windows の「ターミナル」アプリ (PowerShell またはコマンドプロンプト) から利用します。通常はこれで十分です。
  - ✓ この実習でもこの SSH クライアントの利用を前提とします。
- **PuTTY（パティ）**
  - ✓ 比較的多くの人に利用されているSSHクライアントです。上述の Open SSH for Windows が標準搭載されるまでの事実上の標準的なソフトウェアでした。
  - ✓ 有志によって日本語化されたものが存在し、接続先の管理などが GUI でできるため、利用しやすいのが特徴です。
  - ✓ 公開鍵・秘密鍵の形式が OpenSSH とは異なるため変換が必要など、やや利用しにくい面もあります。
- **MobaXterm（モバックスターム）**
  - ✓ SFTP も内蔵されているのでサーバとのファイル交換が容易です。
  - ✓ 商用アプリですが機能限定版は無償で利用可能です。
  - ✓ 公式サイト：https://mobaxterm.mobatek.net
  - ✓ 日本語版はありません。
- **その他** Tera Term 等多数



Windows 標準
「Windowsターミナル」
(PowerShell)

PuTTY

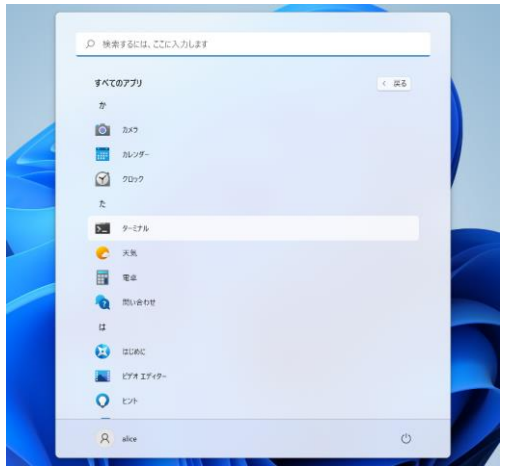
MobaXterm

※ MacOSの場合も選択肢は複数ありますが、基本的に標準搭載の「ターミナル」アプリで十分です。

Windows11標準の「Windows Terminal」

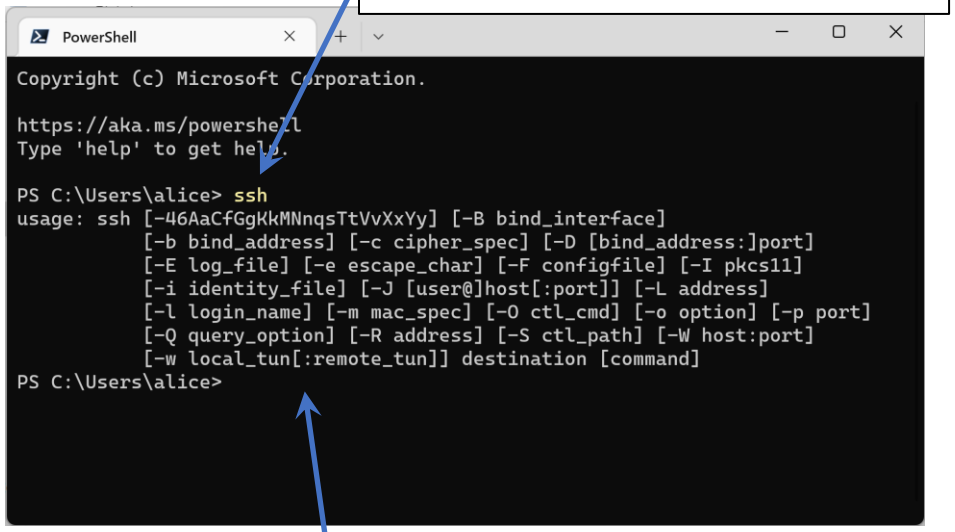
Windows 11 標準の「Windows Terminal（ターミナル）」について解説します。

Windows 11 には標準で「Terminal（ターミナル）」アプリがインストールされています。Windows も Linux のように CUI で操作できる環境が用意されており、「ターミナル」はそのために利用するアプリです。ただし、Windows での CUI には、従来からある「コマンドプロンプト」と比較的新しい「PowerShell」の二つのCUI環境があります。そのどちらでも「ターミナル」から利用可能で、SSH も利用可能です。また従来からあるコマンドプロンプトを直接実行し、そこから SSH を利用することも可能です。「ターミナル」は標準的な Ctrl+C/V のキーボード操作でコピー＆ペーストができます。また、複数の画面をタブでまとめられたりと高機能ですので、こちらの利用をお勧めします。



スタートメニューから「すべてのアプリ>」を選び「ターミナル」をクリックします（"Terminal" と出てくる場合もあります）。起動しない場合はインストールされていない可能性がありますので、Microsoft Store アプリから「Windows Terminal」をインストールしてください。

ssh と入力してリターンキーを押します



このような表示が出れば OpenSSH が利用可能な状態になっています。

もし、エラーメッセージが出る場合、SSHが利用可能ではないので、スタートメニューから「設定」→「アプリ」→「オプション機能」→「オプション機能を表示する[機能を表示]」→検索フィールドに「ssh」と入力→「OpenSSHクライアント」が出てくるのでチェックをつけます。

Linux 環境の準備

Linux 環境を自前で準備する場合、その方法について説明します。

本講習会では、既存の Linux サーバを利用して実習などを行います。もし、各自で自由に使える Linux 環境を用意したい場合は、以下のような選択肢があります。このテキストでは詳細は解説しませんが、自身の Linux 環境構築の参考にしてください。

- **Windows Subsystem for Linux の利用**

  Microsoft 社から提供されている Windows のサブシステムとして動作する Linux 環境です。後述する仮想環境などではなく、Windows のシステムの一部（＝サブシステム）として、Linux と共通の OS の機能と、Linux 標準のツール類（後述するシェルや標準的なコマンド一式）を提供します。従って、Windows PC がほぼ Linux と同等の環境になります。

- **Docker Desktop for Windows / Mac**

  仮想環境の1つで、Windows や MacOS 上で動作し、コンテナ化された Linux アプリケーション実行環境を Windows または MacOS 上で動かすことができるものです。Docker 内で動作する Linux 環境は、実際の Linux サーバを利用する際と同様に、ログインして利用することができます。用途に応じた様々な Docker コンテナ・イメージが有志によって用意されています。

- **その他仮想化**

  VMware などの仮想化ソフトウェアは、OS 内に仮想的なハードウェアを再現し、別の OS をその仮想化環境内にインストールができます。

- **Windows PC などへの Linux の直接のインストール**

  Linux を PC などに直接インストールすれば、立派な Linux サーバとなります。1台の PC に Windows に加え Linux をインストールし、OS を切り替えて利用することもできます。これはデュアル・ブートと言ったりします。使用する OS を切り替える際は再起動が必要になりますが、Windows 環境を残したままにすることができます。Linux 対応が明記されていない、Windows 向けの PC に Linux をインストールすると、ハードウェアの機能が全て使えないなどの不具合が出る場合があります。基本的にこの方法は難度が高いと言えます。

- **MacOS をそのまま使う**

  MacOS は Unix の一種であるため、自分で解析ツールのインストール（コンパイルなど）をすることで、Linux 用のアプリケーションが、大抵の場合そのまま動作します。あとで説明するシェルなども標準で備わっています。

SSH と公開鍵方式

SSH でサーバに接続する際は公開鍵を用いた認証方式を利用することが一般的です。

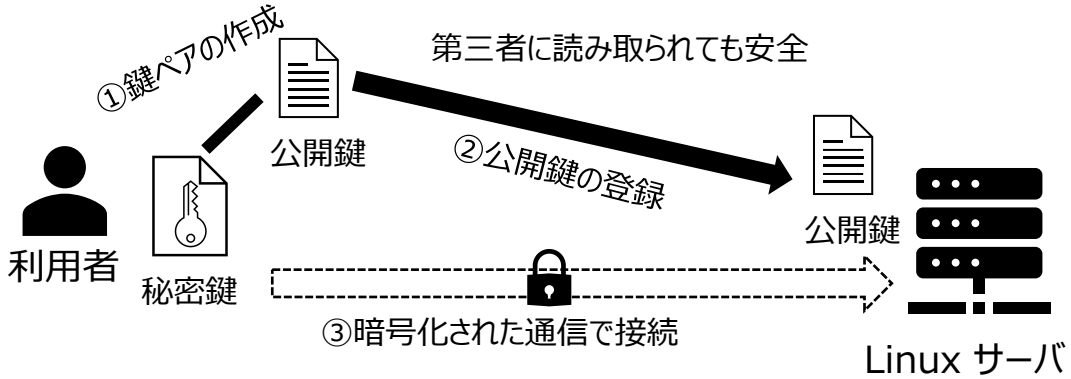
Web 上の会員制サービスなど、IT 系のサービスを利用する場合、基本的に利用者の認証を ID とパスワードで行います。SSH では、パスワードを用いずに「**公開鍵**」を用いる方式によって、より安全な方法を利用して利用者の認証を行うことができます。SSH 自体は ID/パスワード方式による認証に対応していますが、それを禁止している Linux サーバが多いです。

公開鍵方式では認証に2つの鍵ファイルを用います。1つは公開鍵 (public key)、もう1つは秘密（個人）鍵 (private key) です。この2つは必ずペアで利用します。**サーバに公開鍵を登録しておくと、それに対応する秘密鍵でそのサーバにログインできるようになります**。重要な特徴として、公開鍵から秘密鍵を作ることは事実上不可能であるため、公開鍵自体を公開しても安全上の問題がない点が挙げられます。従って、メールなどで公開鍵を送受信しても問題ありません。一方、秘密鍵は他人に知らせたり渡したりしてはいけません。安全のため、秘密鍵はその鍵の利用者だけが知る「パスフレーズ」で暗号化しておきます。パスフレーズがわからなければ、秘密鍵を利用することはできません。

一度作成した公開鍵と秘密鍵のペアはさまざまなサーバで使いまわすことができますので、ログインするさまざまなサーバに同じ公開鍵を登録しておけば、1つの秘密鍵でそれらの複数のサーバにログインすることができます。

公開鍵方式の認証では、このような仕組みによりサーバ側への公開鍵の登録に第三者が介在しても、秘密鍵自体は秘密のまま誰にも知らせる必要がありません。従って、初期パスワードなどを連絡したりする必要もなく安全にサーバ利用を許可したい利用者に対して、そのサーバへのログインをできるようにすることができます。

SSH による通信は暗号化されますので、第三者に通信内容を盗聴される恐れもありません。



秘密鍵を持っていると、対応する公開鍵が登録されているサーバにログイン可能

この章でのコマンド記法について（1）

この章でのコマンドの説明などの際に用いる記法について説明します

特に断りのない場合、コマンドの使い方などの説明の際には以下の様な記法を用います。

```
$ command [-o output ] input ↩   # コメント
Output of command 1
Output of command 2 ...
```

- 先頭の $ はシェル・プロンプトです。ユーザは入力しません。実際には $ より左側には、ユーザやカレントディレクトリに応じて表示が変わりますのが、省略してあります。
- command はユーザが実際に入力するコマンドです。
- [ ] で囲まれた部分はオプション（選択肢・省略も可能）を表します。
- イタリック表記（この例では *input* ）は、ファイル名など、ユーザの実行方法に応じて変更して入力する部分を表します。
- 改行マーク（ ↩ ）でユーザ入力部分の終わりを表します。コマンドを実行するにはリターン（エンター）キーを入力してください。
- # とそれ以降はコメント（注釈）を表します。入力しません。
- プロンプト行以降（改行マークの次の行以降）はコマンドの出力例です。

- 一行で表記が収まらない場合は、下のように折り返して続きを二行目以降に少し字下げして記載しますが、実際に入力する際はリターン・キーなどで改行せず、一行として入力・実行してください。また、カーソルを「_」で示す場合があります。カーソル表示はSSHクライアント毎に異なります。カーソルについては、後で説明します。

```
$ command argument_1 argument_2 argument_3
   argument_4 argument5 _
```

コマンド説明のページでは以下のような記法を用います。

```
command [options] input …
```

- [options] は省略可能なオプションを複数指定できることを意味しています。
- … はファイルなどを複数指定できることを意味しています。

説明中でのユーザ名について

- 以降の説明では、仮のユーザ名として「alice」を用います。alice と表示されている部分は各ユーザ自身のユーザ名に置き換えてください。

この章でのコマンド記法について（2）

この章でのコマンドの説明などの際に用いる記法について説明します（前ページからの続き）

- ASCII （アスキー）文字について

  コンピュータで伝統的に昔から利用できる文字に「ASCII（アスキー）文字」という文字セット（128 文字）があり、Linux でも ASCII 文字の使用が基本になっています。具体的には ASCII 文字は、（半角）英数字 0～9, A～Z, a～z といくつかの記号（!?*など）からなります。現代の Linux では日本語などの文字もほとんどの場合問題なく扱えますが、基本的に ASCII 文字以外は使わないようにしたほうが良いでしょう。

- ファイル名について

  日本語のファイル名も問題なく扱えますが、CUI では多少面倒な点がありますので、原則、ASCII 文字のうち英数字のみにしたほうが無難です。スペースもファイル名に含められますが、コマンド入力時の区切り文字として使用しますので混乱のもとです。避けれるなら避けてください。スペースが使いたいところには `_`（アンダーバー）を使うのが良いでしょう。またそれ以外の記号もコマンド入力時に特別な意味を与えられている場合が多いので、極力使わない方が良いです。英数字以外の記号は `_`（アンダーバー）、`-`（マイナス）、`.`（ピリオド）程度にとどめましょう。

- バックスラッシュについて

  Linux ではバックスラッシュ記号（ \ ）を使いますが、日本語版の Windows では同じ文字コードに `¥`（円マーク）が当てられており、バックスラッシュを入力すると `¥` と表示されますが意味は一緒です。適宜置き換えてください。

  CUI (Linux) では記号を多用しますので読み方を覚えましょう。

記号	読み方	記号	読み方
!	エクスクラメーション マーク	{ }	ブレース
"	（ダブル）クォーテーション（（二重）引用符）	[ ]	ブラケット
#	ナンバー、パウンド、シャープ、ハッシュ	~	チルダ
$	ダラー、ドル、ドル記号	^	キャレット
_	アンダーバー、アンダースコア	.	ドット、ピリオド
`	バッククォート、アキュート・アクセント、リバースクォート	:	コロン
'	（シングル）クォート（クォーテーション）	;	セミコロン
&	アンド、アンパサンド	/	スラッシュ
*	アスタリスク	@	アットマーク
|	バーティカルバー、パイプ	%	パーセント

SSH公開鍵の準備

SHIROKANEにログインするための SSH 公開鍵を作成する方法を解説します。

前述の通り Windows 10/11 搭載 SSH クライアントで SSH 公開鍵・秘密鍵のペアを作成できます。まずはWindowsターミナルを開いてください。MacOS の場合も同様に「ターミナル」アプリを起動します。

ssh-keygen コマンドで SSH の公開鍵・秘密鍵のペアを作成できます。

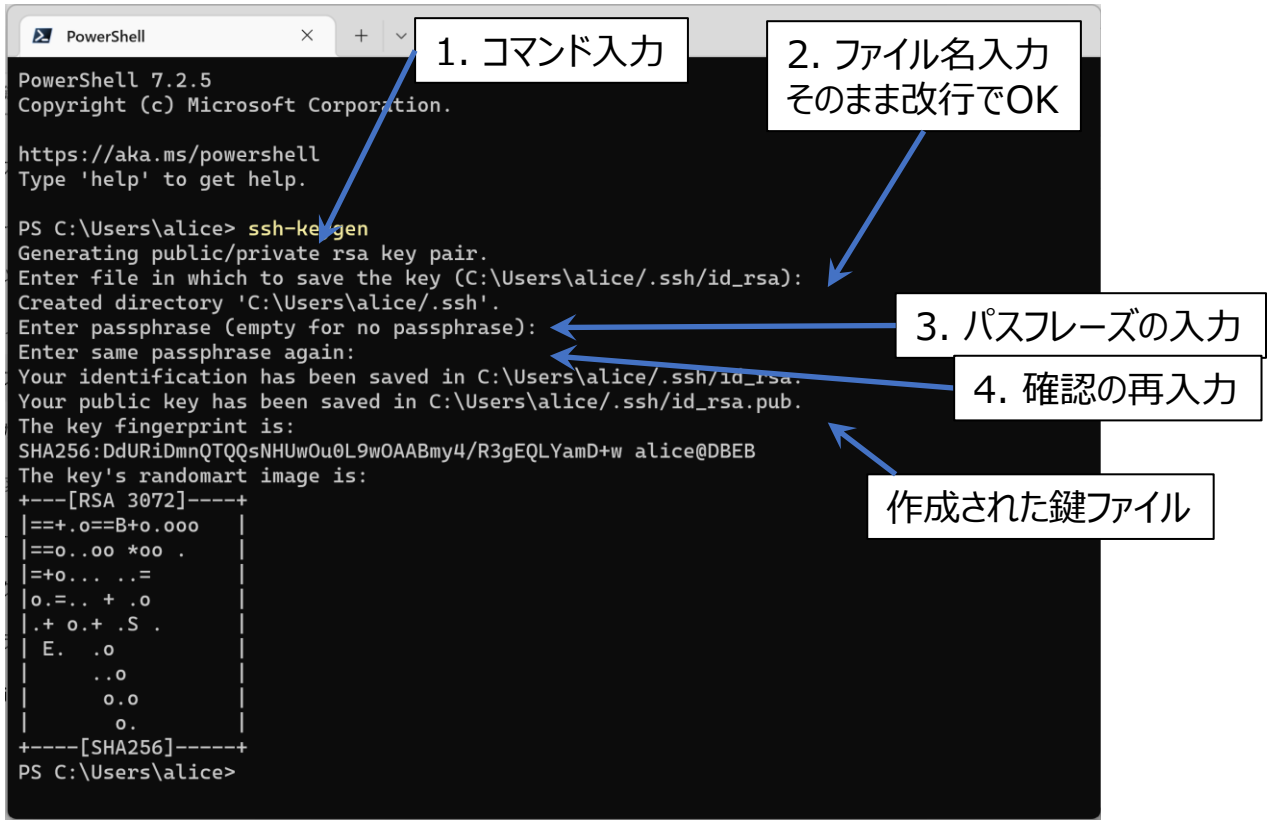
```
C:\Users\alice>ssh-keygen ↩
```

最初にファイル名を聞かれますがデフォルト (id_rsa) のままで良いです。

次にパスフレーズを2回入力します。安全で長めのパスフレーズを設定しましょう。

「C:/Users/（ユーザ名）/.ssh/」以下に id_rsa と id_rsa.pub の2つのファイルが作られます。前者が秘密鍵、後者が公開鍵です。Linux サーバに、後者のファイルを登録します。登録方法は、利用するサーバによって異なります。登録用のウェブページが用意されていたり、管理者にメールで送付する場合もあります。

公開鍵の準備方法は MacOS でもほぼ同様です。ターミナルで `ssh-keygen` コマンドを使用してください。鍵ファイルが「/Users/(ユーザ名)/.ssh/」以下に作成されます。



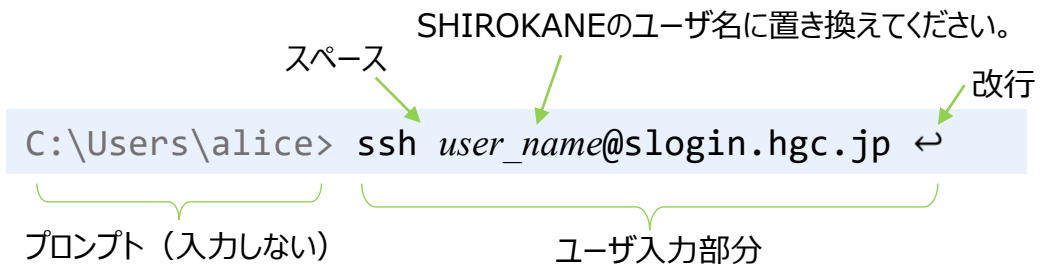
ログインと CUI の基本

まず Linux サーバにログインしてみましょう。

「**ログイン（log-in）**」とは、特定の「**ユーザ**」（詳細は後述）として Linux を利用可能な状態にすることです。SSH によるログイン方法は SSH クライアントごとに異なりますが、ログイン後の操作方法はほぼ同じです。このテキストでは Windows11 標準搭載の「ターミナル」アプリと SSH クライアント (OpenSSH for Windows) を利用して説明します。「ターミナル」は各自で利用する SSH クライアントに読み替えてください。 MacOS では、標準は「ターミナル」アプリになります。まずはあらかじめ稼働している Linux サーバ (SHIROKANE ログイン・ノード) にログインしましょう。順を追って説明します。

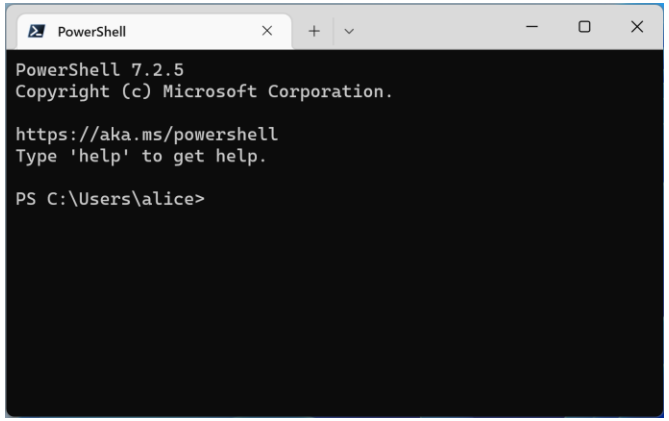
最初に「ターミナル」を起動します。Windows のスタートメニューの「すべてのアプリ」の中に「ターミナル」があります（詳しくは P.7 参照）。Mac の「ターミナル」は「Finder」の「アプリケーション」一覧にある「ユーティリティ」フォルダの中にあります。Windows や MacOS もこれらのソフトウェアを使用すると CUI で操作することも可能です。

次に ssh コマンドを実行して SHIROKANE に接続します。



ログイン方法は SSH クライアントごとに異なりますが、ログイン後の操作方法はほぼ同一です。接続に成功するとパスフレーズを聞いてきますので入力してください。パスフレーズは秘密鍵に自分で設定したものです。認証に成功し、次ページの図のようにシェル・プロンプトが画面に表示されればログイン完了です。この状態で Linux を使った作業ができます。

コマンド（シェル）の画面ではログイン先の**サーバから出力された文字が表示されます**。キーボードから文字を入力すると、即座にサーバに伝えられます。原則的にその時同じ文字がサーバーからユーザのシェルに対して出力されます。これをエコーバックと言います。入力された文字がエコーバックされるため、シェル画面に表示されていることになります（SSH クライアントが直接入力文字を表示しているわけではないということです）。



Windows 11
「ターミナル」アプリ

シェルとカーネル

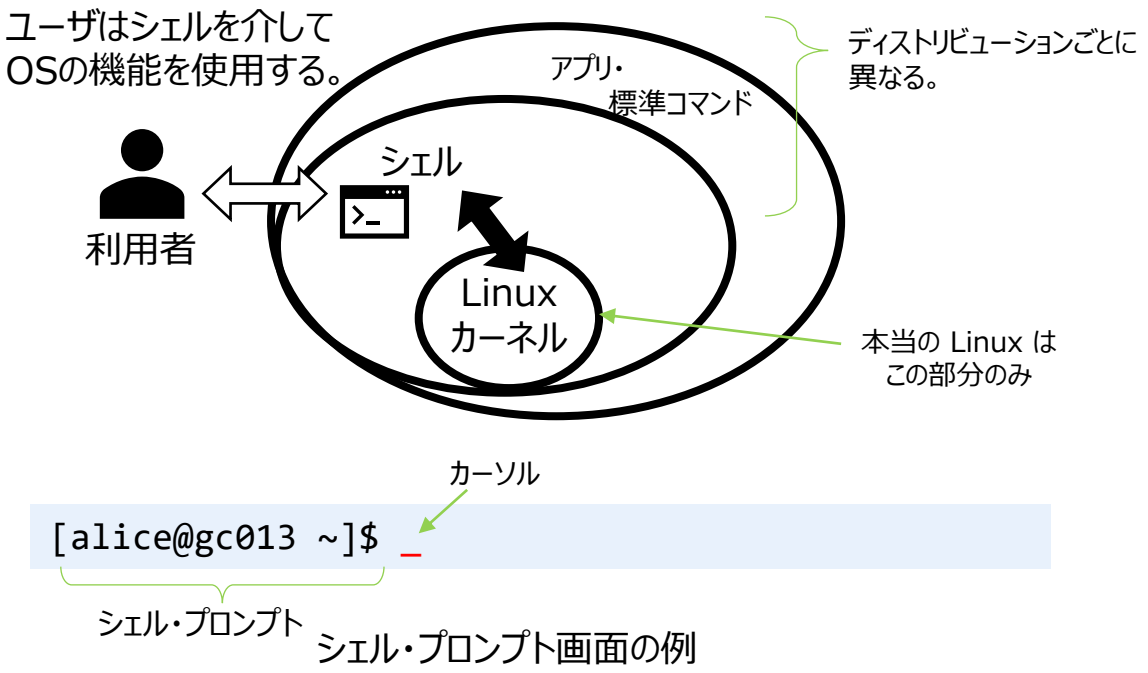
シェルとカーネルについて

Linux の OS としての機能を提供しているのは**カーネル (kernel)** と呼ばれる部分で、コマンド入力など利用者とコンピュータのインターフェイスを提供しているソフトウェアは**シェル (shell)** と呼ばれています。つまり利用者はシェルを介して Linux を操作している、ということになります。シェルは Linux を操作するためのソフトウェアですが、それ自体は OS とは独立したソフトウェアです。

シェル自体にも様々な種類がありますが、SHIROKANE を初め、多くの Linux サーバで標準として採用されているのが Bash (Bourne Again Shell、バッシュ) と呼ばれるシェルです。このテキストでも Bash を利用することを前提としています。他にも zsh （ジィーシェル）や tcsh (ティーシーシェル) などがあります。

古い Unix では Bourne shell (ボーンシェル) と呼ばれるシェルが使われていました。今でも単に「シェル」と言った場合、Bourne shell またはそれと互換性のあるシェルを指します。Bash は Bourne shell の改良版として開発され、Bourne shell とも互換性があります。Linux で Bourne shell を使う場合は、Bourne shell の代わりに bash が起動します。

シェルのプロンプト（コマンド入力）画面はユーザからのコマンドの入力を受け取り、Linux 側に伝える役目をします。**プロンプトが表示されているときはシェルがユーザ入力を待っている状態です。**文字が入力されリターンキーが押されると、入力されたコマンド（＝操作指示）をシェルが解釈し、Linux 側に伝えます。つまり、ファイルを読み取ったり、プログラムを実行したり、停止したりさせます。CUI での操作は、その大部分がコマンド入力による操作ということになります。



シェル・プロンプト画面の例

ユーザとグループ

Linuxには「ユーザ」と「グループ」という概念があります。

Linux には一人の「**ユーザ**」（アカウント）としてログインし利用することになります。Linux ではユーザごとに、ファイルへのアクセスやソフトウェアの実行などが管理されます。つまり Linux 上ではファイルや動いているプログラムは全て特定のユーザと紐づいています。自分以外のユーザのプログラムを勝手に止めたりすることはできませんし、自分以外のユーザが動かしているプログラムがメモリに保持しているデータを覗き見ることもできません。

ユーザはユーザ名とユーザ ID を持ちます。ユーザ名はユーザを表す一意の文字列で、32 文字の小文字アルファベットと先頭文字以外は数字・アンダーバー・ハイフンという制限があります。古い Unix システムではユーザ名に 8 文字までという制限があるため、現代的なシステムでも 8 文字で制限している場合もあります。ユーザ ID (UID) はユーザ毎に割り振られる一意な数値で、一般ユーザは100以降の値が割り当てられます。UID 0 は OS の設定変更などの全権限を持つスーパーユーザ（ルート）用に予約されています。ユーザ名はログイン時に必要ですし、実行しているプログラムやファイルの管理などで常に意識する必要がありますが、ユーザ ID 自体はほとんど意識する必要はありません。ユーザにはそれ以外にもユーザ毎の情報が付随します。例えば、あとで説明する「ホーム・ディレクトリ」や「シェル」などの情報です。

ユーザは1つ以上の「**グループ**」に所属します。グループ単位で Linux 上の様々な利用権限を設定できますが、主にファイルの共有単位として用いられます（詳細は次スライド）。グループにもグループ名とグループ ID 番号 (GID) があります。

（自分のアカウントのユーザ・グループに関する情報は `id` コマンドで表示することができます。コマンドの実行の仕方は後で説明します。）

ユーザ（＝アカウント）は Linux 利用の基本単位

ユーザ

・ユーザ名 (alice)
・ユーザID (101)
・パスワード
・ホーム・ディレクトリ (/home/alice)
・シェル (/bin/bash)

1人のユーザは、1つ以上のグループに所属する。

グループ

・グループ名 (users)
・グループID (1001)
・パスワード

ファイルとディレクトリ

ファイルとディレクトリについて説明します。

Windows 11 や MacOS と同様に Linux でもプログラムやデータを「**ファイル (file)**」という単位で扱います。ファイルはデータの塊（バイト単位のデータ列）です。ファイルには中身であるデータの他に、ファイル名、ファイル所有者（ファイル・オーナー）、グループ、アクセス権などの情報を持ちます。

ファイルは階層構造を持つ「**ディレクトリ**」に保存されます（次スライド参照）。1つのディレクトリの中に複数のファイルまたはディレクトリを保存できます。Windows では「ドライブ」という概念があり、ドライブごとに別の階層構造を持ちます。ハードディスクやUSB メモリなどが1つのドライブ（＝階層構造）に該当します。一般的な Windows では OS がインストールされているドライブを C ドライブと呼び、USB メモリを PC に挿せば別のドライブ（E ドライブなど）が割り当てられ、その中がディレクトリ（フォルダ）の階層構造に分かれます。

一方 Linux （や MacOS）では、階層構造は1つのみで、全てのハードディスクや USB メモリなどの外部記憶装置もこの同一のディレクトリ階層構造の一部として表現されます。階層構造の一番上のディレクトリを「**ルート・ディレクトリ**」(root directory) と言います。外部記憶装置を繋ぐと、この階層構造の特定の場所が割り当てられます。外部記録装置を繋ぐことを「**マウント**」、そして、その場所を「**マウント・ポイント**」と言ったりします。したがって、Windows のような「ドライブ」といった概念はありません。

ファイルの場所を示したものをパス (path) と言ったりします。ルートディレクトリを `/`（スラッシュ記号）で表現し、ディレクトリの階層構造の深さごとにディレクトリ名を `/` で繋げ、最後にファイル名で終わるものが、そのファイルのパスです。またディレクトリもファイルと同様に扱われますので、ディレクトリ名で終わるパスもあり得ます。

パスの表現方法、つまり特定のファイルの位置を示す方法には「**絶対パス**」と「**相対パス**」の二通りの方法があります。パスが `/` で始まっているものを「絶対パス」と言います。一方で `/` で始まっていないものは相対パスで、現在のディレクトリからの相対位置を表します。「現在のディレクトリ」については後で説明します。

特殊なパス表現の1つに `..`（ピリオド2つ）があります。これは現在のディレクトリの親ディレクトリ（1つ上の階層のディレクトリ）を指します。例えば「`../dir/`」は現在の1つ上にある `dir` というディレクトリを示します。似たようなパスに `.`（ピリオド1つ）があります。これは現在のディレクトリを示します。ファイルのコピーなどのコピー元あるいはコピー先が現在のディレクトリの場合にそれを示すのによく使われます。

ホーム・ディレクトリ

ホーム・ディレクトリについて

ユーザにはユーザごとに「**ホーム・ディレクトリ (home directory)** 」が割り当てられます。自分のホームディレクトリにはそのユーザが自由に読み書きできます。Linux サーバは1台を複数のユーザで利用することが一般的であるため、ユーザごとに別の作業用ディレクトリが用意されています。それがホーム・ディレクトリです。

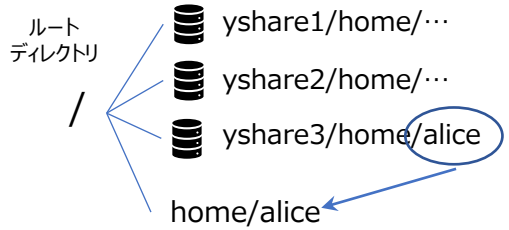
ホーム・ディレクトリにはユーザごとの設定ファイルなども置かれます。ユーザのホーム・ディレクトリのパスは伝統的に「 /home/ユーザ名 」となっています。各ユーザのホーム・ディレクトリを示す特殊なパス表現があります。「~ユーザ名」です。これでそのユーザのホームディレクトリを表します。また ~ だけだと、自分のホーム・ディレクトリを示します。

共同利用のサーバでは、ユーザごとにホーム・ディレクトリの使用容量に制限がかかっている場合がほとんどです。これをディスク・クォータ (disk quota) と言います。HDD容量上限までホーム・ディレクトリを使用できるわけではありませんので注意して利用しましょう。特にゲノム解析では日常的に大きなファイルを扱いますので、気をつけていないと簡単に制限に引っかかってしまいます。そうすると、ソフトウェアはディスクに書き込みができず、実行が失敗してしまいます。自分が使用しているディスクの容量やディスク・クォータの値を確認する方法はシステムごとに異なりますので、利用しているシステムのマニュアルなどを参照ください。

大規模なシステムでは、複数の外部記録（ディスク）装置がファイル・システムの階層構造上にマウントされています。その場合、ユーザやグループごとに異なるディスクが割り当てられることもあります。システムにディスクの障害が発生した時には自分のホーム・ディレクトリがどのディスクに割り当てられているかによって利用ができなくなったりしますので、自分のホーム・ディレクトリのディスクとマウント先を把握しておくと良いでしょう。

Linux のファイル・システムでは、階層構造上のある地点から別の地点に「**リンク (link)**」を作ることができます。システムに接続しているディスク装置は、ディレクトリ階層構造上にある場所にマウントされますが、このリンクによって、例えばホーム・ディレクトリなら実態がどこにマウントされていようと「/home/ユーザ名」でユーザにアクセスしてもらうことが可能になります。

例。SHIROKANE のホーム・ディレクトリは 3 台の HDD に分かれ、それぞれ /yshare1, /yshare2, /yshare3 にマウントされています。



/home/alice は /yshare3/home/alice に「リンク」されている。（リンクが貼られている、ともいう）

ファイル、ディレクトリとパスの関係性

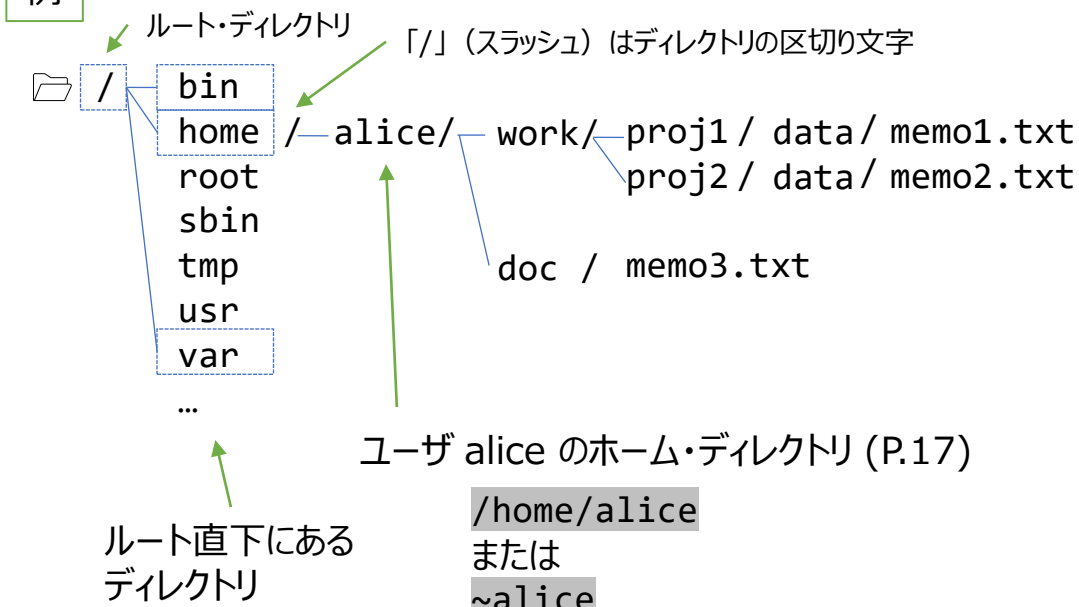
ファイル：データやプログラムの保存単位

ディレクトリ：ファイルをまとめられる入れ物（これもファイルの一種）

ファイルやディレクトリが保存されるファイル・システムは
ルート・ディレクトリからの階層構造になっている

ルート (root) は「根」の意味
ディレクトリ・ファイルの階層構造は木構造になるので、その根っこにあるのがルート・ディレクトリです。

例



パス＝階層構造の中のファイルの位置（を示す文字列）

絶対パス：ルート・ディレクトリから始まるファイルの絶対的な位置

例

```
/home/alice/work/proj1/data/memo1.txt
/home/alice/work/proj2/data/memo2.txt
/home/alice/doc/memo3.txt
```

相対パス：現在のディレクトリからの相対的な位置

(working directory: 作業中のディレクトリ)

現在のディレクトリを `/home/alice/work/proj1` とします。

`memo1.txt` の相対パスは `data/memo1.txt` です。
`memo2.txt` の相対パスは `../proj2/data/memo2.txt` です。
`memo3.txt` の相対パスは `../../doc/memo3.txt` です。

`..` は親ディレクトリ

`~/work/proj2` はどのディレクトリでしょう？

シェルの基本操作（1）とコマンドの文法

シェル（コマンド入力画面）の基本操作について説明します。

ログイン直後のシェル・プロンプトの表示は下図の様になっています。

ログイン直後のシェル・プロンプト

```
[alice@gc016 ~]$ _
```

点滅している「_」はカーソルを表す。

この画面は、コンピュータへの指示を文字によって伝えるための画面です。すでにログインしていますので、ここでの入力は SHIROKANE（のログイン・ノード）での操作になります。CUI でのコンピュータへの指示は「コマンド」とそれらの「引数」や「オプション」をタイプして、リターンキーを最後に入力することで操作（指示）を決定します。リターンキーを入力するまでは入力は実行されません。

前述の通りシェル・プロンプトはシェルがユーザの入力を待っている状態であることを示しています。点滅している「_」はカーソル位置を表します（Windows 10 の場合）。この状態の時、キーボードから文字を入力するとカーソル位置にその文字が入力されます。文字を入力するとカーソルが右に移動していきます。シェル・プロンプトでは**マウスやトラックパッドなどはカーソル移動に使えません**。誤入力などの際にカーソル位置を変えたい時は、カーソル・キーの左右（← →）でカーソルを移動させます。バックスペース・キーで、カーソルの直前の文字が削除され、カーソルが左に移動します。デリート・キーでカーソル位置にある文字を削除し、それ以降の文字を左に詰めます。

カーソル・キーの上下で実行履歴を辿ることができます。上キーを押すと、1回前に実行したコマンドが表示されます。上キーを押すたびに古い履歴が表示されます。下キーでより最近使ったコマンドの履歴表示に戻ります。履歴を表示した状態では、それを通常のコマンド入力のように編集できます。

上下キーで履歴を移動しますが、編集した履歴はそのまま残っているので、履歴を編集中に誤って上下を押してしまっても、慌てずに逆方向のカーソル・キーを押して編集中の履歴あるいは入力中のコマンドに戻りましょう。カーソルがどの位置でもリターン・キーを押せば、表示されているコマンド入力が実行されます。

コマンド入力では**スペース（空白文字）が特殊な意味を持ちます**。スペースで区切った最初の部分がコマンド名として認識され、それ以降は、そのコマンドへオプションあるいは引数（パラメータ）として実行時に渡されます。複数の引数を指定する場合は、スペースで区切ります。

```
[alice@gc016 ~]$ cat foo  bar
```

コマンド名　スペース　複数のスペース

この例では `cat` コマンドに `foo` と `bar` の2つの引数を指定して実行する、という意味になります。スペースが複数連続していても1つの区切りとして認識されます。

シェルで実行されるコマンドとは基本的にソフトウェア（プログラム）そのものです。つまり実行可能なプログラムのファイルそのものです。特殊なコマンドとして、シェルの内部コマンドがあります。それらはファイルとしてではなくシェル自体にその機能が実装されていますので、コマンド名の独立したプログラム・ファイルが実行されるわけではありません。よく使われる基本的なコマンドは内部コマンドとして実装されています。

シェルでのコマンド入力編集機能

覚えると非常に効率よくコマンド入力ができる様になります。ぜひ覚えましょう。

シェルのコマンド入力画面での操作方法の一覧です。カーソル・キー以外にもコマンド入力がしやすいように、デフォルトの（初期）状態でいくつか特殊な編集機能が使えます。表中の「Ctrl + a」は、Control キーを押しながら a を押す、という意味です。マスターするとホーム・ポジションから手を動かさずコマンド入力編集ができ、非常に高速に操作ができるようになります。

キー入力	意味
左右カーソル・キー（← →）	カーソルの左右移動
上カーソル・キー（↑）	履歴を戻る
下カーソル・キー（↓）	戻った履歴を新しい方に進む
Ctrl + a	プロンプト直後、つまり入力中のコマンド文字列の先頭（行頭）にカーソルを移動します。
Ctrl + e	入力中のコマンド文字列の末尾に移動します。
Ctrl + p	カーソルキーの↑キーと同様です。履歴を戻ります。
Ctrl + n	カーソルキーの↓キーと同様です。戻った履歴を新し方に移動します。
Ctrl + f	カーソルキーの→キーと同様です。
Ctrl + b	カーソルキーの←キーと同様です。

キー入力	意味
Ctrl + r	履歴を検索します。続いて検索したい文字を入力します。検索文字入力中に同じ入力をすると、次の履歴を検索します。
Ctrl + g	検索を停止し、元に戻ります。
Ctrl + k	カーソル位置よりも右側をカットします。
Ctrl + y	カーソル位置にカットした文字列をペーストします。
Alt+> (Alt + shift + .)	最新の履歴に戻ります。

コマンド実行履歴について

Ctrl + r でコマンドの実行履歴を検索することができます。検索中にカーソル移動させると、そのまま表示されている履歴を編集可能です。

コマンドの実行履歴はログアウトするときに `~/.bash_history` というファイルに 500 件分、保存されます。したがって、次回ログインしたときに、ログアウト前に実行した履歴がそのまま残っています。ただし、複数の端末からログインしていると、最後にログアウトしたシェルがこのファイルを上書きしてしまうので、注意してください。

`history` コマンドで、履歴を表示できます。`!` に番号をつけると、その番号の履歴を実行できます。また `!!` で直前のコマンドを再実行します。

シェルの基本操作（2）

シェル（コマンド入力画面）の基本操作について説明します。

シェル・プロンプトには「**カレント・ディレクトリ (current directory)**」（現在のディレクトリ）という概念があります。相対パスでファイルを指定するとカレントディレクトリからの相対位置ということになります。多くのコマンドで、明示しない限りカレント・ディレクトリにあるファイルが操作の対象になりますし、ファイルが出力される場合もカレント・ディレクトリにファイルが作られます。

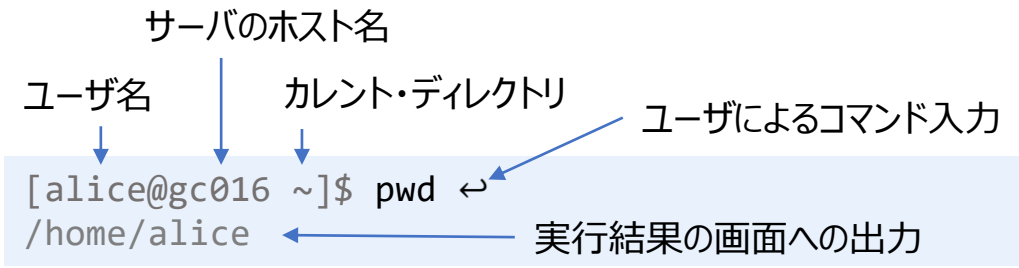
ログイン直後のカレント・ディレクトリはそのユーザのホーム・ディレクトリです。一般的には `/home/ユーザ名` です。カレント・ディレクトリを確認するには `pwd` コマンドを使用します。デフォルトの設定ではプロンプトにカレント・ディレクトが表示されています。「`~`」の部分がそうです。前述の通りこれはホーム・ディレクトリの省略形です。

カレント・ディレクトリを移動するには `cd` (change directory) コマンドを使用します。引数として移動したいディレクトリのパスを指定します。

`cd` コマンドは引数なしで実行するとホーム・ディレクトリに移動します。またマイナスだけ指定すると、つまり「`cd -`」と実行すると最後に `cd` した1つ前のディレクトリに移動できます。一時的にディレクトリを移動し、元に戻る場合に便利です。

ディレクトリの作成には `mkdir` コマンドを使用します。

実際にディレクトリを作って、そこに移動してみましょう。



電子メールアドレスは「ユーザ名@ドメイン名（サーバ名）」の形式になっていますが、実はこのプロンプトの表示部分と全く同じ表記方法になっています。これは電子メールシステムそのものが、Unix サーバにおける「特定のサーバ」の「特定のユーザ」へのメッセージ送信機能で実現されていることによります。

- 実習例

```
[alice@gc016 ~]$ ls ↩  # 現在あるディレクトリの確認

[alice@gc016 ~]$ mkdir testdir ↩
[alice@gc016 ~]$ ls ↩
testdir
[alice@gc016 ~]$ cd testdir ↩
[alice@gc016 testdir]$ pwd ↩
/home/alice/testdir
[alice@gc016 testdir]$ _
```

シェルの基本操作（3）

コマンド名補完とファイル名補完について

シェル・プロンプトでコマンド名を入力中に TAB キー （または Ctrl + i ）を押すと**コマンド名を補完する**ことができます。この時、それまで入力したコマンド名の一部からコマンド名を一意に決められる場合はコマンド名が補完され、スペースが入力されます。一方、コマンド名を一意に決められない場合は、beep 音がします。その後、もう一度 TAB キー（または Ctrl + i) を入力すると、補完の候補リストが表示されます。候補リストが1画面に収まらない場合、最終行に「--More--」と表示されるか、あるいは「Display all XXX possibilities? (y or n)」と表示されます（XXX は候補コマンド数）。前者の場合、スペース・キーを押せば、画面に収まらなかった分の候補が順次表示されます。「q」を入力すると元のコマンド入力中の画面に戻ります。後者の場合、「y」を押すと、前者と同様の状態になります。このコマンド名補完機能を使うことで、コマンド名を正確に覚えていなくても、先頭から数文字だけからコマンド名を探すことができます。コマンドを完全に覚える必要がないため、非常に便利です。ぜひ、操作を覚えてください。

コマンド名を入力し終えた後、ファイル名を指定する場合が多いですが、コマンド名以降で TAB キー（または Ctrl + i ）を押すと、今度はファイル名の補完ができます。操作方法は先程のコマンド名補完と同様です。CUI だとコマンド名もファイル名も正確に覚えていないと操作ができない、と思われがちですが、この TAB 補完機能を使うことで、コマンド名やファイル名の一覧からコマンド・ファイルを指定することができます。

CUI を極端に怖がる人は、この TAB キーによる補完機能（TAB 補完）を知らない、あるいは使いこなせていない場合が多いようです。TAB 補完が使えないと、ファイル名を指定するときに一字一句間違えずに正確に入力する必要があるため非常に手間がかかるようになってしまいます。TAB 補完をマスターしていれば、そのような必要はありません。また、ファイル名を間違えることが減りますので、基本的には、既存のファイル名を入力する際は、必ず TAB 補完を使うようにしましょう。

補完のたびに beep 音が鳴ってうるさいと感じる場合、音を停止することもできます。その場合、補完が機能することがわかりづらくなりますので、画面の blink に変えている人が多いです。

ログアウト

ログイン後、作業が終了したらサーバからログアウトします。

Linux サーバの利用が終了したら「**ログアウト (log-out)**」します。ログアウトは `exit` コマンドを利用します。また Ctrl + d でもログアウトが可能です。ログアウトはその綴りの通り、ログインの逆の操作になります。

ログアウトすると、SSH の接続が切れ（＝サーバとの接続が切れ）、ログイン前の状態に戻ります。コマンド・プロンプトへの入力は接続していた Linux サーバではなく、手元のローカル端末への入力になります。SSH クライアントによっては、画面が閉じたり、初期画面に戻ったりします。

Unix コマンドの流儀

Unix (Linux) コマンド・ソフトウェアはある一定の流儀の元で作られています。標準で用意されているコマンドだけでなく、Linux とは独立した多くのソフトウェアがその流儀を守っています。そうすることで、操作に統一感が生まれ、非常に使いやすいものになります。

このような流儀の1つに「実行に成功したら何も出力しない」というものがあります。2ページ前の `mkdir` の実行例も、不具合があるとエラーメッセージが出ますが、**実行に成功した場合は何も出力されません**。

ヒント：

同じサーバに対して複数のSSHクライアントから同じユーザで同時にログインすることができます。複数のディレクトリを頻繁に行き来して作業する場合は、ディレクトリごとにログインしておいた方が便利かもしれません。

ファイルの送受信

SSH で接続する Linux サーバへのファイルの送受信方法について説明します。

SSH を経由してファイルを送受信するにはいくつか方法があります。SCP または SFTP という通信規格に対応していれば SSH 接続先のサーバとファイルの送受信ができます。SCP/SFTP に対応した Windows/Mac アプリケーションがあり、それを使うと、GUI による操作でファイルを Linux サーバにドラッグ&ドロップで送受信できます。WinSCP や FileZilla が有名です。また Windows 用 SSH クライアントである MobaXterm には SCP クライアントが内蔵されています。

もう1つの方法として、Windows のコマンド・プロンプトまたは MacOS のターミナルで利用できる CUI の `scp` コマンドを利用する方法です。Linux のシェル・プロンプトとほぼ同等の操作ができます。`scp` コマンドの使い方は `ssh` コマンドとあとで説明する `cp` コマンド（→ P.33）を混ぜたような使い方になります。以下、Windows でのコマンド・プロンプトでの使用例です。

ファイルの送信方法

```
C:\Users\alice> scp file user_name@slogin.hgc.jp:dir ↩
```

file は送信したいファイル、*dir* はそのファイルの転送先となる Linux サーバ上のディレクトリです。省略することもでき、そうするとホーム・ディレクトリに送信されます。

※ログインした Linux 上ではなく、手元のローカルPCのターミナル（コマンドプロンプトまたはPowerShell）での操作になりますので注意してください。

ファイルの受信方法

```
C:\Users\alice> scp user_name@slogin.hgc.jp:file dir ↩
```

Linux サーバからローカルに転送するには先ほどの例を逆にすれば OK です。file は Linux 側にあるファイルですが、ディレクトリも含められます。ホームディレクトリからの相対パスになります（→ P.20）。「/」から始めれば絶対パスになります。

ディレクトリを送受信する場合は、`cp` コマンドと同様 `-r` オプションをつけてください。（→ P.33）

例：ディレクトリ *dir* を丸ごと Linux サーバのホームディレクトリに転送する

```
C:\Users\alice> scp -r dir user_name@slogin.hgc.jp: ↩
```

例：Linux サーバ上のホームディレクトリにある *dir* をローカル PC に転送する

```
C:\Users\alice> scp -r user_name@slogin.hgc.jp:dir . ↩
```

ファイル名の拡張子・隠しファイルについて

ファイルの拡張子と隠しファイルについて説明します。

Linux でも Windows/MacOS 同様に、ファイル名に**拡張子**をつける場合があります。拡張子とはファイル名の末尾に . (ドット)に続けて付けるファイルの種類を表す文字列です。例えば、JPEG画像であれば `.jpg` 、テキスト・ファイルであれば `.txt` などがファイル名の最後に付きます。（ただし Windows では設定により初期状態では、拡張子は表示されないようになってます。その代わりファイルを表すアイコンの種類が拡張子に応じて変わります。）

しかし、Linux 自体は、拡張子に応じて処理を変えるような特別な機能や意味は OS 自体にはなく、主にユーザの利便性のためにつけられる場合がほとんどです。そのため、拡張子を必ずつけなければいけないというルールもないですし、そもそも拡張子をつけない場合も多いです。あくまで拡張子は単なるファイル名の一部の文字列です。もちろん拡張子をつける前提のアプリケーションが多いですが、それはあくまでそれぞれのアプリケーションが独自に行っている処理になります。

例えば Windows では `.exe` で実行可能形式（プログラム）ファイルを表しますが、Linux でそれに該当する拡張子はありません。実行可能かどうかは、ファイルに付与された実行権限及びファイルの中身で判断されます。

ファイル名、またはディレクトリ名が `.` （ドット）で始まるものは隠しファイルとして扱われます。あとで説明する ls コマンドでファイル一覧を表示する際に明示的に隠しファイルを表示するオプションを付けないと表示されません。隠しファイル・ディレクトリは主にアプリケーションの設定ファイルに用いられます。例えば Bash のユーザごとの設定ファイルホーム・ディレクトリにある `.bashrc` というファイルになります。これは拡張子が .bashrc というファイル、という意味ではなく、ファイル名がそのまま「.bashrc」というテキスト・ファイルになっています。（テキスト・ファイルについては後述します）

ディレクトリも `.` で始まれば隠しディレクトリになりますので、アプリケーションによってはユーザが直接操作しないファイル群を、専用の隠しディレクトリを用意してそこに自動的に保存する、といった使い方がよくされます。

ファイルのアクセス権について

ファイルにはアクセス権が設定されていて、アクセスできるユーザを制限できます。

Linux 上のファイルやディレクトリには誰がアクセスできるかを設定できるようになっています。つまりユーザが誰でもどのファイルを読んだり書いたりできるわけではありません。これをファイル・ディレクトリの「アクセス権」または「パーミッション (permission) 」と言います。以下特に断りのない場合、ファイルと書かれた部分はディレクトリにも当てはまりますので、ファイルとディレクトリを同一視して説明します。

ファイルには所有者（オーナー）とグループという概念があり、同じ所有者（つまり自分）、同じグループのユーザ、そしてそれ以外（その他）のユーザが、そのファイルに対して何ができるか、がアクセス権ということになります。これをファイルごとに設定することができます。アクセス権には3種類あり、読み込み権限、書き込み（編集）権限、実行権限です。実行権限はディレクトリの場合は、そのディレクトリに入れる権限という意味に変わります。基本的にファイルを作成した人がそのファイルのオーナーになります。またそのユーザの初期所属グループが、そのファイルのグループになります。

例えば、あるファイルがその他のユーザに対して読み込み権（リード権）をつけていれば、誰でもそのファイルを見ることができる、ということになります。またグループで共有したい場合は、そのグループに対してリード権をつければグループメンバー全員がそのファイルを読めるようになります。また編集を許可したい場合は書き込み権もつければそれができるようになります。共同作業しているときはアクセス権をつけ忘れてメンバーに迷惑をかけたりする場合が多いので気をつけましょう。

実行権限は、権限のない人によるプログラム実行の防止や不用意にプログラムが動かないようにするためのもので、Linux ではそのファイルがプログラムであっても、実行権限がついていない場合は、シェル・プロンプトなどから実行することはできません。

ファイルのアクセス権の設定は `ls` コマンドで確認できます（→ P.35）。またアクセス権の設定は `chmod` コマンドで変更できます（→ P.38）。

man コマンド

コマンドのマニュアル（説明）を見るためのコマンドが man コマンドです。

一番最初に覚えるべきコマンドは man (manual) コマンドです。引数としてコマンド名を指定すると、そのコマンドに対する説明（マニュアル manual）を表示することができます。

```
$ man command ↩
```

command に説明を表示したいコマンド名を指定します。

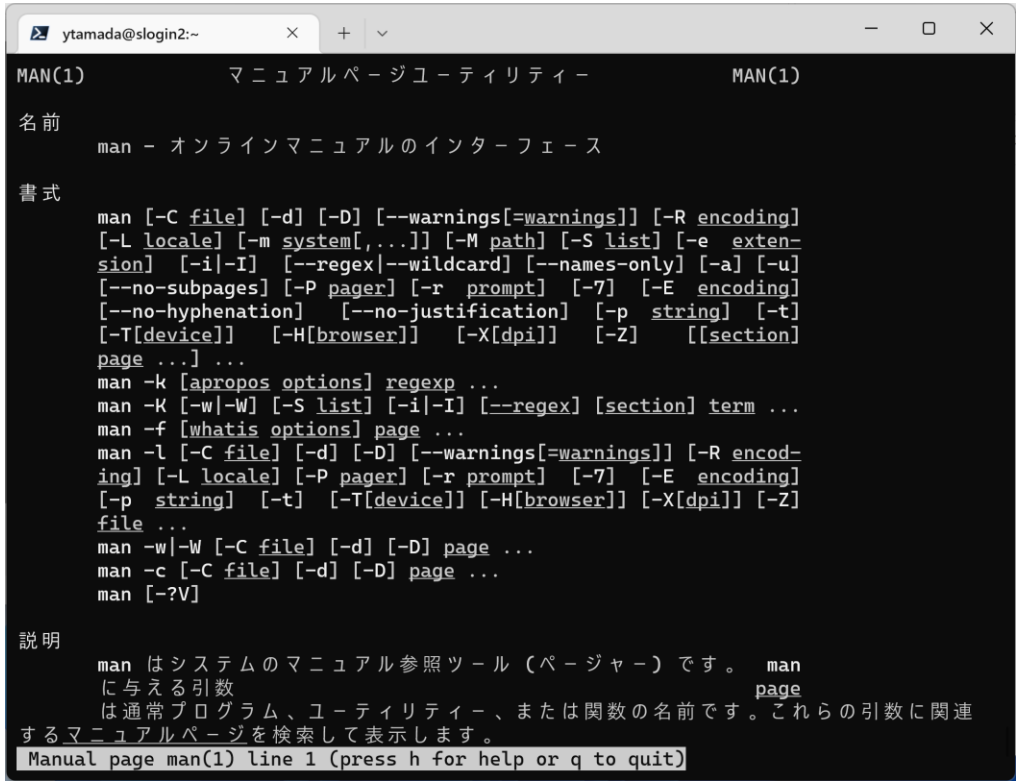
実行すると画面全体がマニュアル表示になります。デフォルトでは less コマンドを使用してマニュアルが表示されますので、操作方法は less コマンドと同様です。詳しい操作方法は後述しますのでそちらをご覧ください。

表示を終了するには q を入力します。スペース・キーで、1画面分スクロールします。カーソルの上下でも画面を移動できます。利用環境によってはマウスのスクロール機能で、表示をスクロールさせることもできます。

当然 man コマンド自体のマニュアルも man コマンドで表示させることができます。

```
$ man man ↩          # man のマニュアルを表示
```

コマンドのオプションがわからなくなった場合、まずは man で確認する習慣をつけましょう。ただし内部コマンドの man を見ようとすると、Bash 自体のマニュアルが表示されますので注意してください。



man コマンドの manual を表示したとき

ファイル操作のための基本コマンド

まず初めにファイル操作の基本コマンドを覚えましょう。

ファイル操作のための基本的なコマンドを以下に列挙します。次のページ以降で、それぞれのコマンドの使い方を詳しく説明します。

ls (list) ファイル・ディレクトリ一覧を表示する。ls コマンドだけで使用すると、複数列でなるべく多くのファイルを一度に表示しようとしますが、`-l` オプションをつけると、1行1ファイルで、ファイルのオーナーやアクセス権などの情報も表示します。

mkdir (make directory) ディレクトリを作成する。

cp (copy) ファイルを複製する。

tree ファイル一覧を木構造で表示する。

chmod (change mode) ファイルの権限を設定する。

mv (move) ファイルを移動する。ファイル名を変える時にも使います。

rm (remove) ファイルを削除する。

rmdir (remove directory) 空のディレクトリを削除する。ディレクトリ内にファイルがあると削除できません。`rm -r` でも削除することができますのであまり使いません。

touch ファイルの編集時刻を更新する。または空ファイルを作成する。

cd (change directory) 現在のディレクトリを変更する。

pwd (print working directory) 現在のディレクトリを表示する。

次ページ以降の説明ではオプションはよく使う代表的なものだけ紹介します。
詳細は `man` コマンドで調べるようにしてください。

cd / pwd コマンド

cd コマンドは現在のディレクトリを変更します。pwd でその確認ができます。

使い方：

```
cd [directory]
cd -
```

```
pwd
```

`cd` コマンドで現在のディレクトリを変更します。現在のディレクトリを変更することをそのディレクトリに「行く」あるいは「移動する」と表現する場合があります。引数を何も指定しなかった場合、現在のディレクトリをホーム・ディレクトリに変更します。引数として `-` を指定すると、最後に `cd` を実行した時に、1つ前の「現在のディレクトリ」に移動します。

`pwd` コマンドで、現在のディレクトリを表示します。

使用例:

```
$ cd ↩
```

ホーム・ディレクトリに戻ります。

```
$ cd dir ↩
```

ディレクトリ `dir` に移動します。

```
$ pwd ↩
/home/alice
```

pwd コマンド使用例。現在のディレクトリが表示されます。

mkdir コマンド

mkdir (make directory) コマンドはディレクトリを作成します。

使い方：

```
mkdir [options] directory […]
```

基本的なオプション：

オプション	解説
-p	ディレクトリが存在しない場合、エラーを出さずに作成する。

現在のディレクトリにディレクトリ *directory* を作成します。作成するディレクトリは相対パスでも指定できます。例えば

```
$ mkdir a/b/c ↩
```

で現在のディレクトリにある `a` というディレクトリの中にある `b` というディレクトリの中に `c` というディレクトリを作るという意味になります。この時、`a` と `a/b` がない場合、エラーになります。`-p` オプションをつけると `a` や `a/b` がない場合に自動的に作成します。

実行例:

```
$ mkdir -p a/b/c ↩
```

現在のディレクトリに `a a/b a/b/c` それぞれのディレクトリを作成します。

```
$ mkdir dir1 dir2 dir3 ↩
```

現在のディレクトリに `dir1 dir2 dir3` それぞれのディレクトリを作成します。

cp コマンド

cp (copy) コマンドはファイルやディレクトリを複製します。

使い方：

```
cp [options] source dest
cp [options] source […] directory
```

基本的なオプション：

オプション	解説
`-i`	上書きする場合に確認する。
`-r`	ディレクトリを再帰的にコピーする。

ファイル・ディレクトリ *source* を *dest* としてコピーします（複製を作ります）。動作は `mv` コマンドとほぼ同一です。複製先としてディレクトリを指定した場合、そのディレクトリ内に作られます。また複数ファイルを指定し最後にディレクトリを指定すると、その複製ディレクトリ内に指定した全てのファイルの複製が作られます。ディレクトリをコピーする場合は `-r` を付けます。

実行例:

```
$ cp a.txt b.txt ↩
```

`a.txt` を `b.txt` としてコピーする。

```
$ cp a.txt dir ↩
```

`a.txt` を `dir` にコピーする（`dir` はディレクトリ）。`dir` の中に `a.txt` の複製が作られる。

```
$ cp a.txt b.txt dir ↩
```

`a.txt` と `b.txt` を `dir` にコピーする（`dir` はディレクトリ）。

```
$ cp -r dir1 dir2 ↩
```

`dir1` の複製を `dir2` に作成する（`dir1`, `dir2` は既存ディレクトリ）。

mv コマンド

mv (move) コマンドはファイルを移動またはファイル名の変更をします。

使い方：

```
mv [options] source target
mv [options] source [...] directory
```

基本的なオプション：

オプション	解説
-i	上書きになる場合、確認する。

ファイル・ディレクトリ *source* を *target* に変更します。*target* にディレクトリを指定すればそのディレクトリに移動します。*target* にディレクトリ名とファイル名を指定すると移動とファイル名の変更を同時にできます。*target* (*directory*) がすでに存在するディレクトリの場合、*source* は *target* (*directory*) に移動しますが、互いがファイルの場合、*target* は上書きされてしまいますので注意してください。

複数ファイルを指定し、最後にディレクトリを指定することで、複数ファイルを同時に移動できます。

実行例:

```
$ mv a.txt b.txt ↩
```

a.txt のファイル名を b.txt に変更する。

```
$ mv a.txt dir ↩
```

a.txt を dir に移動する（ dir はディレクトリ）。

```
$ mv a.txt dir/b.txt ↩
```

a.txt を dir に移動してファイル名を b.txt に変更する。
（ dir はディレクトリ）

```
$ mv a.txt b.txt c.txt dir ↩
```

a.txt , b.txt , c.txt を dir に移動する。
（ dir はディレクトリ）

ls コマンド

ls (list) コマンドはファイルの一覧を表示します。

使い方：

```
ls [options] [file …]
```

基本的なオプション：

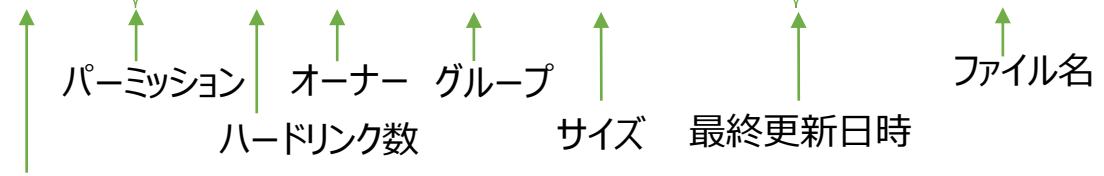
オプション	解説
-l （小文字のL）	詳細な情報を表示する。デフォルトでは画面にできるだけ多くファイル・ディレクトリを表示しますがこのオプションを付けると一行に1ファイル・ディレクトリの表示に変わります。
-F	ファイルの情報を付加する。（ディレクトリの末尾に / が、実行形式の場合 * がつきます）
-a	隠しファイルを表示する。
-d	ディレクトリ自身の情報を表示する。

現在のディレクトリにあるファイル・ディレクトリの一覧を出力します。引数に *file* を指定した場合はそのファイルの情報だけを表示します。ディレクトリを指定するとそのディレクトリにあるファイルが表示されます。ディレクトリ自身の情報を表示したい場合は `-d` オプションを使用します。

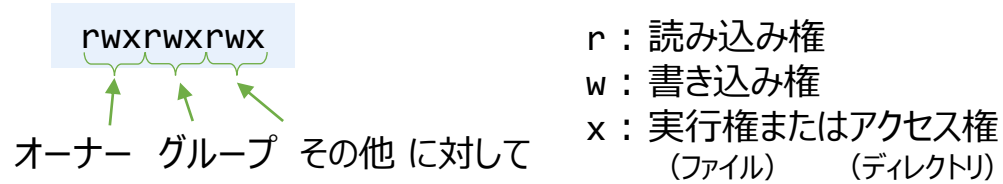
`-l` オプションによる詳細表示では、デフォルトでは以下の情報が出力されます。

`-l` の出力例:

```
-rw-r----- 1 alice group   122 9 月 10 15:00 text.txt
drwxr-x--- 1 alice group  4096 1 月 10  2020 work
```



パーミッション（アクセス権限の意味）の見方：



それぞれを8進数 (0～7) で表現することがある。r=4, w=2, x=1
e.g. 755 は rwxr-xr-x と同じ。640 は rw-r----- と同じ。（P.38）

（ファイル・アクセス権についての詳細は P.28）

rm コマンド

rm (remove) コマンドはファイルを削除します。

使い方：

```
rm [options] file […]
```

基本的なオプション：

オプション	解説
`-r`	ディレクトリとその中身を再帰的に削除する。
`-f`	ファイルが存在しない場合もエラーを表示せずに強制的に削除する。

ファイル・ディレクトリ *file* を削除します。Linux では一旦ファイル・ディレクトリを削除するとゴミ箱のようなところには移動せず、元に戻せなくなりますので注意してください。

`-r` オプションでディレクトリも再帰的に、つまり削除しようとするディレクトリの中にあるディレクトリも全て削除します。

使用例:

```
$ rm a.txt ↩
```

ファイル `a.txt` を削除します。

```
$ rm -r dir ↩
```

ディレクトリ `dir` を再帰的に（中の全てのディレクトリも）削除します。

tree コマンド

tree コマンドはファイルの一覧を木構造で表示します。

使い方：

```
tree [options] [directory …]
```

基本的なオプション：

オプション	解説
-F （小文字のL）	ファイルの情報を付加する。（ディレクトリの末尾に / が、実行形式の場合 * がつきます）
-a	隠しファイルを表示する。
-d	ディレクトリのみ表示する。

指定したディレクトリにあるファイル・ディレクトリの一覧を木構造 (tree) で出力します。ディレクトリ内に含まれるファイル・ディレクトリが全て表示されます。したがって、上位ディレクトリ階層での実行やファイル数が多い場合は、非常に時間がかかり、また表示が多くなるので実行には注意が必要です。

P.18 の /home/alice での実行例：

```
$ tree . ↩
.
├── doc
│   └── memo3.txt
├── proj1
│   └── data
│       └── memo1.txt
└── proj2
    └── data
        └── memo2.txt
5 directories, 3 files
```

chmod コマンド

ファイルのアクセス権を設定します。

使い方：

```
chmod mode,… [file …]
```

file のアクセス権の設定を mode に変更します。mode の指定の仕方は何種類かあります。

1. 8進数の数値で指定

前ページで説明した通り、ユーザ、グループ、その他への権限の設定は 0～7 の数値で表現できます。読み込み権 (r) が 4, 書き込み権 (w) が 2、実行権 (x) が 1 としたとき、それぞれ権限を付与するものを足せば、権限設定の数値になります。*mode* にこれら3桁の数字を指定します。例えば 770 を指定するとユーザ、グループに読み込み、書き込み、実行権限をつけ、その他には一切権限をつけない、という意味になります。`ls -l` コマンドでの表示での rwxrwx--- と同じ意味です。644 であれば rw-r--r-- となります。

2. 変更したい権限を文字で指定

以下のフォーマットで、3文字で変更したい権限を指定します。

```
{ugo}{+-}{rwx}
```

{ } は囲んだものから1つずつ選ぶ、ということを意味しています。1文字目は対象で u がユーザ、g がグループ、o がその他です。2文字目は権限を加えるか、取り除くかを + か - で指定します。最後は、読み込み権、書き込み権、実行権のどれを変更するか、をそれぞれ r, w, x で表します。例えば、グループに読み込み権をつけたい場合、*mode* に g+r という3文字を指定すればOKです。カンマで区切り複数指定できます。

使用例：

```
$ chmod 640 a.txt b.txt ↩
```

a.txt と b.txt 2つのアクセス権を 640 つまり rw-r----- に設定します。

```
$ chmod u+x command ↩
```

command というファイルに実行権を付与します。

rmdir コマンド

rmdir (remove directory) コマンドは空ディレクトリを削除します。

使い方：

```
rmdir [options] directory […]
```

基本的なオプション：

オプション	解説
-p	親ディレクトリも削除する。

現在のディレクトリにある空のディレクトリ *directory* を削除します。*directory* が空でない場合、エラーになります。ディレクトリの中のファイルの有り無しに関わらず削除したい場合は

```
$ rm -r directory ↩
```

でできますので、rmdir コマンド自体の使用機会はあまりないでしょう。

使用例:

```
$ rmdir dir ↩
```

空のディレクトリ dir を削除します。

```
$ rmdir dir1 dir2 dir3 ↩
```

空のディレクトリ dir1 dir2 dir3 を削除します。

touch コマンド

touch コマンドはファイルの更新日時を変更します。

使い方：

```
touch [options] file […]
```

基本的なオプション：

オプション	解説
-c	ファイルを作成しない。

ファイルまたはディレクトリ file のアクセス日時及び更新日時を現在日時に更新します。もし file が存在しない場合、空の（サイズが 0 バイトの）ファイルを作成します。

Linux アプリの中にはファイルの更新日時を確認して、あるファイルが更新されていて、それに依存する別のファイルの生成が必要かどうかを判断するものがあります。そういったアプリに、処理を行わせるために、touch コマンドでそのファイルを編集したことにすることができます。

使用例:

```
$ touch a.txt ↩
```

ファイル a.txt の更新日時に更新します。ファイルが存在しない場合、空の（サイズ 0 バイトの）ファイル a.txt を作成します。

ファイル閲覧・加工系コマンド

ファイルの中身を閲覧したり簡単な加工をするコマンドを紹介します。

ファイル操作の基本を学んだら、次はファイルの閲覧関係のコマンドを紹介します。

cat (concatenate) ファイルの中身を画面（標準出力）に**そのまま**出力するコマンドです。テキスト・ファイル以外で使用すると、画面が崩れるので気をつけましょう。`cat` コマンドはファイル中のテキスト行数が画面に表示できるサイズより多い場合、流れていってしまいます。大抵の端末アプリではスクロールで戻れますが、ファイルサイズが大きい場合は、表示し切るのに時間がかかる場合もありますので、ファイル閲覧には `less` コマンドをお勧めします。

less テキスト・ファイルの中身を端末画面サイズ分表示して、一旦停止し、それ以上はキー操作でスクロールさせることができます。`less` コマンドは非常に頻繁に使用するコマンドの1つです。

余談ですが、`less` コマンドが登場する前まで、`more` コマンドというものがありました（今でも使えます）。これは `less` が上カーソルキーなどで、上に移動できるのに対し、`more` は先に進むだけで戻ることができないコマンドでした。そもそも昔はソフトウェアから端末の画面表示をあまり高度に制御できなかったのです。つまり、基本は表示しっぱなし、でした。

wc (word count) 文字数や行数を数え上げるコマンドです。

head テキスト・ファイルの先頭から指定された行数だけ出力します。

tail `head` とは逆に、末尾から指定行数ファイルの中身を出力します。

cut タブやスペースで区切られたテキスト・ファイルの各行から、特定のカラム（列）だけ抜き出して表示します。

paste 複数のファイルを列方向に繋げます。

sort 行をアルファベット順に並べ替えます。

uniq (unique) 重複行を削除します。

split ファイルの分割を行います。

column タブ区切りなどのテキスト・ファイルの各カラムに、スペースを挿入して位置を合わせます。

join 2つのファイルの行を特定の列をキーとして一致するように繋げます。

cat コマンド

cat (concatenate) コマンドはファイルを標準出力（画面）に出力します。

使い方：

```
cat [ file ] ...
```

cat コマンドは指定されたファイルの中身を標準出力に出力します。標準出力については P.57 を参照ください。ここではシェル画面のことと考えて問題ありません。ファイルが指定されない場合、標準入力からの入力をそのまま標準出力に出力します。前ページにも書いた通り、指定されたファイルがテキスト・ファイルでもバイナリ・ファイルでも関係なく出力します。バイナリ・ファイルを出力した場合、画面表示が異常になる場合があります。

ファイルが複数指定された場合、指定された順番に出力します。このことを用いて、複数のファイルを順番に結合する場合に cat コマンドを使うことができます。（コマンド名の由来 concatenate の通り、本来はそのためのコマンドです。）テキスト・ファイルに限らず、どんなファイルでも結合に使えます。逆に、ファイルの分割は split コマンドで行えます。

ゲノム解析では分割された fastq ファイルの結合などに使います。

結合されたファイルの中身を1つのファイルに保存する場合には、リダイレクト機能 (P.57) を用いて標準出力の出力先をファイルに変更すれば良いです。

使用例:

```
$ cat *.txt > output.txt↩
```

現在のディレクトリにある、拡張子が .txt であるファイル全てを連結し、output.txt として保存します。

less コマンド

ファイルを閲覧するために頻繁に使用するコマンドの1つです。

使い方：

```
less [options] [ file ]
```

基本的なオプション：

オプション	説明
-S	行を折り返さずに表示します。左右カーソルで横スクロールする様になります。
-N	行番号を表示します。

less コマンドはテキスト・ファイルを閲覧するためのコマンドです。*file* を端末画面上に表示できる分だけ表示します。キーボード操作によって表示する部分を前後にスクロールさせることができます。

基本的な操作方法（キー操作の，は、または、の意味）

キー操作	説明
q	終了する。
h	less コマンドの操作方法を表示する。
ENTER, ↓, Ctrl + n	1行分、下にスクロールする。
[SPACE]	1画面分、下にスクロールする。
y, ↑, Ctrl + p	1行分、上にスクロールする。
b, Alt+v	1画面分、上にスクロールする。
p	一番上に戻る。
F	最終画面に移動する。ファイルが更新されているか監視し、されている場合、読み込み最終画面までスクロールする。Ctrl+c で監視を停止する。
/ （スラッシュ）	下方向に検索する。
n	検索時に次を検索する。
N	検索時に1つ前を検索する。

wc コマンド

wc (word count) コマンドは文字数や行数を数えます。

使い方：

```
wc [options] [ file ] …
```

基本的なオプション：

オプション	説明
-l	行数のみを数えます。
-w	単語数のみを数えます。
-c	バイト数のみを数えます。
-m	文字数を数えます。

ファイル *file* の改行（行数）、語数、文字数（バイト数）を数えます。語数とは連続するスペースやタブ記号、改行記号で区切られたそれ以外の文字列の個数です。オプションを何も指定しない場合、それぞれ数えてファイル名とともに表示します。複数のファイルが指定された場合は、合計値も表示します。

ゲノム解析では遺伝子リストなどをテキスト・ファイルで保持することが多いですが、その際、遺伝子数を数えるのに、または確認するのに `wc -l` を頻繁に用います。

実行例:

```
$ wc -l *.txt ↩
```

現在のディレクトリにある拡張子が `.txt` のファイルの行数とその合計値を出力します。

head / tail コマンド

テキスト・ファイルの先頭あるいは末尾の数行を確認するためのコマンドです。

使い方：

```
head [options] [ file ] …
tail [options] [ file ] …
```

基本的なオプション：

オプション	説明
-*n* -n *n*	先頭・末尾から *n* 行を出力します。

head コマンドはファイル *file* の先頭から、tail コマンドは末尾から指定された数行を出力します。デフォルトでは（オプションを指定しない場合）、それぞれの先頭あるいは末尾から 10 行を出力します。

head コマンドと tail コマンドを組み合わせると特定の行番目から数行を表示する、ということも可能ですが、その場合は後で解説する sed コマンドを使った方が効率的です。

使用例:

```
$ head -4 foo.txt ↩
```

foo.txt の先頭から 4 行を表示します。

```
$ head -5 bar.txt | tail -1 ↩
```

bar.txt の 5 行目のみを表示します。（非効率なやり方）

cut コマンド

特定の列（カラム）を抜き出します。

使い方：

```
cut [options] [ file ] …
```

基本的なオプション：

オプション	説明
`-f` *list*	list で指定されたカラム（列）を抜き出します。list の書式は以下の通りです。カンマで区切ると複数のリストを指定できます。 *n* 　*n* カラム目 *-n* 　1 カラム目から *n* カラム目まで。 *n-* 　*n* カラム目から最後まで。 *n-m* 　*n* カラム目から *m* カラム目まで。
`-d` *c*	区切り文字 (デリミタ, delimiter）を指定します。

各行のタブ記号で区切られた列のうち、特定の列を抜き出して表示するコマンドです。区切り文字は `-d` オプションで変更可能です。区切り文字が連続する場合は、間に空の列が存在するとみなされます。従って連続する複数のスペースを1つのフィールドの区切りとして使用しているテキストではうまく動作しません。

使用例:

```
$ cut -f1,3-5 foo.txt ↩
```

テキスト・ファイル `foo.txt` の各行の、1, 3, 4, 5 カラム（列）目 を出力します。

```
$ cut -f6- bar.txt ↩
```

`bar.txt` の各行の 6 カラム目以降を出力します。

```
$ cut -d' ' -f4 buzz.txt ↩
```

スペースで各列が区切られた `buzz.txt` の 4 カラム目を出力します。

sort コマンド

アルファベット順に行を並び替えます（ソート）。

使い方：

```
sort [options] [ file ] …
```

基本的なオプション：

オプション	説明
-r	逆順にソートする。
-n	数値としてソートする。
-k *n*[,*m*]	ソートに用いる列（カラム）を指定する。*n* だけ指定すると同順の際に *n* 以降の列が全て使われる。*n* カラム目だけを使いたい場合は -k *n*,*n* と指定する。
-t c	列の区切り文字を指定する。
-c	ソートされているか確認する
-u	-c と使う場合、厳密に順番を確認する。単独で指定した場合、最初の同一行のみ出力する。

テキスト・ファイルの行をアルファベット順にソート（並び替え）します。数値の大きさでソートする場合はオプションが必要です。デフォルトでは空白文字（スペース・タブ）は全て区切り文字として扱われます。

使用例:

```
$ sort -n -k2,2 foo.txt > bar.txt ↩
```

2カラム目を利用して数値として昇順にソートします。

uniq コマンド

重複行を削除します。

使い方：

```
uniq [options] [[input ] output]
```

基本的なオプション：

オプション	説明
`-d`	重複行のみ出力する。
`-f n`	最初の *n* 列を比較しない。

重複する行を見つけ削除します。つまり出力されるのは行単位で重複の無いテキスト・ファイルです。このコマンドの入力ファイル *input* はあらかじめソートされている必要があります。*output* が指定されていると、結果は標準出力ではなく、*output* に出力されます。

ゲノム解析での想定用途としては、複数の遺伝子リストをマージして、重複のない新しい遺伝子リストを作る、といった作業があげられます。その場合、 `cat` , `sort` , `uniq` コマンドを組み合わせることでこれを実現できます。

使用例:

```
$ cat list_*.txt | sort | uniq > merged.txt ↩
```

list_ で始まる拡張子 .txt のファイル全てを結合し、重複を取り除きます。

paste コマンド

複数のテキスト・ファイルを行ごとに連結します。

使い方：

```
paste [options] [ file ] …
```

基本的なオプション：

オプション	説明
-d *c*	区切り文字として *c* を用います。

複数のファイルの各行を区切り文字で連結して出力します。区切り文字は何も指定しないとタブ記号が利用されます。cat コマンドが行方向にファイルを繋げたのに対し、paste は列方向にファイルを繋げるコマンドになります。

タブ区切りテキスト・ファイルのデータ・ファイルを加工するときに頻繁に利用するコマンドの1つです。

結果は標準出力に出力されますので、保存する場合はファイルにリダイレクトします。cut コマンドとよく組み合わせて利用されます。

使用例:

```
$ cut -f1 foo.txt > f1.txt ↩
$ cut -f5 foo.txt > f5.txt ↩
$ paste -d- f1.txt f5.txt > bar.txt ↩
```

この例では foo.txt の1カラム目と5カラム目を抽出し、それを x-y の形で繋いだものに加工して、bar.txt として保存する、ということをしています。プログラムやスクリプトを書かなくても、こういったテキスト・ファイルの加工はコマンドの組み合わせで簡単にできてしまいます。

split コマンド

ファイルを分割するコマンドです。

使い方：

```
split [options] [[input ] prefix]
```

基本的なオプション：

オプション	説明
-d	接尾辞を数字に変更する。
-l *n*	1ファイルあたりの行数を指定して分割する。
-b *n*	1ファイルあたりのバイト数を指定して分割する。

デフォルトでは *input* を 1000 行ごとに *prefix*aa, *prefix*ab, ... というファイル名のファイルに分割します。*prefix* のデフォルトは x です。従って *prefix* に何も指定しないと xaa, xab, ... というファイル名のファイルに *input* が分割されます。

バイナリ・ファイルの分割も可能です。その場合、`-b` オプションを指定しないと、うまく分割できません。（バイナリ・ファイルには行の概念がないため）

分割されたファイルは `cat` コマンドで結合が可能です。

使用例:

```
$ split -l 10 input.txt output_ ↩
```

input.txt を 10 行ごとに output_aa, output_ab, ... というファイル名のファイルに分割します。

column コマンド

テキスト・ファイルを整形するコマンドです。

使い方：

```
column [options] [file] …
```

基本的なオプション：

オプション	説明
-t	列数を判定し自動的に整形する。
-s *c*	列の区切り文字を指定します。

入力ファイル *file* を整形するコマンドです。何もオプションを指定しないと、行ごとのファイルを列にできるだけ並べて表示します。`-t` オプションをつけると、タブ区切りのデータファイルの列の位置を合わせて出力してくれます。`-s` でカンマを渡せばカンマ区切りファイルも見やすく整形できます。データ・ファイルの確認に less コマンドと組み合わせると便利です。少しわかりにくいですが例をみれば一目瞭然です。例えば a, b, c と1行ずつ書かれた以下のようなファイルは

```
$ cat sample.txt ↩
a
b
c
```

```
$ column sample.txt ↩
a     b     c
```

のように表示されます。カンマ区切りのファイルの場合、

```
$ cat sample.txt ↩
a,bb,c
d,e,fffff
ggg,h,i
$ column -t -s, sample.txt ↩
a    bb  c
d    e   fffff
ggg  h   i
```

このように、列を綺麗に整形して出力してくれます。タブ区切りテキストで8文字以上のデータがあると less などで表示した場合、列の表示位置がずれてしまいますが、同様に `column` コマンドを使えば綺麗に整形できます。`less –s` と組み合わせるのがおすすめです。

join コマンド

2つのファイルを鍵となる列を指定して連結する。

使い方：

```
join [options] file1 file2
```

基本的なオプション：

オプション	説明
`-1` *n*	1つ目のファイルの鍵として使う列の位置
`-2` *n*	2つ目のファイルの鍵として使う列の位置
`-t` *c*	列の区切り文字を指定します。

このコマンドは SQL での join を行います。つまり2つのテキスト・ファイルに対して、それぞれ鍵となる列の値が同じ行を、列方向に連結します。ただし、2つのファイルは鍵として使う列に対してソートされていなければいけません。

遺伝子名を鍵に2つのテキスト・ファイルをマージする、といったことをしたい場合はゲノム解析では頻発しますが、プログラムを書かずこのコマンドだけでそれを実現できます。ただしタブ区切りの場合、`-t` オプションの指定方法に少しだけ工夫が必要です。具体的には `-t $'\t'` のように指定する必要があります。

例：file1.txt 及び file2.txt はタブ区切りのテキスト・ファイルです。それを1カラム目を鍵として列方向にマージします。

```
$ cat file1.txt ↩
gene1    10
gene2    20
gene3    30
$ cat file2.txt ↩
gene1    abc
gene2    def
gene3    ghi
$ join -t $'\t' file1.txt file2.txt ↩
gene1    10      abc
gene2    20      def
gene3    30      ghi
```

ワイルドカードによるファイル指定

ワイルドカード記号を使うことで複数のファイルを一度に指定できます。

コマンド入力時のファイル指定をする際に、ワイルドカード記号を用いることで複数のファイルを同時に指定できます。例えば「 * 」（アスタリスク）は現在のディレクトリにある全てのファイルを指します。つまり

```
$ rm * ↩
```

で、現在のディレクトリにある全てのファイルを削除できます（ただし隠しファイルは含まれない）。 * はファイル名の一部にも使用することができます。

```
$ rm *.txt ↩
```

で、「.txt」という拡張子を持つファイル全てを削除できます。

もう少し正確に表現すると「 * 」記号は現在のディレクトリでファイル名の検索を行い、任意の長さに「マッチ」する（＝条件にあう）ファイルを探して、そのマッチしたファイル名に置き換えるという記号になります。似たようなものに「 ? 」記号があり、これは任意のファイル名の一文字にマッチします。例えば、

```
$ rm abc?.txt ↩
```

は、 abc で始まり、次が任意の1文字、そして拡張子が .txt のファイル名とマッチします。

[*x*-*y*] と指定すると *x* から *y* の範囲にある文字1文字にマッチします。ここで *x* 及び *y* は任意の英数字で、文字の範囲とは、文字に付けられたアスキーコードの順番です。大まかには、英数字では 0 からはじめ 9 まで, 大文字 A から Z、小文字 a から z の順に並んでいると思ってください。

例えば

```
$ rm abc[0-9].txt ↩
```

は、 abc で始まり、そのあと数字が 1 文字ついている拡張子が .txt のファイルを削除します。

ワイルドカードによるファイル名指定（ファイルの探索）は、それぞれのコマンドが行なっているのではなく、シェルがコマンドを実行する前に行なっています。従って、例えば

```
$ echo * ↩
```

のように実行すると、マッチするファイルを表示することができます。 echo コマンドは引数で指定された文字を表示するコマンドです。シェルによって「 * 」の部分がマッチするファイル名に展開され、その後で echo コマンドが実行されますので、マッチしたファイル名が画面に出力されるようになります。

コマンドのオプションについて

多くのコマンドでオプション指定方法は統一的なルールに従っています。

多くのコマンドで動作の設定を変えるオプションが付けられます。オプションは通常は `-` （ハイフンマイナス、通称「マイナス」）で始まり、続いて1文字のアルファベットで指定します。例えば、ls コマンドで隠しファイルを表示するオプション `-a` で、`ls -a` というように実行します。

オプションを複数指定する場合、`-a -l` のように複数指定できますが、この 1文字のオプションは「短いオプション」とも呼ばれ、Linux の一般的なコマンドでは `-al` のように1つのマイナスに続けて同時に複数指定できます。ただし、これは全てのコマンドで使えるわけではないので注意してください。

多くのオプションは上述の「短いオプション」に対応する「長いオプション」による指定方法があります。逆にいうと一文字で指定するオプションは長い指定方法の短縮形という扱いになっています。長い指定方法は `--` （ハイフンマイナス2個）で始めます。例えば `ls` コマンドの `-a` オプションは `--all` が長い指定方法です。長い指定方法の場合は、複数のオプションを一回で指定する方法はありません。例えば `ls` コマンドに `-r` オプションがあります。この長い指定方法は `--reverse` です。短いオプションの場合には `-a` と同時に指定したい場合、`-ar` と指定できますが `--allreverse` のように指定はできません。長いオプション指定で複数指定する場合は、スペースで区切り複数記入します。例えば `-ar` は `--all --reverse` になります。

多くのコマンドはファイル名を引数として受け取りますが、この方法ですと、ハイフンマイナスで始まるファイルを指定することができません。そのため、これ以降は全てファイル名として扱う、というオプションが用意されています。「`--`」（ハイフンマイナス2個）です。例えば `-a` というファイルを `ls` で指定したい場合、

```
$ ls -- -a ↩
```

と指定します。

コピペの際には - の違いに気をつけて！

「- キー」（マイナスキー）を叩いて入力される文字は正式には「ハイフンマイナス」と呼ばれる文字になります。ワードやパワーポイントなどでハイフンマイナスに続いて数字を入力すると「マイナス」記号に自動で変換されることがあります。「マイナス」と「ハイフンマイナス」は違う文字ですので、「マイナス」を Linux のシェルに入力しても「ハイフンマイナス」として扱われません。つまりオプションを指定する記号として認識されないので注意してください。

- `-` ハイフンマイナス
- `−` マイナス

さらに似たような文字に、en ダッシュ記号（–）や em ダッシュ記号（—）があります。日本語の長音（ー）も違う文字です！

オプションとして空白を指定する方法

空白をオプションに含めるには工夫が必要です

通常、コマンド入力での空白文字（スペース）はコマンド名および複数のオプションを区切る意味として認識されます。オプションとして空白文字を指定する場合、あるいはオプションに空白文字を含めたい場合は ’ (シングルクオート)で文字列全体を囲みます。

```
touch ’hoge fuga.txt’ ↩
```

この例は「 hoge fuga.txt 」というファイルが作られます。

```
touch hoge fuga.txt ↩
```

この例はシングルクオートで囲っていないため、「 hoge 」と「 fuga.txt 」がそれぞれ引数として扱われますので、この二つのファイルが作られます。

もう一つの方法としてバックスラッシュ（ \ ）でエスケープ（回避）する方法があります。バックスラッシュに続けてスペースを入力するとそのスペースはコマンドや引数の区切りではなく、スペースの文字として扱われます。

```
touch hoge\ fuga.txt ↩
```

この例では、スペースの直前にバックスラッシュがあるため「 hoge fuga.txt 」という一つのファイル（引数）が指定されたとみなされます。

ファイルの種類とテキスト・ファイル指向について

Linux ではテキスト・ファイルを多く扱います。

ファイルは単なるデータの列ですが、大きく分けて2つの種類に分けられます。1つは「**テキスト（形式）・ファイル**」、そしてもう1つは「**バイナリ（形式）・ファイル**」と呼ばれるファイルです。

テキスト・ファイルはファイル中に文字だけが存在し、そのまま画面上で見たり読んだり、またはテキスト・エディタと呼ばれるソフトウェアで編集することのできるファイルです。バイナリ・ファイルは基本的に画面に文字としては表示できないデータを含むファイルです。そのため、無理矢理画面に表示しようとすると不正な文字コードや画面を制御する制御コードが無作為に入っているため画面表示がおかしくなってしまいます。画面表示がおかしくなった場合、Ctrl + l（エル）で、画面をクリアすることができます。コンピュータ（CPU）が直接実行できる実行形式ファイルはバイナリ・ファイルの一種です。

ゲノム解析ツールだけでなく Linux のソフトウェアの多くがテキスト・ファイルを扱います。例えばプログラミング言語 Python で書かれたプログラムもテキスト・ファイルですし、ゲノム解析ツールの1つである BLAT もテキスト・ファイルとして書かれたゲノムファイルを扱います。テキスト・ファイルを入出力とするコマンドが多くあり、またテキスト・ファイルの加工や処理を行うコマンドも多く用意されています。

Unix (Linux) コマンドやアプリは基本的に入出力をテキストで行うことを前提としているものが多くあるため、1つのコマンドの出力を別のコマンドの入力としたり、入力ファイルや出力結果を直接編集することも容易です。Linux はこういった「テキスト指向」の設計を採用しているアプリが多いため、さまざまなアプリケーションを組み合わせることも容易になっています。

テキスト・ファイルには形式がありませんが、大まかなルールはあります。1つは行指向で、例えば何かのリスト（遺伝子名など）を用意する場合は、1行に1遺伝子のファイルを用意するとさまざまなアプリで入力データとして使いまわせます。また、1行1データが基本ですが、複数カラムのデータを記載する場合は、タブ記号あるいはスペース記号で区切るのが一般的です。これを前提としたコマンドやアプリが多くあります。このルールをユーザ側も守ることで、さまざまなアプリを組み合わせることが容易になりますので、ぜひ実践するようにしましょう。

テキスト・ファイルを扱う上で1つ注意点があります。 Linux と Windows でテキスト・ファイル中で改行を表す文字コード（改行コード）が異なる点です。Windows でのテキスト・ファイルでは、 CR と LF という2文字分の制御コードで改行を表しますが、Linux では LF だけで改行になります。したがって、Windows で作成したテキスト・ファイルを Linux で編集しようとすると、各行末に CR が余分についているとみなされます（いわゆる「ゴミがつく」）。ただし、あとで紹介するテキスト・エディタは Windows 用のテキスト・ファイルを自動認識しますので、エディタでの編集は問題なくできます。

標準入力・標準出力・標準エラーとリダイレクト

標準入出力とリダイレクトについて説明します。

コマンドが出力する実行結果の文字は、通常端末ソフトウェア（SSHクライアントなど）の画面に表示されますが、この出力先を「**標準出力 (standard output)**」と言います。コマンドは画面に出力しているのではなく、標準出力に出力しています。

コマンドの実行時にパラメータなどの指定が間違っていたり、実行に不具合があったときエラー表示がされます。これも画面に見えますが、実は、エラーメッセージを出力する場合コマンドは「**標準エラー (standard error)**」と呼ばれるものに出力していて、標準出力とは別の出力先になっています。直接結果に関係のないエラーや実行ログのようなものも一般的に標準エラーに出力されます。

何も指定していない場合、標準出力、標準エラーへの出力は画面にそのまま表示されますが、この出力先を画面ではなくファイルに変更することができます。これを「**リダイレクト (redirect)**」と言います。

具体的には、コマンド実行時に、末尾に「 > ファイル名」と付けると、標準出力の出力先が指定されたファイルに変わります。また「2> ファイル名」とすると、標準エラーの出力が指定されたファイルに変更されます。これは同時に指定できますので、標準出力と標準エラーに出力される内容をそれぞれ別に保存することも可能です。

例えば、

```
$ command1 > result.txt ↩
```

これで `command1` の標準出力への出力、つまり実行結果がファイルにリダイレクトされ、result.txt として保存されます。

また「 `2>1` 」と指定すると標準エラーへの出力が標準出力にリダイレクトされます。また「 `2>&1 ファイル名` 」とすると標準出力、標準エラー両方のファイルが1つのファイルに同時に出力されます。

「 `>` 」はファイルがすでに存在する場合、一旦消去してしまいます。出力結果を既存ファイルに追記したい場合は「 `>>` 」（>を2つ）を使用します。

```
$ command1 >> result.txt ↩
```

この例では result.txt は消されずに出力がファイルに追記されます。

ユーザからの入力を受け付けるコマンドの場合、「**標準入力 (standard input)**」から入力を受け付けます。これも同様にリダイレクトでき、「 < ファイル名」とすると画面にキーボードで入力する代わりに指定したファイルから標準入力に送られます。

パイプ

1つのコマンドの出力をそのまま次のコマンドの入力にすることができます。

コマンド A の結果（標準出力に出力されるテキスト）をそのままコマンド B の入力にしたい、とします。リダイレクト機能を使って一旦ファイルに保存することで、これは実現できます。

```
$ command_a input.txt > result_a.txt ↩
$ command_b result_a.txt > result_b.txt ↩
```

しかしこれではコマンド実行がファイルに入出力の速度に影響を受けますし、無駄な中間ファイルも作られるなど非効率です。これをファイルを介さずに一度にやる方法があります。具体的には | (パイプ記号) で2つのコマンドをつないで実行すると、1つ目のコマンドの標準出力への出力が、次のコマンドの標準入力にそのまま送られます。

```
$ command1 | command2 ↩
```

これをパイプといいます（またはパイプでつなげる、と表現します）。パイプで繋げられるのは2つだけではなく好きなだけ（メモリの許す限り）増やせます。

よくやるコマンドによるテキスト処理に、「複数のファイルを結合後、アルファベット順に並べて重複行を削除する」というのがあります。個別にやろうとすると

```
$ cat *.txt > all.txt ↩    # 結合
$ sort all.txt > sorted.txt ↩   # ソート
$ uniq sorted.txt > result.txt ↩   # 重複の削除
```

となりますが、パイプを使うと

```
$ cat *.txt|sort|uniq > result.txt ↩
```

と、一行で一切中間ファイルを作らずに実行できます。

※ sort は複数ファイルを指定できるので cat ではなく sort *.txt でもできます。

※ これらのコマンドは P.42 以降で説明します。

シェル変数と環境変数

シェル変数と環境変数について説明します。

シェルでは変数が使えます。変数には主に2種類あり、シェル変数と環境変数に分けられます。シェル変数はプログラミング言語における変数と同様で、さまざまな値を入れておき、後で参照することができます。

シェル変数に値をセットするには

```
$ variable=value ↩
```

というようにシェル・プロンプトで実行します。ここで *variable* は変数名、*value* は代入したい変数に値（文字列）です。例えば var に文字列 test.txt をセットするには

```
$ var=test.txt ↩
```

と入力します。変数の値は変数名の前に $ をつけることで参照できます。

```
$variable↩
```

例えば、先ほどセットした変数 var を表示するには echo コマンドを使用して

```
$ echo $var ↩
test.txt
```

という様に実行します。

環境変数はシェル変数に似たものですが、シェルの動作の設定に使われています。シェル変数はそのシェルでのみ有効な変数ですが、環境変数はシステム全体で有効になります。環境変数の値を変えることで、シェルの動作を変更できます。

env コマンドで現在設定されている環境変数を表示することができます。

環境変数を設定するには変数設定の際に export を前につけます。

```
$ env ↩
```

```
$ export variable=value ↩
```

コマンド・サーチ・パス

シェルがコマンドを探す仕組みについて説明します。

シェル・プロンプトでコマンドをコマンド名だけで入力した場合、特定のディレクトリに存在するコマンド（実行可能ファイル）が実行されます。この特定のディレクトリのことを「**コマンド・サーチ・パス (command search path)** 」と言います。コマンド入力中にコマンド名をタブ補完することができますが、補完の候補となるコマンドは、このコマンド・サーチ・パスから検索しています。コマンド・サーチ・パスにないコマンドは、相対パスまたは絶対パスで指定しないと実行できません。特に、現在のディレクトリにある実行可能ファイルは「 ./ 」をコマンド前に付けないと実行できません。

コマンド・サーチ・パスは前ページで説明した環境変数である PATH で設定が可能です。echo コマンドで現在の設定内容を見てみましょう。

```
$ echo $PATH ↩
/usr/local/bin:/usr/bin:/bin
```

このように PATH にはコマンド・サーチ・パスが「 : 」（コロン）で区切られた文字列として設定されています。あるコマンドやプログラムがコマンド・サーチ・パスのディレクトリ内に含まれることを「パスが通る」と表現します。コマンドにパスが通っていないとシェル・プロンプトで実行できません（シェルがコマンドを見つけられません）。

前ページの説明の通り、PATH の値を変更することで、ユーザ自身でコマンド・サーチ・パスを変更することができます。気をつけないといけないことは、PATH を変更する際に、これまでの設定に追加するようにディクトリを指定しないと、普段のコマンドが使えなくなってしまいます。したがって通常は以下のように設定します。

```
$ export PATH=$PATH:new_directory ↩
```

このようにすることで、現在の設定に追加でディレクトリを PATH に追記できます。

アプリケーションによっては、このパスの設定をユーザに要求するものもありますので、その際は自分でやらなければいけません。

PATH のデフォルトの設定にあるように、/usr/bin や /bin あるいは /usr/local/bin といったディレクトリが、コマンドがインストールされる一般的な場所になります。ls コマンドで見てみれば、どんなコマンドがインストールされているのかわかります。

コマンド・エイリアス

コマンドに別名が付けられます。

シェルの機能として、コマンドに別名（エイリアス, alias）を付けられます。これを「コマンド・エイリアス」と言います。単にエイリアス、という場合も多いです。エイリアスにはオプションを含められますので、必ずセットで使うオプションがある場合など、コマンドと同名のエイリアスをオプション付きで設定しておくことで、毎回そのオプションを指定することを省略できます。

エイリアスの設定には `alias` コマンドを使用しますが、オプションなしで実行することで現在の設定の一覧を表示できます。

```
$ alias ↩
alias ls='ls --color=auto'
alias ll='ls -l --color=auto'
…
```

エイリアスの設定はこの一覧表示にあるように

```
$ alias alias='設定したいコマンド' ↩
```

で設定できます。*alias* が「設定したいコマンド」に付けたい別名になります。以後、alias がコマンドとして使えるようになります。
エイリアスが設定されているコマンドを、その設定をキャンセルし、元のコマンドそのままで実行したい場合には、コマンド名の前に「 `\` 」（バックスラッシュ）をつけることでその時だけ設定を無視させることができます。

例

`ls` コマンドに、常に --color=auto （色付け設定）オプションをつけるようにする。

```
$ alias ls='ls --color=auto' ↩
```

`ls` コマンドのエイリアス設定を無視して実行する

```
$ \ls↩
```

`rm` コマンドでのファイル削除の際に常に確認するようにする。

```
$ alias rm='rm -i' ↩
```

Bashの設定ファイル

Bash の設定ファイルについて説明します。

環境変数やエイリアスの設定はシェル・プロンプト上で実行しても、シェルを終了したり、ログアウトしたりすると、その設定した内容は消えてしまいます。シェルにはログイン時などに自動で読み込まれる設定ファイルがあり、そこに記述しておくと、ログインするたびに自動的に設定されるようになります。

Bash の設定ファイルは主に2つあります。1つ目は ~/.bash_profile （ホーム・ディレクトリにある .bash_profile という隠しファイル）です。これはユーザがログインした時に読み込まれます（後述のシェル・スクリプトとして実行されます）。Linux によっては、デフォルトで、次に説明する2つ目の設定ファイルを単に読み込む、という設定になっている場合も多いです。

2つ目の設定ファイルは ~/.bashrc で、こちらは Bash が起動されるたびに読み込まれます。これらの2つの設定ファイルは、各行がコマンド入力と同じように実行されます。したがって、環境変数 PATH やコマンド・エイリアスへの永続的な設定が必要な場合は .bashrc に書いておくと良いでしょう。

これらの設定ファイルを編集するにはこの後で説明するテキスト・エディタを使用します。

アプリケーションによっては、インストール時に自動的に .bashrc に設定を書き込むものもあります。

シェルの便利な機能・使い方

シェル・プロンプトを使いこなす上でのヒントです。

- スペース自体を入力する

  シェル・プロンプトではスペースは区切り文字として扱われるのはすでに説明した通りです。引数にスペース文字を渡したい場合はどうしたら良いでしょうか。いくつか方法があります。1つはバックスラッシュを入力し次にスペースを入力するとコマンドの区切り文字ではなくスペースとして扱われます。つまり「`\ `」のように入力します。ファイル名にスペースが入っている場合もバックスペースに続けてスペースを入力すれば問題ありません。補完も効きます。もう1つの方法はシングルクオートなどで囲む方法です。

- 長いコマンドを入力する

  1行に収まらない長いコマンドを入力するとシェル・プロンプト上で折り返され問題なく入力編集できますが、コマンドを実行せずに改行だけする方法もあります。行末に「`\`」（バックスラッシュ）を入力してすぐに改行すると、行頭に「＞」と表示され、続けて入力できます。同じ方法で何行にもコマンド入力を分割できます。最後にバックスラッシュをつけずに改行すれば、全てが1つのコマンド入力として実行されます。1つ気をつけなければいけないことは「`\`」で改行しても、空白としては扱われませんので、コマンドの引数の区切りごとに改行したい場合はスペースの後に「`\`」を入力するようにしましょう。

- 一度に複数コマンドを実行する。

  1行の入力で複数のコマンドを続けて実行したい場合はセミコロン（;）で区切ります。また、パイプに似ていますが `&&` でコマンドを繋げて実行すると、最初に実行したコマンドが成功した場合のみ、次のコマンドが実行されるようになります。逆に最初のコマンドが失敗した時のみ次のコマンドを実行する方法もあります。その場合 `||` （パイプ2個）で繋げます。

  例：command1 が成功したときだけ command2 を実行する。

```
$ commmand1 && command2 ↩
```

- コマンド実行の保留方法

  コマンド入力中に、別のコマンドを実行したくなったらどうしたら良いでしょうか。実際にそのようなシチュエーションは結構あります。色々方法はありますが、筆者が多用している方法は、Ctrl + a で行頭に移動し、 # を入力し、リターンです。これで、そのコマンドはコメントとして扱われますので、何も起こりません。が、履歴には残りますので、別のコマンドを実行した後に、この履歴を表示し、先頭のコメントを削除すればOKです。

シェルの便利な機能・使い方２

実行結果の置換

バッククォート(`)で囲むとコマンドとして扱われ、その実行結果（標準出力結果）に置換されます。

例えば：

```
$ ls `cat hoge.txt` ↩
```

この例では、`cat hoge.txt` の実行結果（つまり hoge.txt のファイルの中身）が、`ls` コマンドの引数として渡されます。

同様な機能として $(コマンド) という書式も使えます。

```
$ ls $(cat hoge.txt) ↩
```

バッククォートを使った場合と同じ結果になりますが、`$( )` は入れ子にできるため、置換結果をさらに置換したい場合はこの方法が便利です。

エディタについて

Linuxでのエディタについて説明します。

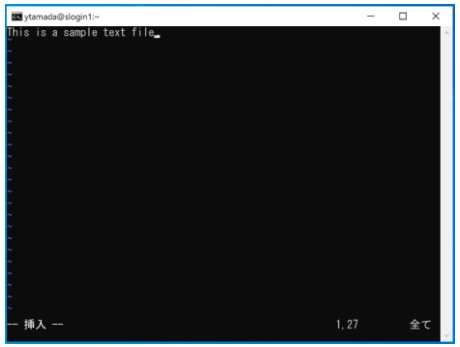
Linux で「**エディタ**」といった場合、テキストを編集するためのソフトウェアである「テキスト・エディタ」を指します。Linux （バイオインフォマティクス） で使われるファイルはほとんどがテキスト・ファイルです。従って、入力ファイルの準備や設定ファイルの作成変更などの際にエディタでテキスト・ファイルを編集する機会が多くあります。

テキストを編集したい時はサーバにログインした状態で行いますので、シェル上で動作する エディタが必要になります。そうでないと、テキスト・ファイルを編集するたびに、いちいちサーバからローカルにテキスト・ファイルを転送し、ローカル PC で編集し、それをサーバに転送し直す、という作業が必要になってしまいます。シェル上で動作するエディタの代表的なものとして Vim（ヴィム）というものがあり、ほとんどの Linux (Unix) サーバにインストールされています。しかし、やや操作に癖があるのが難点です。比較的操作が直感的で容易なものとして Nano（ナノ）というエディタがあります。本テキストでは Nano と Vim の使い方を紹介します。

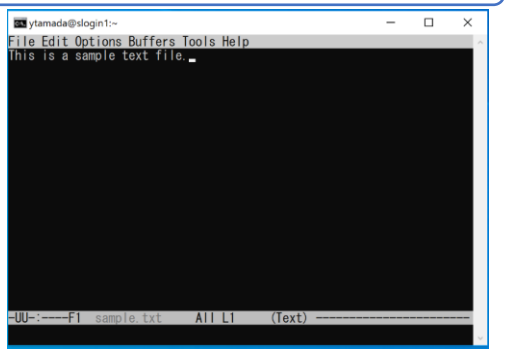
Vim は昔あった VI (Visual editor の略) というエディタの改良版です Vim (VI Improvedの意)。VI と Vim の基本操作は全く一緒です。Vim がインストールされていない環境でも VI はインストールされている場合が多いので、Vim の基本操作ができる様になれば、どの Linux サーバでも困ることはないでしょう。

Vim と並んで有名なエディタに Emacs（イーマックス）というものがあります。Emacs は Vim と比べるとやや動作が遅いですが、専用のプログラミング言語でありとあらゆる機能を追加できるのが特徴です。Emacs 上で動作する電子メールソフトなどもあるくらいです。シェルでのキーボード操作は Emacs と共通のものがあり（例えば Ctrl+aで行頭に移動する、など）、こちらも慣れると非常に効率よくテキストやプログラムの編集ができ、愛好者も少なくありません。筆者は Emacs を普段用いています。

VI が普及する前は、行単位で表示・編集するライン・エディタと呼ばれるテキスト・エディタが一般的に使われていました。今でも Linux で使えます (`ed` コマンド) 。ライン・エディタに対して、画面を自由に移動できるエディタをスクリーンエディタと呼びます。VI が visual editor である所以です。



Vim



Emacs

Nano の使い方

テキスト・エディタである Nano の使い方を説明します。

Nano（ナノ）はシェルで使えるエディタの中で、比較的直感的に使えるため、操作が容易です。大抵の Linux にはインストールされています。まずは Nano を使えるようになりましょう。Nano は `nano` コマンドで起動します。

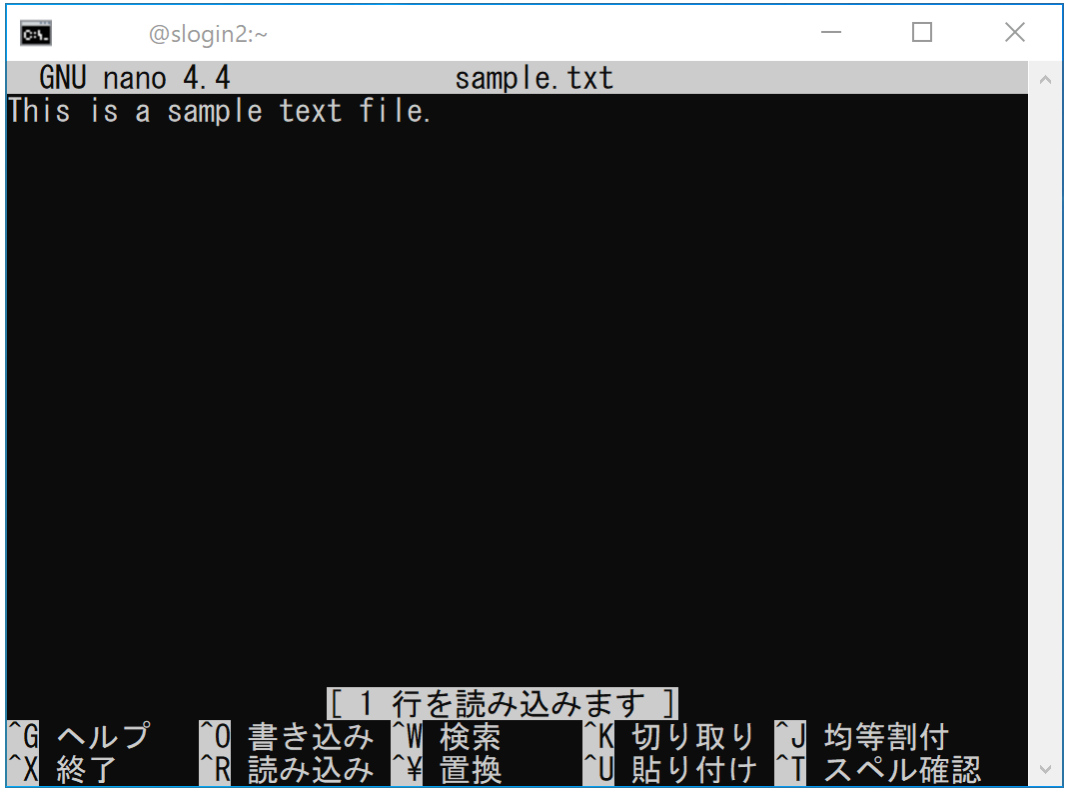
```
$ nano file ↩
```

file には編集したいファイル名を指定します。指定したファイルがなければ新しく作られます。Nano を起動すると右図のような表示に切り替わります。編集しているテキスト・ファイルの内容がそのまま表示されていますので、キーボードから文字を入力するとカーソル位置にそのまま入力されます。カーソルキーでカーソルが移動します。

画面最上部1行と最下部の3行分が状態やメニュー表示に使用されています。「^」は Ctrl キーの意味です。Ctrl キー＋1文字でメニュー表示されているような基本機能を全て操作できます。例えば Nano を終了するには「^X 終了」と書いてあるとおり、Ctrl + x を入力します。終了する際に、ファイルが編集されていれば保存するか聞いてきます。その場合、「Y」を入力すれば、保存して終了します。

このように Nano では基本的に次にすべき操作が画面に表示されていますので、特別覚える必要があることはありません。

SHIROKANE 上で Nano を起動した様子



Nano の便利な操作

基本の操作以外にも便利な操作方法があります。

基本的な操作方法は覚えなくても画面を見ればわかるのがNanoの良いところですが、以下のような操作も可能です。表中の「Ctrl ＋」は Control キーを押したまま次のキーを押すという意味です。シェル・プロンプトと一部共通の操作ができます。

キー操作	説明
Alt ＋ u	直前の操作の取り消し（アンドゥ）
Alt ＋ e	取り消した操作のやり直し（リドゥ）
Ctrl ＋ b	カーソルキーの ← （左移動）
Ctrl ＋ f	カーソルキーの → （右移動）
Ctrl ＋ p	カーソルキーの ↑ （上移動）
Ctrl ＋ n	カーソルキーの ↓ （下移動）
Ctrl ＋ a	行頭に移動
Ctrl ＋ e	行末に移動
Ctrl ＋ h	カーソルの前を削除 (バックスペース)
Ctrl ＋ d	カーソル位置を削除（デリート）

キー操作	説明
Alt ＋ x	画面下部のメニューの表示・非表示の切り替え
Alt ＋ a	マークセットと解除（範囲選択開始）
Alt ＋ 6	マークから範囲選択した箇所のコピー
Alt ＋ \	ファイルの先頭に移動
Alt ＋ /	ファイルの末尾に移動
Ctrl ＋ y	1画面分、前に移動
Ctrl ＋ v	1画面分、後ろに移動
Alt ＋ n	行番号表示の切り替え
Alt ＋ s	長い行を折り返して表示

Vim の使い方

エディタである Vim の使い方について説明します。

前述の通り、CUI でよく使われる（テキスト）エディタに Vim（ヴィム）があります。非常に動作が高速で、CUI で使用することができ、どのサーバでもたいてい使うことができるため、事実上の標準的なエディタとなっています。Nano と違い、直感的な操作は難しいですが、機能が豊富なのが特徴です。 Vim の起動には `vim` コマンドを用います。

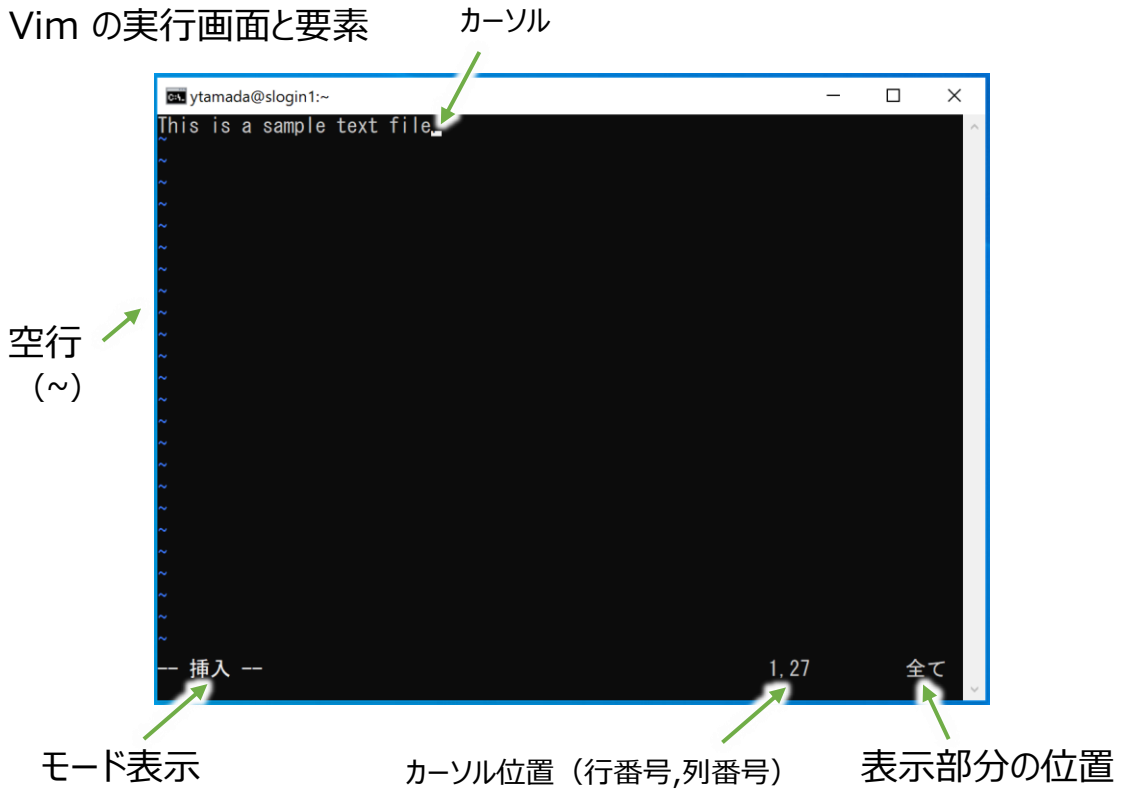
```
$ vim file ↩
```

file には編集したいファイル名を指定します。指定したファイルがなければ新しく作られます。Vim がインストールされていない場合でも VI ならば使える可能性が高いです。`vim` コマンドがない場合、`vi` コマンドを試してください。以降の説明は Vim と VI で共通です。Vim が使える環境では、`vi` コマンドで大抵 Vim が起動します。

Vim には複数の状態（モード）があり、それを切り替えながら使用します。基本的なモードとして「ノーマルモード」と「挿入モード」があります。起動直後は「ノーマルモード」になっています。**ノーマルモードでは文字の直接の入力はできません。**

ノーマルモードではファイルの保存、カーソルの移動、コピーやペーストなどの編集作業ができます。ノーマルモードでは「コマンド」を入力するか、あるいは特定の文字に機能（コマンド）が割り当てられているため、それを入力して実行します。

Vim の実行画面と要素



Vim 基本コマンド

ノーマルモードでの、 Vim の基本コマンドの使い方です。

- コマンド入力方法
  - ✓ ノーマルモードで「 : 」（コロン）キーを押すと、コマンド入力モードになり、コマンドを入力できます。コマンド入力モードではコマンドを入力し、リターンキーを押すとそのコマンドが実行されます。
  - ✓ 「 : 」で始まらないものはそれを入力した時点で即実行されます。
- Vimの終了
  - ✓ Vim を終了させるには :q コマンドを使用します。つまりノーマルモードで「 : 」を入力しコマンドモードに移行し、「 q 」を入力し、リターンキーを押します。これでVimは終了します。
- Vimの強制終了
  - ✓ ファイルを編集した後など、それを保存せずに終了しようとすると警告が出て終了できません。この警告を回避し強制的に終了するには :q! コマンドを使用します。
- ファイル保存
  - ✓ 編集しているファイルを保存するには :w コマンドを使用します。
- ファイル保存と同時に Vim を終了
  - ✓ :wq で保存と終了を同時にできます。ZZ （大文字 z を2回）でも同様になります。

コマンド	説明
カーソル移動コマンド	
h j k l	それぞれ ←↓↑→ の方向にカーソルが移動します。
0	行頭に移動します。
$	行末に移動します。
gg	ファイルの先頭行に移動します。
G	ファイルの最終行に移動します。
編集コマンド	
yy	カーソルのある行全体をコピーします。
p	コピーしたテキストをカーソルの次の行に挿入（ペースト）します。
P（大文字p）	現在の行に挿入します。
x	カーソル位置の文字を1文字削除します。
dd	一行削除（カット）します。カットした行はペーストできます。
u	操作のやり直し（アンドゥ）をします。

Vim文字入力・検索・その他の使い方

Vimでのテキスト入力方法について説明します。

- ノーマルモードの時に `i` でカーソル位置から、`a` でカーソルの次の位置から文字を入力できる「挿入モード」に変わります。
- 挿入モードではワードやメモ帳のように通常の文字入力ができます。
- 挿入モードからは ESC （エスケープ）キーでノーマルモードに戻ります。
- 他の「挿入モード」への切り替え方
  - ✓ `o` でカーソル行の次に、`O` （大文字 o）で現在の行に空行を挿入して、その行の先頭で挿入モードに切り替わります。
  - ✓ `A` で行末から挿入モードに切り替わります。

- ノーマルモードで `/` を入力すると検索モードに切り替わります。このコマンドではカーソルから下方向に検索します。`less` コマンドと共通なので覚えやすいでしょう。
  - ✓ `?` コマンドで検索すると逆に上方向への検索になります。
  - ✓ 検索モードで検索したい文字列を入力し、リターンキーを押スト入力した文字列を検索します。見つかればハイライト表示されます。
  - ✓ `n` で、検索された次の文字列に移動します。`N` で逆方向に移動します。
  - ✓ 検索文字列のハイライトは `:noh` コマンドで止めることができます。

- その他の便利な使い方（ノーマルモード）
  - ✓ 「行番号」＋ `G` コマンドで、入力した行番号までカーソルが移動します。

まだまだ機能は豊富にあります。興味がある方は各自で調べるようにしてください。

ファイルの圧縮と展開

Linux で使われる圧縮ファイルとそのコマンドによる操作について説明します。

Windows/MacOS でよく使われる圧縮ファイルに ZIP があります（拡張子が .zip ）。Linux でもコマンドで ZIP ファイルの圧縮・展開（解凍）をすることができます。ZIP ファイルの展開には `unzip` コマンドを使用します。また、圧縮には `zip` コマンドを利用します。パスワード付き ZIP ファイルにも対応しています。

Linux では zip ファイルよりも `tar` コマンドと `gzip` コマンドの組み合わせがよく使われます。拡張子が `.tar.gz` または `.tgz` のものは `tar` と `gzip` を組み合わせた圧縮ファイルになります。2つのコマンドの組み合わせですが、基本的に `tar` コマンドだけで tar と gzip による圧縮・展開が可能です。

元々、`tar` コマンドは複数のファイルを1つのファイルにまとめる機能（とそれを元に戻す機能）しかありませんでした。これは tar がその名の通り、テープ・アーカイバ（オープンリール・テープを使用した外部記録装置）の利用を想定しているためです。また `gzip` コマンドも1つのファイルを圧縮するコマンドでファイルをまとめたりする機能はありません。したがって、複数のファイルを圧縮する場合、`tar` でファイルを1つにまとめてそれを `gzip` で圧縮するのが一般的でした。gzip で圧縮したファイルの拡張子は `.gz` です。tar でまとめたファイルの拡張子が .tar で、それをさらに圧縮したもの、ということで `tar` と `gzip` による圧縮ファイルの拡張子は `.tar.gz` ということになります。

現在では前述の通り `tar` コマンドだけで圧縮およびその展開もできるようになりました。また最近ではより圧縮率の高い形式である bzip2 または xz の利用が増えています。それぞれ圧縮・展開するためのコマンドが用意されていますが tar コマンドだけでも処理をすることができます。

TAR と gzip / bzip2 などを組み合わせた圧縮ファイルを tar ball （タール）と呼んだりします。あとで説明するオープンソースの解析ソフトウェアは大抵の場合、パッケージが tar ball で提供されています。

主な圧縮ファイル形式のまとめ:

名称（拡張子）	説明
ZIP (.zip)	Windows/MacOS でよく使われる圧縮ファイル
TAR (.tar)	複数ファイルを1つにまとめたもの。それ自体に圧縮機能はない。
gzip (.gz)	Linux (Unix) で昔から使われる圧縮形式
bzip2 (.bz2)	gzip より圧縮率の高い圧縮形式
xz (.zx)	bzip2 より圧縮率は高いが圧縮処理に時間がかかる。

zip / unzip コマンド

ZIPファイルの圧縮・展開をします。

使い方：

```
zip [options] file.zip [ files ] ...
```

基本的なオプション (zip)：

オプション	説明
-r	ディレクトリを再帰的に圧縮する。
-e	パスワード付きで圧縮する。

使い方：

```
unzip file.zip
```

ZIP 形式への圧縮および展開をするコマンドです。パスワード付き ZIP にも対応しています。

使用例:

```
$ unzip file.zip ↩
```

ZIP ファイル file.zip を現在のディレクトリに展開します。

```
$ zip -r file.zip dir ↩
```

ディレクトリ dir とその中のファイルを ZIP ファイル file.zip して圧縮する。

tar コマンド

ファイルの圧縮・展開（解凍）をします。

使い方：

```
tar [options] [ file ] …
```

基本的なオプション：

オプション	説明
c	アーカイブを作成する。
x	アーカイブを展開する。
f	ファイルを処理する。
v	処理したファイルの一覧を表示する。
z	gzip で圧縮・展開する。
j	bzip2 で圧縮・展開する。
J	xz で圧縮・展開する。

ファイルやディレクトリとその中身を1つのファイルにまとめます。または、それを展開します。同時に圧縮・展開することができます。圧縮形式として gzip, bzip2, xz が利用できますが、その場合は z, j, J をそれぞれオプションに付加します。

使用例:

```
$ tar xvfz file.tar.gz ↩
```

圧縮ファイル file.tar.gz を展開します。

```
$ tar xvfj file.tar.bz2 ↩
```

tar と bzip2 で圧縮されたファイル file.tar.bz2 を展開します。

```
$ tar cvfJ file.tar.xz dir ↩
```

ディレクトリ dir を xz で圧縮し、file.tar.xz というファイル名で保存する。

gzip / gunzip

gzip ファイルを圧縮・展開するコマンドです。

使い方（圧縮）：

```
gzip [options] file
```

基本的なオプション：

オプション	説明
-d	展開する。

使い方（展開）：

```
gunzip file.gz
gzip -d file.gz
```

単一のファイルの圧縮・展開をします。圧縮の場合、ファイル *file* は削除され、圧縮されたファイル *file*.gz が残ります（圧縮され、自動的に拡張子がつく）。展開の時も同様で、*file*.gz を展開すると *file*.gz 自体は削除され、展開されたファイルは *file* として保存されます。

使用例:

```
$ gzip file.txt ↩
```

ファイル file.txt を圧縮し、file.txt.gz として保存します。

```
$ gunzip file.txt.gz ↩
```

ファイル file.txt.gz を展開し、file.txt として保存します。

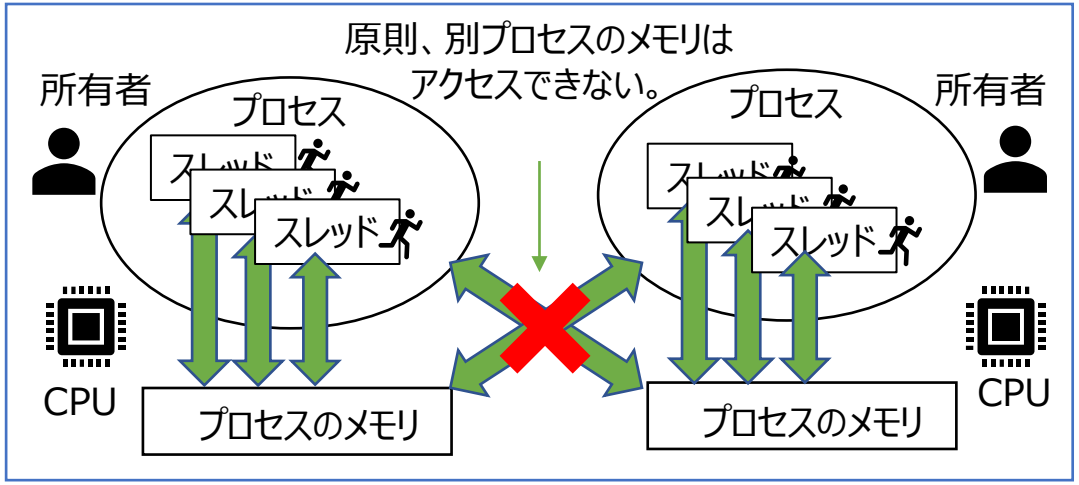
プロセスの管理と停止

Linux におけるプロセスとその制御について説明します。

Linux で動作するプログラム（ソフトウェア）は「**プロセス (process)**」という単位で動作します。シェル・プロンプトからコマンド（ソフトウェア）を実行すると、そのソフトウェアの新しいプロセスが生成され CPU core 上で動作します。同じソフトウェアが複数同時に実行される場合がありますがその場合、それぞれ別のプロセスとして動作します。プロセスは実行した人がその所有者になり、原則的に所有者以外はそのプロセスを勝手に停止できません。ユーザがログインした時は、シェルが1つの新しいプロセスとして起動します。ユーザは自分のプロセスであれば、一時的に停止させたり、強制的に終了させたりできます。こういったプロセス管理のためのコマンドが用意されていますので、次頁以降で説明します。

プロセスには親子関係があります。あるプロセス A が別のプロセス B を生成した場合、プロセス A が親プロセスで B はその子プロセスになります。原則的に、親プロセスが停止すると、子プロセスも停止します。

プロセスと似たような概念に「**スレッド (thread)**」というものがあります。基本的にプロセス同士はメモリが別に割り当てられており、勝手に他のプロセスのメモリの内容を読み込んだり書き込んだりすることはできません。一方、スレッドはプロセスに紐づいて実際に core 上で動くプログラムの単位で、プロセスのメモリを共有します。1つのプロセス内で複数のスレッドが同時に動くこともあります。 CPU が複数の core を持っている場合、スレッドはお互いに速度的に干渉せずに独立に動作することができます。1つの core である瞬間的に動作することのできるプログラム（スレッド）は1つだけです。CPU core は動くプログラムを高速に切り替えることで、同時に動作しているように見せていますが、core が複数あると、それは本当に同時に動作することができます。複数のスレッドを用いて動作するアプリケーションを、「**マルチスレッド・アプリケーション**」と呼びます。アプリケーションがマルチスレッドで動作するかどうかは、そのアプリケーション次第です。マルチスレッドで動作すれば、コア数の多い CPU ではその分高速に動作することになります。ゲノム解析ツールでもマルチスレッド対応のものがあります。スレッドはソフトウェア内で生成され管理されるのでユーザが直接管理することはありません。



ps コマンド

プロセスの一覧を表示します。

使い方：

```
ps [options]
```

基本的なオプション：

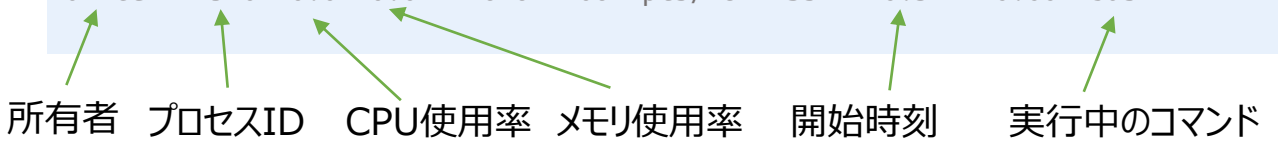
オプション	説明
a	全てのプロセスを表示する。
u	CPUやメモリ使用量を表示する。
x	動作しているプロセスのみ表示する。

オプションを何もつけずに実行すると、そのシェルのプロセスの子プロセス一覧が表示されます。オプション aux の組み合わせでよく使用されます。

プロセスには「**プロセス ID**」が割り当てられています。この ID は後述の `kill` コマンドで、プロセスを停止する際に必要になります。

使用例・出力例:

```
$ ps aux ↩
USER     PID  %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
alice   1369   0.0  0.0 153320  1508 pts/28   R+   21:13   0:00 ps aux
alice  19570   0.0  0.0 127696  4064 pts/28   Ss   10:34   0:00 -bash
```



※ これ以外の説明は manual を参照ください。

kill コマンド

プロセスを停止させるコマンドです。

使い方：

```
kill [options] PID ...
```

基本的なオプション：

オプション	説明
`-s signal`	プロセスに指定したシグナルを送る。
`-l`	シグナル一覧を表示する。

指定されたプロセス *PID* を停止します。より正確には、kill コマンドはプロセスに「シグナル (signal)」というものを送るためのコマンドで、デフォルトでは TERM シグナルというものを送ります。基本的にはこれでプロセスは終了します。TERM シグナルを受け取ったプロセスは必要に応じて終了処理をしたのちに終了します。

TERM シグナルで停止しない場合、強制的に終了させるシグナルを送ることができます。それが KILL シグナルで、-s SIGKILL を指定することで、KILL シグナルを送ることができます。これでプロセスの強制終了をすることができます。

使用例:

```
$ kill -s SIGKILL 1154 ↩
```

プロセス ID 1154 のプロセスを強制終了します。

プロセス ID を調べるには、ps コマンドなどを使用してください（P.76）。

バックグラウンド実行（1）

プロセスのバックグラウンド実行について説明します。

コマンド実行中に、別のことをしたい場合など、コマンドをバックグラウンドで、つまり裏で動かしたまま、表で別のコマンドを実行することができます。

コマンド実行時に、最後に「 & 」（アンド、アンパサンド）をつけて実行するとそのコマンドはバックグラウンド実行になります。バックグラウンド実行するとジョブ番号とプロセス ID が表示され、実行したコマンドが終了する前に、即座にプロンプトが表示され、別のコマンドを入力できるようになります。プロンプトが表示されている間も実行したプロセスは動作し続けます。従ってコマンド入力できる状態のまま、実行したコマンドから出力があれば画面に表示されます。一般的には不便ですので、出力はリダイレクトしておきます。

使用例：

```
$ command &> log.txt &↩
[1] 24443
$
```

バックグラウンドで実行

画面に結果やエラーが表示されないようにリダイレクトする

ジョブ番号 ([1]) とプロセスID (24443)

実行時にバックグラウンド指定する方法以外にも、現在実行中のコマンドをバックグラウンド実行に変更することもできます。そのために、実行コマンドの一時停止・再開機能をまず説明します。

コマンド実行中に Ctrl + z を入力するとコマンドの実行が一時停止されます（サスペンド）。この状態で fg コマンドを実行すると、サスペンドしたコマンドが再開されます。コマンドをサスペンドした後 bg コマンドを実行するとサスペンドしたコマンド（ジョブ）がバックグラウンド・ジョブとして実行されます。

動作中のバックグラウンド・ジョブの一覧は jobs コマンドで表示できます。fg コマンドにはジョブ番号を指定できますので、複数のジョブのうち、fg コマンドでフォアグラウンド実行（通常の実行方法）に戻すジョブを選択できます。

```
$ sleep 10m ↩
^Z
[1]+ 停止      sleep 10m
$ bg ↩    # bg コマンドの実行
[1]+ sleep 10m &
$
```

Ctrl + z を入力し、一時停止する

実行していたコマンドがバックグラウンドに回ったので、別のコマンドを入力できる。
sleep コマンドは再開され、実行中になる。

バックグラウンド実行（2）

ログアウトしても終了しないバックグラウンド実行方法を説明します。

シェルを exit コマンドなどで終了すると、それまでのシェルで実行していたコマンド（プログラム）は停止してしまいます。シェルから実行したプロセスはシェルの子プロセスなので、親プロセスのシェルが終了すると、子プロセスも終了するからです。従って、バックグラウンド実行で、時間のかかるプログラムを実行したままにしたい場合には注意が必要です。実行したままにするにはログアウトできないことになります。

この動作を抑制するには nohup コマンドを使って実行します。これで exit した後（ログアウトした後）もバックグランド・ジョブが走り続けます。

```
$ nohup sleep 10m & ↩
[1] 21344
$ exit ↩
```

← sleep コマンドは終了しない

nohup を指定し忘れた場合、後から nohup で実行したように変更することができます。 disown コマンドを使用して「%ジョブ番号」というように指定します。

```
$ sleep 10m & ↩
[1] 21344
$ disown %1 ↩
$ exit ↩
```

← [1] のジョブを nohup と同様に扱う

nohup コマンドを使うことで、ログアウトした後もプログラムが動作し続けますが、シェル自体からはログアウトしてしまっているため、後になって、コマンド実行時の状態にログインして戻ることはできません。例えばフォラグラウンド動作に変更したりできません。

ログアウト後も、ログアウト前のシェル・プロンプトの状態に完全に戻ることができるアプリケーションがあります。 screen または tmux というアプリケーションがそれですが、本テキストでは解説しません。各自調べてみてください。

プロセスと実行制御のための特殊キー入力

プロセス実行制御のための特殊なキー入力方法があります。

すでに Ctrl + z で停止（サスペンド）する方法に触れましたが、他にもいくつか特殊なキー入力が用意されています。不用意に入力すると意図せず画面が固まったりしますので、慌てず右のいずれかを試してみてください。画面が固まった場合、Ctrl + s が押されて、ロックがかかっている場合が多いです。Ctrl + qで解除しましょう。

キー操作	意味
Ctrl + c	プログラム（コマンド）を（強制）終了する。終了後、シェル・プロンプトに戻る。
Ctrl + z	プログラム（コマンド）を一時停止（サスペンド）する。サスペンド後、fg や bg コマンドが使用できる。
Ctrl + s	入力（画面）をロックする。スクロールも止まる。Ctrl + q でロック解除。
Ctrl + q	Ctrl + s のロックを解除する。
Ctrl + l（小文字のL)	画面のクリア
Ctrl + d	EOF を送信。標準入力を受け付けるプログラムに入力中に、入力を終了する。シェル・プロンプトでは exit と同等。
Ctrl + g	検索などの処理の中止

uptime コマンド

サーバの動作状況を表示します。

使い方：

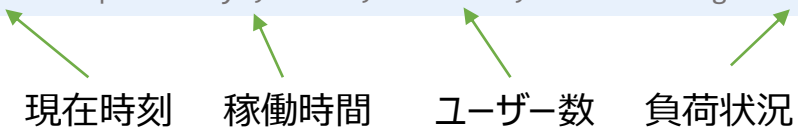
```
uptime
```

uptime コマンドはサーバの動作状況を表示します。具体的には、現在時刻、稼働時間やログイン人数、負荷状況を出力します。

負荷状況は直近1分、5分、15分での実行プログラム（ジョブ数）の平均数を表しています。これが、サーバに搭載されている CPU core 数以上だと、ジョブ数に対してサーバ側の処理が十分にできていないことの目安となります。

使用例:

```
$ uptime ↩
 12:26:25 up 204 days, 12:16, 24 users,  load average: 1.03, 1.19, 1.62
```



top コマンド

Linux 上のタスク状況をリアルタイムで表示します。

使い方：

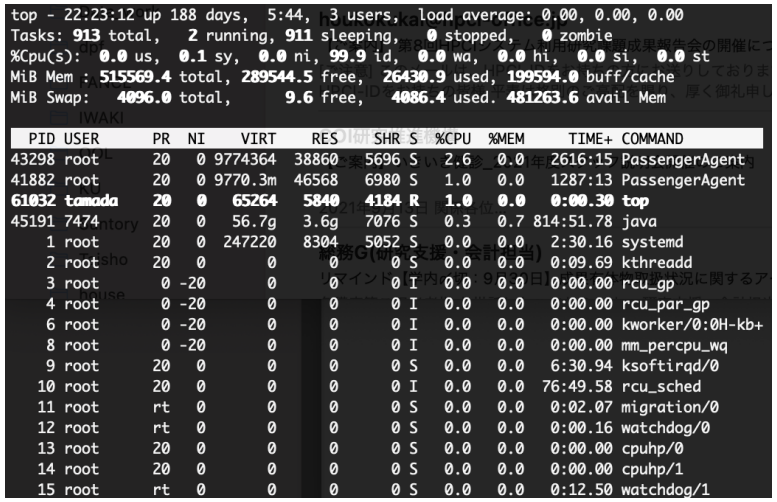
```
top
```

サーバで稼働中のプログラム（プロセス）を負荷が高い順に（CPU利用率順に）表示します。デフォルトでは1秒ごとに情報を更新します。`uptime` コマンドの出力に加え、メモリ、プロセスのリストが表示されます。

終了するには `q` を入力します。`?` で使い方を表示します。

プロセス・リストのソート順なども変更できます。詳しくはマニュアルを参照ください。

ある Linux サーバでの表示例



高度なコマンド

少し使い方が難しい、高度なコマンドのうち比較的頻繁に使うものとして以下のコマンドがあります。次ページ以降で説明します。

grep（グレップ）

正規表現というものを用いてテキスト・ファイルを探索し、マッチした行を出力します。

awk（オーク）

行指向で複雑な処理が可能なプログラミング言語ですが、コマンドとして簡単に実行可能なため比較的頻繁に使われます。

sed（セッド）

こちらも行指向のテキスト処理が可能なコマンドで、特定の行を抜き出したり、文字の置換が簡単にできます。

grep コマンド

正規表現を用いてマッチした行を抜き出すコマンドです。

使い方：

```
grep [options] pattern [ file ] …
```

基本的なオプション：

オプション	説明
-i	大文字と小文字を区別しない。
-v	マッチしない行を出力する。
-n	行番号を表示する。
-a	バイナリファイルをテキストファイルとして処理する

grep（グレップ）は「**正規表現（regular expression）**」というものを用いて文字列パターン *pattern* を指定し、入力テキスト・ファイル *file* 中にマッチする行を表示します。複数のファイルを指定すると、マッチしたファイル名と行を両方表示します。

正規表現はそれだけで本が1冊書けてしまうくらい複雑ですが、単純な使い方であればそれほど難しくはありません。パターン（正規表現）の書き方は次ページを参照してください。

使用例：

```
$ grep gene *.txt ↩
```

現在のディレクトリにある拡張子が .txt のファイル全てから gene と書かれた行を探し、マッチしたファイル名とその行を出力します。

よくwc コマンドと組み合わせて、マッチした行数を数えたりします。

```
$ grep p53 input.txt |wc -l ↩
```

この例では input.txt にある p53 という文字列が含まれる行数を返します。

詳細は次ページで説明しますが、行頭部分にマッチさせる場合は

```
$ grep ^gene input.txt
```

のように ^ をつけると gene で始まる行だけを抜き出せます。

grepで使える正規表現

grep コマンドのパターンの指定方法（正規表現）を説明します。

基本的に ^ $ [ ] * . \ の7文字を「メタ文字」と言い、メタ文字以外の文字はその文字そのものにマッチします。

メタ文字自身にマッチさせたい場合は \ に続けてメタ文字を書きます。例えば \^ は ^ とマッチします。

メタ文字の使い方は以下の通りです。

メタ文字	意味
^	テキストの行頭にマッチする。
$	テキストの行末にマッチする。
.	任意の1文字にマッチする。
*	その直前の文字またはパターンの0個以上の繰り返しにマッチする。
[…]	括弧内のいずれか1文字にマッチする。ハイフンで範囲指定ができる。
[^…]	括弧内の含まれない1文字にマッチする。ハイフンで範囲指定ができる。

パターン例：

パターン	意味
abc	任意の位置の abc にマッチする。
^abc	abc で始まる行にマッチする。
abc$	abc で終わる行にマッチする。
[a-z]	小文字の a ～ z のいずれか1文字にマッチする。
[0-9]	数字1文字にマッチする。

awk コマンド（1）

awk の使い方を説明します。

使い方：

```
awk [options] 'script' [ file ] …
```

基本的なオプション：

オプション	説明
`-F c`	区切り文字を指定する。
`-f file`	スクリプトを *file* から読み込む。

awk（オーク）は、行指向で複雑な処理が可能なプログラミング言語です。コマンドとして簡単に実行可能なため、テキストの処理によく使われます。awk コマンドは *file* を読み込んで *script* に応じて処理を行います。

script は パターン { 処理 } … と指定します。

パターンは BEGIN, /正規表現/, END, 評価式のいずれかで、パターンにマッチした行に対して記述された処理をする、というのが awk の基本になります。パターンを省略することもできます。その際は、全ての行が対象になります。

BEGIN と END はそれぞれ、行を読み込む前、全ての行を読み込んだあと、の意味で、つまり前処理、後処理を記述するためのパターンです。

スラッシュで囲んで正規表現を書くと、マッチした行が処理の対象になります。

パターンと処理は幾つでも続けて書くことができます。

処理の基本

各行のフィールド（列）は、$*n* で *n* 番目の列にアクセスできます。$0 で行全体を表します。

画面に出力するには print 関数を使います。

```
$ awk '{print $2}' file.txt ↩
```

で、file.txt の2列目だけを表示する、という処理になります。

パターンには評価式を書けると説明しましたが、例えば

```
$ awk '$1 == "abc" {print $2}' file.txt ↩
```

で、1列目が abc の時のみ2列目を出力する、という処理になります。

awk コマンド（2）

awk では任意の変数が使えます。変数は 0 で初期化されています。また、数式も書けます。
例えば

```
$ awk '{sum = sum + $1} END {print sum}' file.txt ↩
```

で、1カラム目の合計値を出力できます。（C 言語の += 演算子も使えます）

いくつか組み込みの（事前に定義された）変数が用意されています。

変数	意味
NR	行番号（処理した行数）
NF	処理している行のフィールド（列）数
FILENAME	処理しているファイル名

例えば、最初の行以外について処理をしたい場合、

```
$ awk 'NR != 1 {sum += $1} END {print sum/NR}'
    file.txt ↩
```

と書けます。この例では、1行目以外の1カラム目の平均値を出力しています。

awk では組み込みの関数も定義されています。いくつか紹介しておきます。

変数	意味
int(x)	整数への丸めを行う。
rand()	(0,1) の範囲の乱数を返す。
index(x,y)	文字列 x の中で、y が最初に出てくる位置を返す。
length(x)	文字列 x の長さを返す。
substr(x,a,b)	文字列 x の位置 a から b 文字分の部分文字列を返す。

sed コマンド

sedコマンドは行指向のテキスト処理言語です。非常に機能が多いですが、ここでは典型的な使用方法に絞って解説します。

使い方：

```
sed [options] 'command' [ file ] ...
```

入力ファイルが与えられていない場合、標準入力に対して処理します。

置換

```
sed 's/xxxx/yyyy/g' [ file ] ...
```

出現する全ての *xxxx* を *yyyy* に変換します。*xxxx*（検索文字列）にはgrepコマンドと同様に正規表現を用いることができます。

行の抜き出し

```
sed -n 'np' [ file ] ...
```

n 行目を出力します。 p は指定された行を標準出力に出力する、という意味になります。 sed コマンドは、デフォルトでは入力行を処理後にすべて出力します。-n オプションは処理した行以外の出力を抑制します。この組み合わせで指定行だけを出力することになります。

```
sed -n 'n,mp' [ file ] ...
```

n 行目から *m* 行目までを出力します。

行の削除

```
sed 'n,md' [ file ] ...
```

n 行目から *m* 行目までを削除します。 d は指定された行を削除する、という意味になります。

シェル・スクリプト（1）

実行コマンドをテキスト・ファイルに列挙することで順番に実行させることができます。

シェル・スクリプト (shell script) とはシェルのコマンドが書かれたテキスト・ファイルです。コマンドを実行順に複数書いたシェル・スクリプトを用意しておけば、それを実行することで、スクリプトに書かれたコマンドを順次実行してくれます。

シェル・スクリプトでは、変数や実行の制御（ループなど）を行うことができますので、プログラミング言語と同様のことができます。プログラム言語との違いは、原則シェル・スクリプトに書かれることは、シェル・プロンプトで入力するものと同様のコマンド実行であることです。シェル・スクリプトの書き方だけで本が1冊できてしまうため、詳細は書きませんが、ごく簡単な使い方をこのテキストでは紹介します。

ゲノム解析などでは様々なコマンドを組み合わせて実行して解析を進めますが、これら複数のコマンドの実行手順が決まっている場合、それをシェル・スクリプトとして書いておくことで、いちいち1つのコマンドを入力、実行し、終了を待ち、次のコマンドを実行する、ということを繰り返さず1度のシェル・スクリプトの実行でこれに替えることができます。

もう1つの簡単な使い方として、テキスト・ファイルの編集を複数コマンドを組み合わせて実行する場合などでその実行手順を残しておきたい場合がよくあります。その時、シェル・プロンプトから対話的に実行するのではなく、シェル・スクリプトとして実行手順を書き、それを実行、ということを繰り返すことで、コマンド実行履歴に頼らずに、実行の記録を残す方法があります。

シェル・スクリプトの例

先頭行は必ず `#!/bin/sh` で始めます。これはこのスクリプトがシェル（`/bin/sh`）で実行されることを示すためのものです。

```
#!/bin/sh ↩

# （シャープ）で始まる行はコメントですので無視されます。↩

# Hello World と表示したあと、5秒停止して終了する↩
echo Hello World↩
sleep 5↩
```

`nano` などで上述の内容のテキスト・ファイルを用意してください。

```
$ nano sample.sh ↩
```

このシェル・スクリプトはこのままでは実行権限がないため実行できません。`chmod` コマンドで実行権限を付加します。

```
$ chmod u+x sample.sh ↩
```

シェル・スクリプト（2）

これで実行できる様になりましたが、PATH の通っていないディレクトリにあるプログラム（シェル・スクリプト）を実行する場合、相対パスで指定する必要があります。現在のディレクトリに実行したいファイルがある場合、ファイル名に `./` をつけて、明示的にこのディレクトリにあるファイルを実行することを伝えないといけません。

```
$ ./sample.sh↩
Hello World
$                    # 5秒後にプロンプトが表示される
```

引数の受け取り方

シェル・スクリプトも通常のコマンドと同じように引数を受け取って処理できます。

`$n` で *n* 番目の引数を参照できます。(*n* は数字)

```
#!/bin/sh

echo $1
sleep 5
```

```
$ ./sample2.sh foo↩
foo
```

これを応用すると、ファイル名を受け取って、それに対してあらかじめ決められた複数の処理をする、というようなシェル・スクリプトを簡単に用意することができます。

シェル・スクリプトでは条件分岐や繰り返し処理などの制御構造も記述できますが、本テキストでは範囲外として紹介しません。しかし制御構造なしでも、かなりの処理の自動化ができると思います。ぜひ使いこなしてください。

その他のスクリプトの実行

シェル・スクリプト以外のスクリプト（プログラム）の実行について説明します。

シェル・スクリプトは先頭行に #!/bin/sh と書かれていることにより、シェル・スクリプトとして実行されます。シェル・スクリプトはテキスト・ファイルとして書かれますが、シェル以外のさまざまなプログラミング言語も同様にコマンドとして実行可能です。シェル以外によく使われる Python と Perl を紹介します。

Python （パイソン）

Python は AI（人工知能）分野で標準的に使われているプログラミング言語です。例えば以下のような hello.py というファイル（Python スクリプト）があるとします。

```
#!/bin/python
print('Hello world!')
```

このスクリプトは以下の操作で、実行が可能です。

```
$ chmod u+x hello.py ↩ # 実行権限の付加
$ ./hello.py         # Pythonスクリプトの実行
Hello world!
$
```

Perl （パール）

他によく使われるものに Perl 言語（スクリプト）があります。以下のファイルを hello.pl とします。

```
#!/bin/perl
print "Hello world!¥n"
```

```
$ chmod u+x hello.pl ↩ # 実行権限の付加
$ ./hello.pl         # Perlスクリプトの実行
Hello world!
$
```

Perl スクリプトはコマンドに直接渡して入力ファイルの各行に対して実行して結果を出力することも可能です（sed と似たようなことができます）。その場合 -p と -e オプションを同時に使用します。

```
$ perl -p -e 's/¥n/,/g' input.txt ↩
```

この例は改行を , (コンマ) に置換します。

Javaについて

Javaプログラムについて解説します

Java は汎用のプログラミング言語およびその実行環境です。Java のプログラムは、Java 仮想マシンが用意されているさまざまな OS、ハードウェアで同一のものが実行可能です（プラットフォーム非依存）。もちろん一般的な Linux サーバでも Java プログラムは実行可能です。Java プログラムはハードウェアへのアクセスが制限された「サンドボックス」上で実行されるため、セキュリティ的にも優れているとされます。また実行時に、プログラムが部分的にその OS やハードウェア専用の内部コード（機械語）に変換されるため他のスクリプト言語と比較して、高速に実行されるという特徴もあります。ゲノム解析用のツールも Java プログラムとして書かれているものがあります。

Java プログラムは多くの場合 jar ファイル (Java archive ファイル) として提供されます。jar ファイルは java コマンドを用いて実行します。

```
$ java -jar hoge.jar ↩ # hoge.jar の実行
```

パッケージ管理と OSS アプリのインストール

パッケージ管理システムと *OSS* アプリのインストール

Linux サーバでゲノム解析用のソフトウェアを使用するには Linux サーバにインストールする必要があります。多くのソフトウェア・解析ツールは、それ単体では動作せず、他のソフトウェアを利用して動作します。したがって、インストールしたいソフトウェアが依存するソフトウェアがコンピュータにインストールされていない場合、それらも同時にインストールしなければなりません。また依存しているソフトウェアのバージョンが古いと動かないという場合もありますので、必要なソフトウェアがすでにインストールされていても、どのバージョンであるかを把握する必要があります。使用している環境にあったソフトウェアのインストールと、依存するソフトウェアやそのバージョンをまとめて管理するシステムが「**パッケージ管理システム**」で、Linux ではディストリビューションごとに異なるシステムが利用されています。特定のパッケージ管理システム用のパッケージが提供されているソフトウェアは、そのシステムを利用していれば簡単にインストールが可能です。パッケージ管理システムは 自分が Linux サーバの管理者でなければ使用できません。そうでない場合は、管理者などに使用したいソフトウェアのインストールを依頼することになります。

バイオインフォマティクス（ゲノム解析）ツールの多くが**オープン・ソース・ソフトウェア（OSS: Open Source Software)** として提供されています。OSS はソースコードが公開されているため、自分のサーバの環境に合わせたバイナリをソースコードから作成しインストールすることがパッケージ管理システムによらず可能です。またパッケージ管理システム用のパッケージが使えないソフトウェアも自分自身で管理する必要があります。自分自身で管理、というのは、具体的には、依存している必要なソフトウェアを把握し、その必要なバージョンを全て集め、個別にインストールする、ということです。これはかなり大変な作業なのでパッケージ管理システムが存在します。しかしバイオインフォマティクス関連ツールでは各パッケージ管理システム用のパッケージが提供されていないものも多いです。自分自身で管理する場合、サーバ管理者でなくても、自分のホーム・ディレクトリなどにインストールすることも可能です。

具体的なパッケージ管理システムをあげておきます。Red Hat 系 Linux では DNF や Yum というシステムが使われています。Ubuntu (Debian) では apt が使われています。DNF と apt の使い方は次ページで簡単に説明します。また一般的なOSSアプリのインストール方法を説明します。

科学技術計算向けのプログラム・ライブラリ・実行環境などを丸ごとパッケージ化した Anaconda や Miniconda というソフトウェアもあります。よく使われる OSS アプリが一度で使える状態になるため非常に便利です。また、パッケージ管理機能も搭載しているため、対応している OSS アプリであればこれらのソフトウェアから簡単にインストール可能です。統計処理言語である R もバイオインフォマティクスではよく利用されます。R 自体にパッケージ管理機能が付いているため、R 向け OSS アプリを簡単にインストール可能です。 Bioconductor という R 向けバイオインフォマティクスパッケージが有名でよく使われています。

DNF / apt

パッケージ管理システム DNF と apt の基本的な使い方を説明します。

DNF は `dnf` コマンドに、サブコマンドを指定して使います。インストール時にユーザアカウントから行う場合は `sudo` を頭につけて管理者として実行します。

パッケージのインストール（*package* はパッケージ名）

```
dnf install package
```

パッケージの更新

```
dnf upgrade package
```

パッケージの検索（*string* は検索文字列）
パッケージ名の一部や内容に関するキーワードで検索できます。

```
dnf search string
```

インストール済みパッケージの表示

```
dnf list --installed
```

apt も `apt` コマンドにサブコマンドを指定して使います。使い方は DNF と基本サブコマンドは共通です。

パッケージのインストール

```
apt install package
```

パッケージの更新

```
apt upgrade package
```

パッケージの検索

```
apt search string
```

インストール済みパッケージの表示

```
apt list --installed
```

OSSアプリのインストール

OSSアプリをソースコードからインストールする方法を説明します。

OSS アプリのインストール方法に決まった仕組みやルールなどはありませんが、ほとんどのソフトウェアが一定の習慣にしたがっているため、多くの場合は似たような操作でインストールが可能です。まずは一般的なパッケージのインストール方法を説明します。

1. パッケージの展開と説明書

ソフトウェアのパッケージはター・ボール（ .tar.gz ）で提供されることが多いです。パッケージを展開すると README や INSTALL というファイル名のテキスト・ファイルが含まれます。インストール方法の説明が書かれていますのでよく読みましょう。

2. 環境ごとの設定スクリプトの実行

多くのパッケージに configure というファイル名のスクリプトが用意されています。これはインストールする環境を確認して、ソフトウェアのビルドの際の設定を自動でしてくれるシェル・スクリプトです。このスクリプト自身にオプションなどの説明が書かれている場合もありますので、確認するようにしましょう。例えば、Linux のシステムではなく、ユーザ権限でホーム・ディレクトリにインストールする場合などの設定は、 configure スクリプトを実行する際にオプションでインストール先を指定する場合が多いです。

3. ビルド

コンパイルが必要なプログラミング言語の場合（そうでない場合も）、make コマンドで環境に合わせた実行可能バイナリ・ファイルをビルド（構築）します。make コマンドはソースファイルのコンパイルの依存関係の処理などを自動化するツールです。

4. インストール

同じく make コマンドに install というオプションをつけて実行すると、configure 実行時に設定したインストール先にビルドしたソフトウェアがインストールされます。

OSS アプリ・インストールの一般的な流れは以上になりますが、ステップ2 ～ 3 行はおまじない的に頻発しますので、恐れず慣れるようにしてください。

```
./configure
make
make install
```

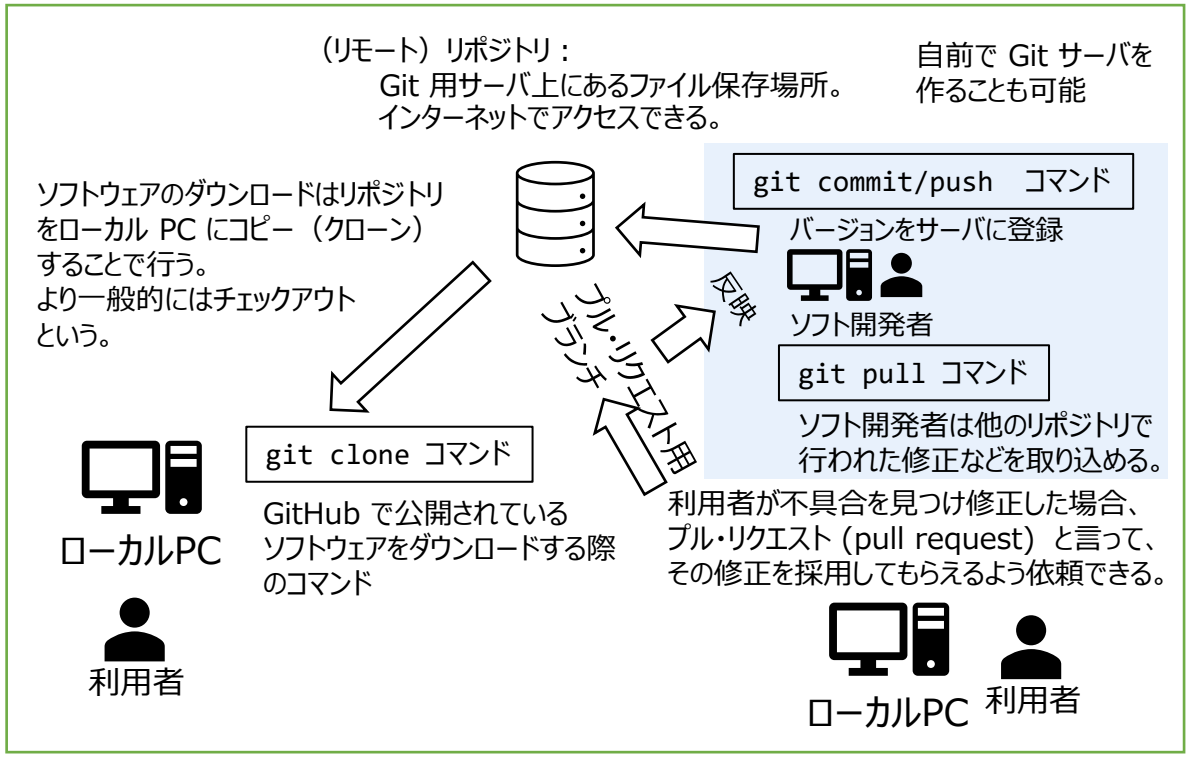
OSSアプリインストール時に頻出する3回のコマンド実行

バージョン管理システム

Git や Subversion などのバージョン管理システムについて簡単に説明しておきます。

バージョン管理システムとは、ファイルの版（＝バージョン）を管理するためのソフトウェアです。バイオインフォマティクスでは様々なソフトウェアを利用して解析を行いますが、解析ソフトウェアは機能が増えたり不具合が修正されたりと常に新しいバージョンに更新されます。ソフトウェアの開発者側からすると、過去の特定のバージョンでの不具合の修正や新機能の開発などの際に備えて全てのバージョンを別々のディレクトリに保存して管理するのは大変です。また、複数人で開発している場合、プログラムの部分的な共有にも手間がかかります。したがって、全ての改変履歴を保存し、いつでも過去の状態に戻れ、ファイルの共有が容易なソフトウェアがあれば非常に効率的に開発ができるようになります。バージョン管理システムはそのようなことを実現するソフトウェアです。

バージョン管理システムとして代表的なものに Git（ギット）や Subversion（サブバージョン）があります。特に Git は、GitHub （ギットハブ）と呼ばれる、Git が使えるサーバを提供しているサービスがあり、非常に多くのオープンソース・ソフトウェアが GitHub 上で開発されています。ソフトウェアの配布や不具合報告のためのシステムも統合されているため、ソフトウェア開発や配布に必要なほとんど全てを賄えます。GitHub で配布されるソフトウェアでは、従来の .tar.gz ファイルをダウンロードしてインストールするというスタイルではなく、公開されているバージョン管理システムのリポジトリ（ファイル保存場所）から直接そのシステムを使用してダウンロードさせるものもあります。したがって、バージョン管理システムで使われる基本的な用語は把握しておくのが良いでしょう。ここでは下の図を用いて、Git (GitHub) での用語を簡単に説明しておきます。



その他の便利なコマンド

以下のスライドで覚えておくと便利なコマンドを紹介します。

wget / curl インターネットからファイルをダウンローするコマンドです。

seq シェル・スクリプトなどで利用する。数字列を発生させるコマンドです。

time コマンドの実行時間の計測ができます。

watch 自動で定期的にコマンドを繰り返し実行します。

sleep 一定時間、停止するコマンドです。

eval 引数として与えられた文字列をコマンドとして実行します。

python Python プログラムを実行します。

perl Perl プログラムを実行します。

java Java プログラムを実行します。

python, perl, java コマンドは説明済みですので詳しい解説は省略します。

wget / curl コマンド

インターネット上のファイルを URL 指定でダウンロードします。

使い方：

```
wget url
curl -O url
```

サーバ上でインターネットからファイルをダウンロードするときに使用するコマンドです。*url* は https: の部分から入力を始めます。ftp にも対応しています。

curl の方が多機能ですが、ファイルをダウンロードするだけなら wget の方が入力が少ないため多少早いかもしれません。curl は -O オプションなしで実行するとダウンロードしたファイルが標準出力に出力されますので気をつけてください。

サーバ上でファイルをダウンロードすることはしばしば発生しますので覚えておくと良いでしょう。これを覚えておけば、わざわざローカルPCでダウンロードし、サーバに転送する必要はありません。

使用例:

```
$ wget ftp://ftp.ncbi.nlm.nih.gov/README.ftp ↩
```

NCBI FTP サイトの README ファイルをダウンロードする。

seq コマンド

数字の列を発生します

使い方：

```
seq last
seq first last
seq first increment last
```

基本的なオプション：

オプション	説明
-s *string*	区切り文字として *string* を用いる。デフォルトは改行(\n)。

引数を1つだけ指定したときは 1 から *last* まで、引数を2つ指定した場合は、*first* から *last* まで、引数を3つ指定した場合は *increment* ずつ増やしながら *first* から *last* までの数字の列を発生させます。シェル・スクリプト中などで、指定回数回実行を繰り返す場合などに使います。

使用例:

```
$ seq 4 ↩
1
2
3
4
```

この例では 1 から 4 までの数字の列を発生させます。

```
# echo を 4 回繰り返す
$ for i in `seq 4`; do echo x $i; done
x 1
x 2
x 3
x 4
```

シェル・スクリプト中などで回数指定でループを回す場合によく使われます。

time コマンド

コマンドの実行時間を計測します

使い方：

```
time command...
```

指定されたコマンド *command...* を実行し、実行時間を測定します。コマンド終了後に、計測結果を出力します。*command...* にはそのコマンドへの引数を含めることができます。

実行時間は、実時間 (real; 正味の時間)、ユーザ CPU 時間 （user; 実際にユーザ・プログラムが使用した時間）、およびシステム CPU 時間（sys; OS 側で使用した CPU 時間）の3つの時間を出力します。

使用例:

```
$ time cp -r dir1 dir2 ↩

real  0m4.416s
user  0m0.000s
sys   0m0.282s
```

ディレクトリ dir1 を dir2 にコピーした際の実行時間を計測します。この例での real と sys の時間差は、ディスク装置の待ち時間などで CPU を使用していない時間が発生しているために生じています。

watch コマンド

一定時間ごとに他のコマンドを繰り返し実行します。

使い方：

```
watch command...
```

基本的なオプション：

オプション	説明
`-n s`	*s* 秒ごとに実行を繰り返します。

指定されたコマンド *command...* を指定された間隔で繰り返し実行します。デフォルトでは 2 秒に1回、コマンドを実行します。*command...* にはそのコマンドへの引数を含めることができます。

特定のコマンドの実行結果を監視し続けたい時に watch コマンド経由で実行すると、非常に便利です。

使用例:

```
$ watch uptime ↩
```

uptime コマンド (P.81) の結果を監視します。

```
$ watch qstat ↩
```

qstat コマンド (SHIROKANE のみ) の結果を監視します。システムに投入された計算ジョブの状態を監視するときに便利です。

sleep コマンド

指定した秒数停止します。

使い方：

```
sleep n[suffix]
```

n 秒停止するコマンドです。*suffix* には `m`、`h`、`d` が使用でき、それぞれ分、時、日の意味になります。すなわち `10m` で 10 分間という意味になります。特に何も出力されません。

使用例:

```
$ sleep 5 ↩
```

5秒間停止し、経過後プロンプトが戻ります。

```
$ sleep 10m ↩
```

10分間停止し、経過後プロンプトが戻ります。

eval コマンド

引数として与えられた文字列をコマンドとして実行します。

使い方：

```
eval string...
```

文字列をコマンドとして実行します。他のファイルに書かれた内容や変数を用いて組み合わせて実行したい文字列を生成したりして、それをコマンドとして実行したい場合に使用します。

使用例:

```
$ x='sleep 5' ↩
$ eval $x ↩
```

“sleep 5”という文字列が eval に渡されますので、5 秒 sleep が実行されます。

用語集・索引

あ

アクセス権 一般的にはデータやシステムに対する操作・閲覧する権限のことをさすが、Linux では主にファイルに対するユーザあるいはグループに対する閲覧・編集・実行の権限を指す。(P.28)

アプリケーション（・ソフトウェア） 基本ソフトウェア (OS) 上で動く一般的な（応用）ソフトウェア。アプリと略す場合も多い。

エディタ (editor) テキスト・ファイルを編集するソフトウェア。テキスト・エディタ。(P.65)

オペレーティング・システム (Operating System: OS) コンピュータのハードウェアを抽象化し、アプリケーション・ソフトウェアに統一した利用手順を提供し、ファイル操作やアプリケーション操作の方法など、利用者とコンピュータを媒介する役割をするソフトウェア。基本ソフトウェアともいう。(P.5)

か

カーネル (kernel) OS の核(=kernel) となる部分。OS 本体。(P.16)

カレント・ディレクトリ（current directory） 現在の作業ディレクトリ。(P.23)

公開鍵 (public key) 公開鍵方式の認証で鍵として利用するファイルの1つ。秘密鍵とペアで使用する。公開鍵はその名称の通り公開しても安全性に支障がない。ログインしたいサーバ側に置くと、対応する秘密鍵でそのサーバにログインできるようになる。↔秘密鍵 (P.11)

コマンド (command) コンピュータに対する命令。シェル上ではシェル・プロンプトで入力して実行する機能またはシェルから起動するソフトウェアを指す。

コマンド・プロンプト（command prompt) Windows に標準で搭載されている CUI の端末ソフトウェア。Windows 自体を CUI で操作できる。SSH クライアントをコマンド・プロンプト上で利用することで、Linux 用の端末ソフトウェアとして利用できる。

コメント (comment) 注釈。スクリプト中などで、ユーザのために書かれるプログラムとしては機能しない注釈文。スクリプトでは主に # （シャープまたはパウンド記号）以降、改行されるまでがコメントとして扱われる。プログラミング言語により異なる文字を用いる。

さ

サーバ (server) ネットワークなどを介し外部から利用することを前提としたコンピュータ。またはそこで動作するソフトウェア。

シェル (shell) OS を操作する環境をユーザに提供するソフトウェア。OS の kernel (核) を覆うような存在からシェル（殻）と呼ばれる。(P.16)

シェル・スクリプト (shell script) シェルで動作するプログラム言語。シェルに対するコマンド入力記法がそのまま使える。(P.89)

用語集・索引

スレッド (thread) プロセス内で並列に動作するプログラム。プロセスから生成される。プロセスは原則メモリ空間が独立しているのに対し、同一プロセス内の別スレッドはメモリ空間を共有している。CPUコア分のスレッドを使えば、その分プロセスは速く動作する。→ マルチ・スレッド

ソフトウェア (software) コンピュータで動作するプログラム。物理的な実体があるハードウェアに対する用語。

た

端末 (terminal) サーバを利用するための手元にあるコンピュータ。

端末ソフト (terminal software) サーバに SSH などで接続し、主に文字情報をやりとりするためのソフトウェア。端末アプリ。

ディストリビューション (distribution) Linux カーネルとその周辺ソフトウェアを集めた配布 (= distribution) パッケージ。同じ Linux でもディストリビューションごとに、使用できるソフトウェアやパッケージ管理システムなどが異なる。代表的なものに Red Hat, CentOS, Debian, Ubuntu などがある。

は

パーミッション (permission) →アクセス権（P.28）

バイオインフォマティクス 情報科学を用いて生物学を行う学問分野。コンピュータを利用した医学・生物学研究やその解析手法全般についても用いられる。

バイト (byte) 情報の単位。ファイルやメモリ上のデータの単位。1 バイトは 8 ビット分 (0～255) のデータ。

バイナリ・ファイル (binary file) 端末画面上に表示できる文字以外のデータを含むファイル。

パイプ (pipe) 縦棒記号 (|) のこと。または、縦棒記号を用いて、コマンドの標準出力をそのまま次のコマンドの標準入力にリダイレクトして、実行する機能のこと。

パス (path) ファイル・システムの階層構造上で、特定のファイルやディレクトリの位置を示す文字列。

ビット (bit) 情報の最小単位。0 か 1 あるいは on か off を 1 つ分の値。コンピュータ上では全ての情報がビットの列として表現されている。

標準入力・標準出力・標準エラー →テキスト本文参照（P.57）

ファイル (file) データを HDD などの記録装置に保存するときの単位。ファイルはバイトの列である。

用語集・索引

プロセス (process) Linux でのプログラムの実行単位。Linux 上で動作しているプログラム。

ヘッダ行 (header line) テキスト・ファイルの1行目などで、データではなくデータの意味を記した行。場合により、コメント文字 (# など) で開始する。

ま

マルチ・スレッド (multi thread) ソフトウェアが複数のスレッドを使用して動作すること。バイオインフォマティクスのツールではマルチ・スレッドに対応しているものも多いので、CPUコアが多くあればその分高速に実行できる。

メモリ (memory) 記憶装置。データ（情報）を格納するための装置。容量の大きさをバイト (Byte) で表す。

や

ユーザ (user) Linux 利用の際の単位。アカウント。Linux 使用時には1ユーザとしてログインする。ファイルやプロセスはユーザに紐づくため、ユーザの所有のものとなる。

ら

リダイレクト 標準入力元、標準出力先などを変更すること。（P.57）

リモート (remote) 遠隔。↔ローカル

ローカル (local) 手元あるいは目の前にあるコンピュータや環境。ローカル PC。↔リモート

ログイン (log-in) 特定の利用者（ユーザ）として Linux のコンピュータを利用可能な状態になること。

ログアウト (log-out) ログイン状態から接続を絶ち、接続前の状態にもどること。

用語集・索引

A-Z

Bash (Bourne Again shell) シェルの1つ。現在多くの Linux ディストリビューションで標準のシェルとして採用されている。

core コア。CPU コア。CPU 内部にある実際に計算を行う独立した回路の単位。計算はコア単位で同時に行うことができる。2コアあれば2つの独立した計算を同時に行うことができる。

CPU (Central Processing Unit) 中央演算処理装置。コンピュータに搭載されている、実際に計算を行う集積回路。

CUI (Character User Interface) 文字によるコンピュータの利用・操作環境、方法。

GPU (Graphics Processing Unit) 主に画像処理のための演算回路であるが、近年、GPU の科学計算への利用が進んでいる。同一の演算を多くのデータに対して同時に実行できるため、計算内容によっては非常に高速に実行できる。ゲノム解析ツールでも GPU に対応しているものが多い。GPU の画像処理以外での利用は GPGPU (General Purpose GPU) と以前は言われていたが今では当たり前になりつつあるため単に GPU の利用で通じるようになった。

GUI (Graphical User Interface) 絵による表示を介したマウスやトラックパッドによるコンピュータの利用・操作環境、方法。(P.6)

k (kilo) キロ。1000 の量を表す国際単位系の接頭辞だが、俗習として 1 KB (kilo byte) = 1024 byte である。

Ki (kibi) キビ。1024 (=2^{10}) の量を表す接頭辞。1 KiB = 1024 byte である。

M (mega) メガ。1,000,000 (=10^{6}) の量を表す国際単位系の接頭辞。俗習として 1 MB = 1,048,576 (=2^{20}) byte を表す場合もある。

OS (Operating System) →オペレーティング・システム (P.5)

P (peta) ペタ。1,000,000,000,000 (=10^{12}) の量を表す国際単位系の接頭辞。

shell →シェル (P.15)

SIMD (Single Instruction, Multiple Data) 1つの CPU 命令で、複数のデータに対して行う計算、またはその回路。

SSH (Secure Shell) 公開鍵暗号方式により暗号化された通信方法を用いてコンピュータ同士を接続するための規格。

T (tera) テラ。1,000,000,000 (=10^{9}) の量を表す国際単位系の接頭辞。

基本コマンド一覧（1）

本テキストで取り上げたコマンド一覧です。

コマンド	説明（掲載ページ）
alias	エイリアスを設定する (P.61)
apt	パッケージ管理コマンド (P.94)
awk	行指向のプログラミング言語 (P.86)
cd	現在のディレクトリを変更する (P.24)
cat	ファイルを出力する・つなげる (P.42)
cp	ファイルをコピーする (P.33)
cut	テキスト・ファイルの列を取り出す (P.46)
curl	インターネットからダウンロードする (P.98)
chmod	パーミッションを変更する (P.38)
dnf	パッケージ管理コマンド (P.94)
echo	画面に文字列を表示する (P.53)
eval	文字列をコマンドとして実行する (P.103)
exit	シェル・プロンプトを終了する (P.25)
grep	正規表現でファイルを探索する (P.84)

コマンド	説明（掲載ページ）
gzip	ファイルを圧縮・展開する (P.74)
gunzip	圧縮ファイルを展開する (P.74)
head	先頭から指定行数表示する (P.45)
history	コマンド実行履歴を表示する (P.23)
id	ユーザ情報を表示する (P.17)
java	Javaプログラムを実行する (P.92)
join	鍵列で行を結合する (P.52)
kill	プロセスを停止する (P.77)
ls	ファイル一覧を表示する (P.35)
less	ファイルを閲覧する (P.43)
mkdir	ディレクトリを作成する (P.32)
mv	ファイルを移動する (P.34)
nano	エディタ Nano を起動する (P.66)
nohup	シェル終了時に停止させない (P.79)

コマンド	説明（掲載ページ）
paste	行を連結する (P.49)
perl	Perlスクリプトを実行する (P.91)
ps	プロセス一覧を表示する (P.76)
pwd	現在のディレクトリを表示する (P.31)
python	Pythonスクリプトを実行する (P.91)
rm	ファイルを削除する (P.36)
rmdir	ディレクトリを削除する (P.39)
scp	SSHでファイルを送受信する (P.26)
sed	行指向テキスト処理を行う (P.88)
seq	数字列を発生させる (P.99)
sort	並べ替える (P.47)
sleep	指定秒停止する (P.102)
ssh	SSH でログインする (P.12)
tail	末尾から行を抜き出す (P.45)

基本コマンド一覧（2）

コマンド	説明（掲載ページ）
tar	ファイルをまとめる・展開する (P.73)
time	コマンド実行時間を計測する (P.100)
top	サーバー状況を表示する (P.102)
touch	ファイルの変種時刻を更新する (P.40)
tree	ファイル一覧を木構造で出力する (P.37)
uniq	重複行を削除する (P.48)
uptime	サーバの稼働状況を表示する (P.81)
vi	エディタ vi を起動する (P.65)
vim	エディタ Vim を起動する (P.68)
watch	コマンドを定期実行する (P.101)
wc	行数・単語数などを数える (P.44)
wget	インターネットからダウンロードする (P.98)

第 Ⅱ 章
公開データベース活用

大学共同利用機関法人
情報・システム研究機構ライフサイエンス統合データベースセンター
仲里猛留

公開データベースとは

これまでの知見は**公開データベース**に蓄積されている。自身のデータと組み合わせた解析などで再利用・活用可能。

- 公開データベース（DB, Database）とは
  - ✓ 端的には製品でないDB
  - ✓ 誰でも自由に利用可能。場合によってはログインが必要。
  - ✓ 多くが無償提供されており、公共DB(Public Database)とも呼ばれる。
  - ✓ なぜ公開DBを使うのか？：
    - ➢ 一から十まで自身でやる必要はない（予算や時間、手間など）

      例：ある癌のデータはすでに公開DB中にあるので、自身で別の癌のデータをとって比較
  - ✓ 一般的な生命科学情報はNCBI（米国）が大規模。
    - ➢ DDBJ（日本）とも連携（NGSデータなど）
  - ✓ NCBIでカバーできないものや大規模プロジェクトの成果などは独自にDBが構築されている。
- Open access と Controlled access
  - ✓ Open access：誰でも自由にアクセス、ダウンロード可能
  - ✓ Controlled access：誰が利用したか管理されているDB。
    - ➢ 個人が特定されるようなデータの場合、Controlled access とされ、利用は審査・管理される
- 公開DBのデータの利用の記載と登録
  - ✓ 公開DBのデータを使ったら Materials and Methods でデータのID（アクセッション番号）を明記することでデータの出所を尊重する。
  - ✓ データの公開DBへの登録・公開が投稿要領で求められている場合もある。（元データの登録場所も同様に論文中に明記する）
  - ✓ データ登録することで手元の大容量のデータを安全かつ半永久的に管理してもらえるという効用もある。

データをダウンロードして使った際の論文での記述例

Single-cell PCA analysis and gene ontology enrichment

The transcriptional profile data for immune cell and GLU1 and GLU3 were retrieved from the NIH SRA database with the accession code SRP040656 (https://www.ncbi.nlm.nih.gov/sra/). The most marker genes used for grouper cell classification can aligned with mouse marker

自身のデータを登録した際の論文での記述例

Data and Code Availability

The accession number for the RNA-seq data and processed data reported in this paper is DDBJ Sequence Read Archive (DRA): DRA008318 and Genomic Expression Archive (GEA): E-GEAD-342.

さまざまな公開DB（一般）

がん情報のデータベースもさることながら、一般的に遺伝子やタンパク質等の周辺情報のデータベースも多く存在する。

カテゴリ	DB名	URL	概要
遺伝子	GenBank	https://www.ncbi.nlm.nih.gov/genbank/	研究者が登録した遺伝子配列をそのまま収載した遺伝子データベース
	RefSeq	https://www.ncbi.nlm.nih.gov/refseq/	GenBank から作成した全長遺伝子DB
	NCBI Gene	https://www.ncbi.nlm.nih.gov/genbank/	RefSeq にゲノム位置や他データベースへのリンクなどアノテーション情報をつけたデータベース
	Ensembl	http://ensembl.org/	モデル生物を中心としたゲノム（＋遺伝子等のアノテーション）のデータベース
タンパク質	UniProt	https://www.uniprot.org/	タンパク質のデータベース。アノテーション情報の記述がすっきりしていて使いやすい
	Pfam	https://pfam.xfam.org/	タンパク質のドメインのデータベース
実験	SRA	https://www.ddbj.nig.ac.jp/dra/	NGS のデータベース（URLはDDBJのサイト）
	GEO	https://www.ncbi.nlm.nih.gov/geo/	もともとはマイクロアレイの、現在は NGS も含んだ発現解析データのデータベース
	ArrayExpress	https://www.ebi.ac.uk/arrayexpress/	EBI による発現解析データのデータベース。 GEO のデータも取り込んでいる
文献	PubMed	https://pubmed.ncbi.nlm.nih.gov/	NCBI による文献データベース。主に1950年代から。3000万件以上
	PMC	https://www.ncbi.nlm.nih.gov/pmc/	概要（Abstract）だけの PubMed に対してフルテキストを収載する文献データベース

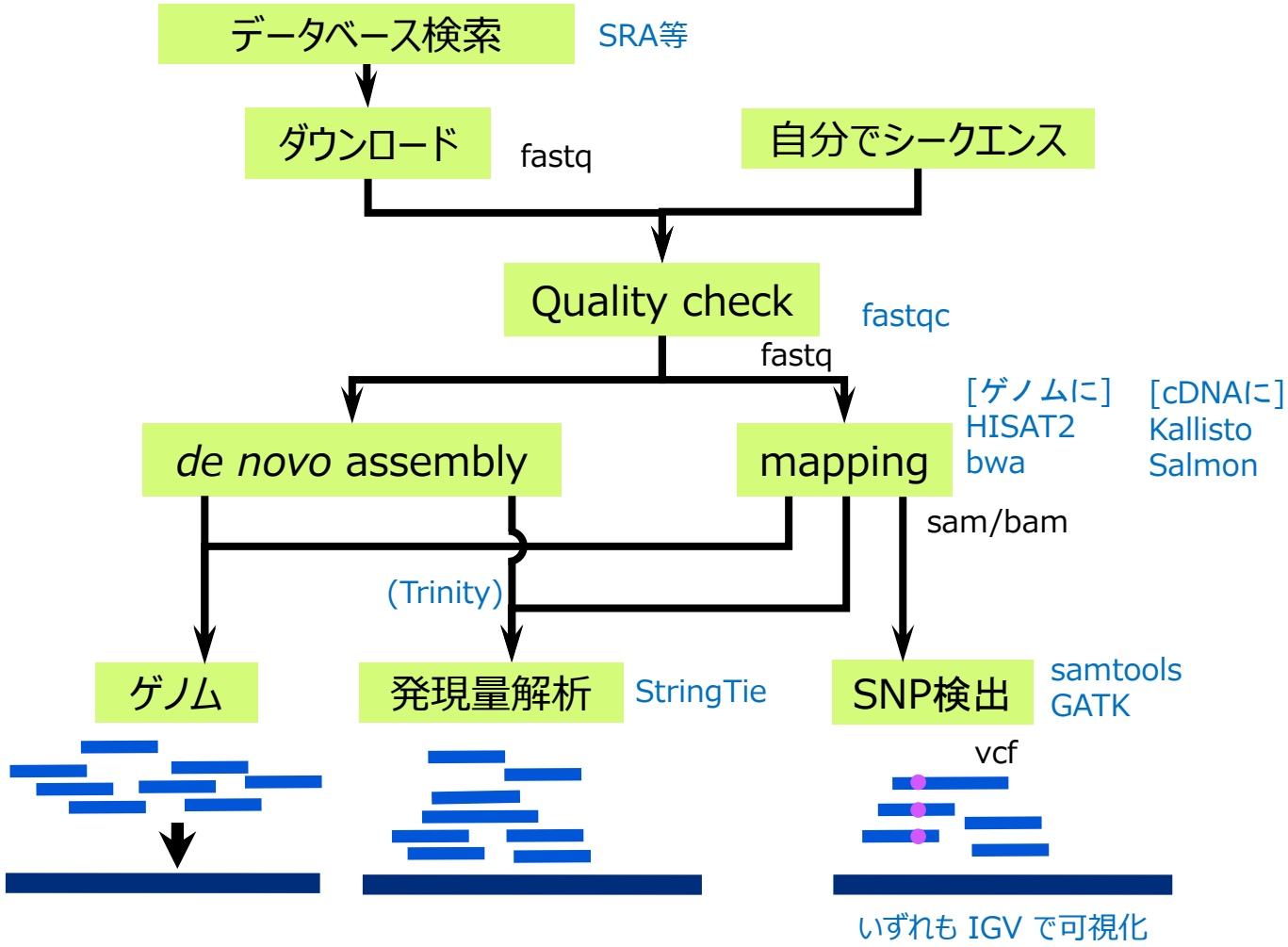
さまざまな公開DB（疾患関連）

疾患に関してもさまざまな公開DBが存在する。

カテゴリ	DB名	URL	概要
疾患	OMIM	https://omim.org/	Johns Hopkins大学。ヒトの遺伝子と疾患のDB。研究分野での疾患データベースとして一般的
	MedGen	https://www.ncbi.nlm.nih.gov/medgen	NCBI 中の疾患情報をまとめたデータベース
	ICD-10	http://www.byomei.org/icd10/index.html	臨床分野（電子カルテの病名など）で用いられる病名用語集（URLは日本語版）
多型	dbSNP	https://www.ncbi.nlm.nih.gov/snp/	NCBIのSNPデータベース
	gnomAD	https://gnomad.broadinstitute.org/	ノマドと読む。ヒト集団ごとの大規模ゲノムバリエーションデータベース
	ClinVar	https://www.ncbi.nlm.nih.gov/clinvar/	疾患に関連するバリアントのデータベース
	TogoVar	https://togovar.biosciencedbc.jp/	日本人ゲノムに存在するバリアントのデータベース
	JGA	https://www.ddbj.nig.ac.jp/jga/	Japanese Genotype-Phenotype Archive。日本人でのヒトデータの NGS 等のデータベース
がん	TCGA	https://www.cancer.gov/about-nci/organization/ccg/research/structural-genomics/tcga	The Cancer Genome Atlas。がんゲノム解析プログラムでの実験結果（NGS等）のデータベース
	COSMIC	https://cancer.sanger.ac.uk/cosmic	Catalogue Of Somatic Mutations In Cancer。ヒトのがんの体細胞変異情報をまとめたDB
	ICGC	https://dcc.icgc.org/	International Cancer Genome Consortium。がんゲノムDB作成を目的とした国際コンソーシアム

NGSデータ解析の流れ

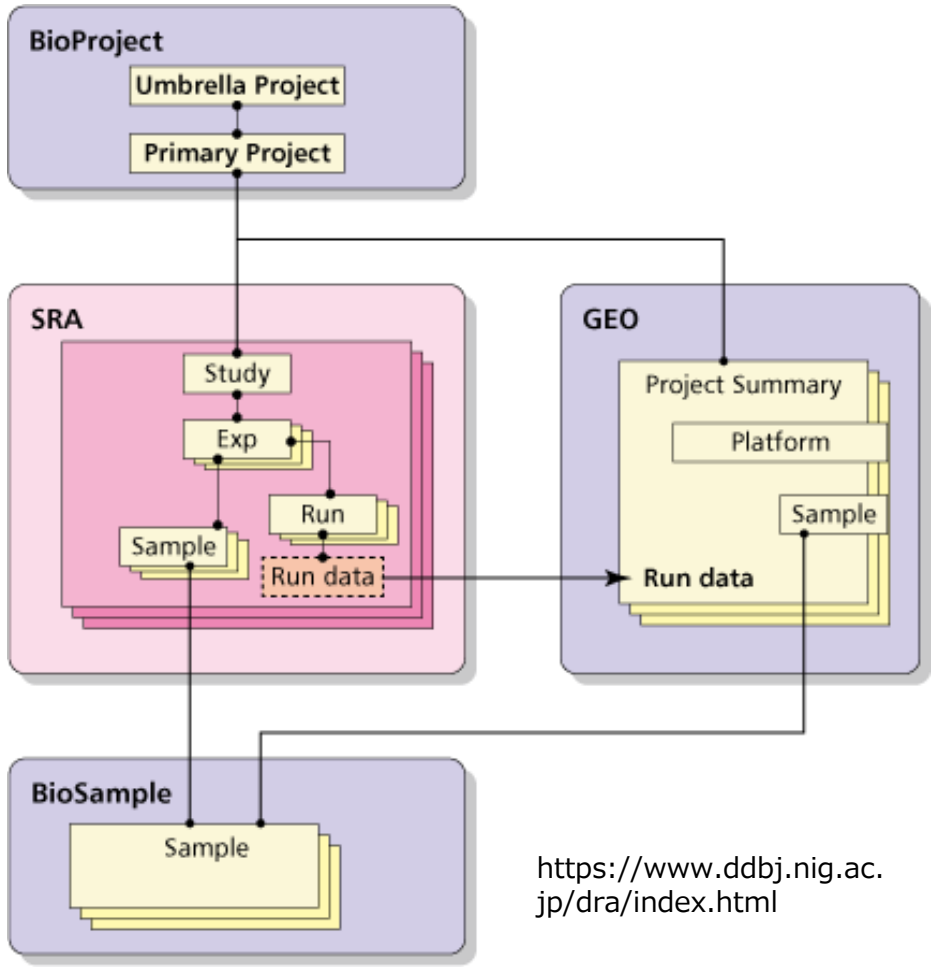
NGSデータ解析の流れを以下のフローチャートで示す。(主に RNA-seq の例)



SRA(Sequence Read Archive)のデータ構造

NGSデータの検索する際に、メタデータ（実験情報）のファイル構造を知っていると便利である。

- Study
  - ✓ プロジェクト情報
  - ✓ 例：がん患者100人ゲノム読みますプロジェクト
  - ✓ 文献情報
- Experiment（Exp）
  - ✓ 例：がん患者について1回の実験についての実験情報
  - ✓ 試料作成ライブラリ
  - ✓ シークエンサー名
  - ✓ Single End / Paired End
- Run
  - ✓ 例：がん患者について1回の実験についての実験データ
  - ✓ fastqファイルへのリンク情報
- Sample
  - ✓ 例： 1回の実験に関するがん患者の情報（年齢、性別、臓器名）
  - ✓ 生物種名
  - ✓ 臓器名
- 発現解析情報はSRAだけでなくGEOにも登録されるので、記載が重複することになるStudyとSampleは各々BioProject/BioSampleのデータベースとして独立



https://www.ddbj.nig.ac.jp/dra/index.html

NGS関連のファイル形式：FASTQ

NGS機器の出力（リード）の共通フォーマットである FASTQ について説明する。

- FASTQ形式
  - ✓ 読んだリードの配列＋そのクオリティが含まれる。
  - ✓ 4行1セット
  - ✓ 通常1つのファイルに数百万～数億のセットが記録されている

FASTQ形式のデータ例

```
@DRR001107.1 GEZQ5FO01EEA7F length=77
GCAACATTCAACACATATGTGTTGAATGTTGCACGACGGNGTG...
+DRR001107.1 GEZQ5FO01EEA7F length=77
C@BBBECCECDBBBAAAAA<441111<?@>?=????44!000...
```

1行目：タイトル
- @ から始まる。あとは特に制限なし
- ソフトによっては形式に厳しいものがあるので、その場合は書き換えが必要

2行目：リード配列
- ほぼすべてがA, C, G, T, N

3行目：見なくてよい
- ＋ から始まる。
- 特に使わない

4行目：クオリティ
- Phredスコア。Q=-10 log p（p=エラー率）
- アルファベットが多ければ問題ない

［参考］
- FASTA形式
  - ✓ GenBankなどで塩基/アミノ酸配列を表す形式
  - ✓ 1行目：> から始まるタイトル行。2行目以降に塩基/アミノ酸配列

FASTA形式のデータ例

```
>AB084425.1 eel SLC26A6
GACCCAAAACTGATAGGTGATGTTCACGTAGTGGC
CATCGCCTGATAGACGGTTTTCGCCCTTTGACGTT
GGAGTCCACGTTCTTTAATAGTGACTCTGAGTAAA …
```

[参考] 詳しい解説：https://bi.biopapyrus.jp/rnaseq/qc/fastq-quality-score.html
[参考] 簡便に絵文字で表してくれるツールがある
FASTQE (FASTQ with Emoji)：https://github.com/fastqe/fastqe

NGS関連のファイル形式：SAM/BAM

Fastqファイルに含まれるリードをゲノム標準配列等にマッピングしたデータのファイル形式の SAM と BAM について説明する。

- SAM（Sequence Alignment Map）形式
  - ✓ テキストファイル（人間が読めるフォーマット）
  - ✓ SAM形式はファイルサイズが大きくなるのでBAM形式に変換する
- BAM（Binary Alignment Map）形式
  - ✓ バイナリファイル（コンピュータ用のフォーマット）
  - ✓ BAM形式はコンピューターが処理しやすい形式なので処理スピードも早い

SAM形式のデータ例

SRR445820.39542705	0	chr17	1	0	4M1I31M	*	0	0	AAAGCTTCTCACCCTGTTCCTGCATAGATAATTGCA	?5>7(+2;'1..'+<
SRR445820.29211975	16	chr17	88	42	36M	*	0	0	CCACGACCAACTCCCTGGGCCTGGCACCAGGGAGCT	######BDB8DACCC
SRR445820.7156374	16	chr17	138	42	36M	*	0	0	CCAGCGAATACCTGCATCCCTAGAAGTGAAGCCACC	BBB=:;BBEABFBFB
SRR445820.22614977	0	chr17	156	30	36M	*	0	0	CCTAGAAGTGAAGCCACCGCCCAAAGACACGCCCAT	GGGD>DBB3D=??=<
SRR445820.19222309	0	chr17	185	42	36M	*	0	0	CGCCCATGTCCAGCTTAACCTGCATCCCTAGAAGTG	IIIIIIIIIIIIHH
SRR445820.32725447	16	chr17	213	31	36M	*	0	0	TAGAAGTGAAGGCACCGCCCAAAGACACGCCCATGT	CGCGGGGDGGGBGAA
SRR445820.43349427	0	chr17	221	31	36M	*	0	0	AAGGCACCGCCCAAAGACACGCCCATGTCCAGCTTA	IIIIIIIIIIIIIII

各リードの名前

mapされた
染色体/scaffold

mapされた場所
（何塩基目か。start/end）

各リードの配列

各リードのクオリティ

NGS関連のファイル形式：VCF

変異を表すためのデータ形式である VCF について説明する

- VCF（Variant Call Format）形式
  - ✓ 変異の情報を含むフォーマット
  - ✓ 変異のID（以下の例ではdbSNPのID）を示している

VCF形式のデータ例

20	14370	rs6054257	G	A	29	PASS	NS=3;…
20	17330	.	T	A	3	q10	NS=3;
20	1110696	rs6040355	A	G,T	67	PASS	NS=2;
20	1230237	.	T	.	47	PASS	NS=3;
20	1234567	microsat1	GTC	G,GTCT	50	PASS	NS=3;

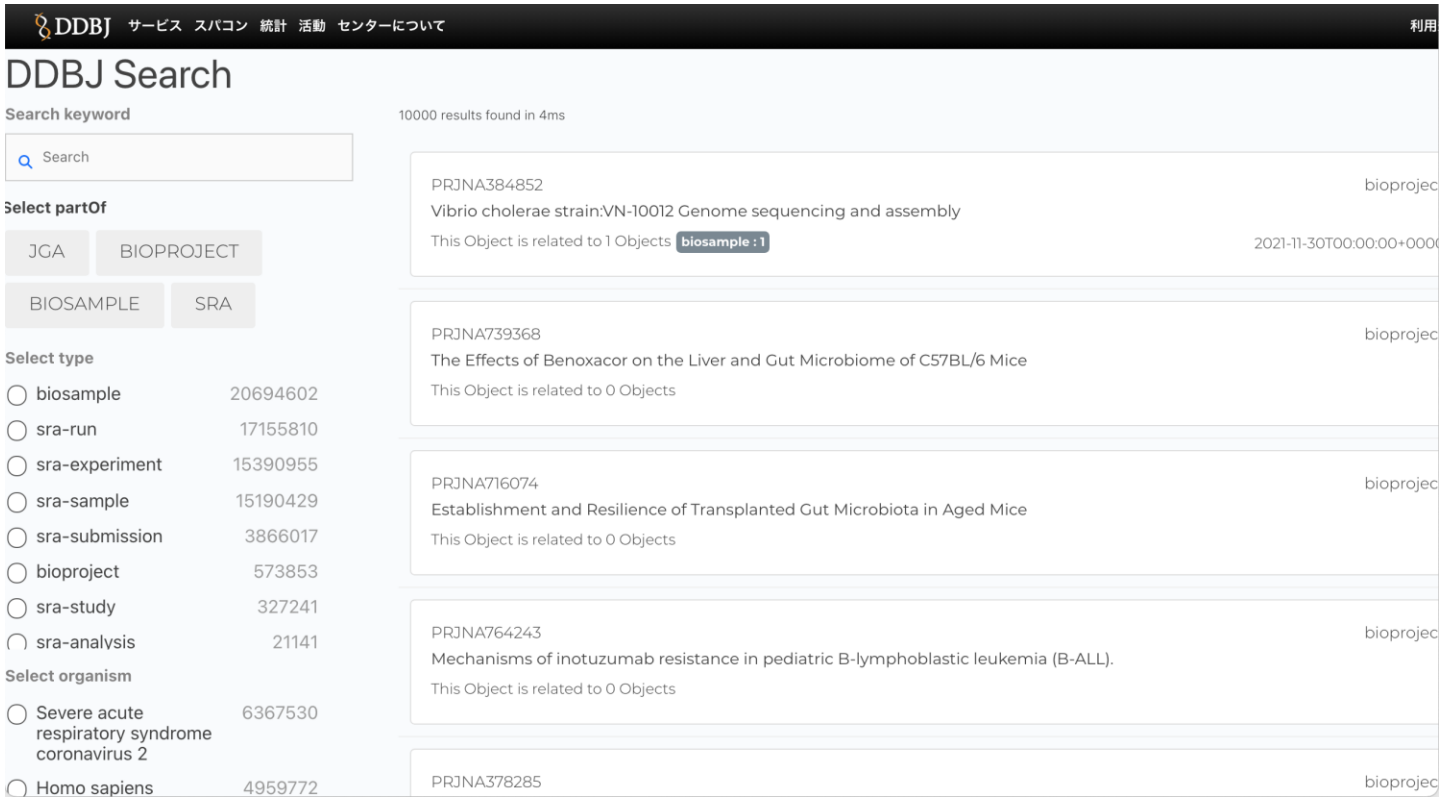
染色体/scaffold　何塩基目か　変異のID　Referenceの塩基　実験結果の塩基

DDBJ SearchでのNGSデータの検索（概要）

NGS の測定結果は **SRA（Sequence Read Archive）** に登録され、公開データベースとして公開されている。

- **SRA** について
  - ✓ NCBI 、EBI 、DDBJ で収集。データは交換し合っている。
  - ✓ それぞれ検索インターフェースがあるが、DDBJ（DRA）が便利
    - ➢ 検索しやすい
    - ➢ データ（fastqファイル）のダウンロードが速く、接続も安定
    - ➢ DDBJ Search：https://ddbj.nig.ac.jp/search
      (以前のものから2021年11月に完全移行しました)
  - ✓ 日本での場合、open accessがSRA、
    controlled accessがJGA
    （open/controlled accessはp.104参照）
    （SRAとJGAはp.106参照）

DDBJ Searchの検索画面



DDBJ SearchでのNGSデータの検索（詳細）（1）

DDBJ Searchにより公開NGSデータを検索できる。

- DDBJ（DRA）での実際の検索手順

① Search keyword には検索用語を入力する。

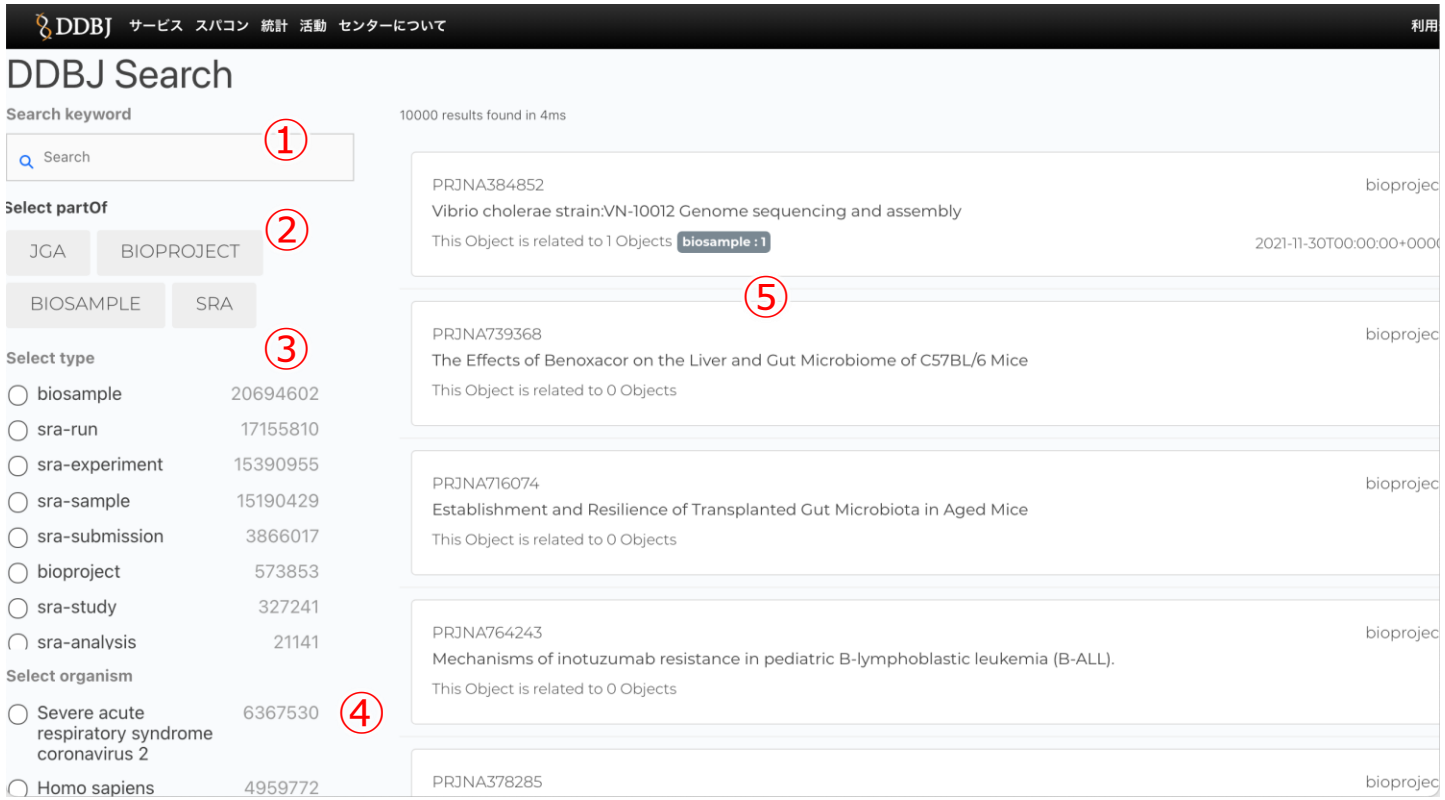
② Select part ofでは、対象のデータベースを絞り込みます。
通常はプロジェクトを検索したいと思うので、
BIOPROJECTとSRAを選択する。（2つ選べる）
ヒトデータ（個人情報を含む）を検索する場合はJGAを、
使った試料から検索する場合はBIOSAMPLEを選ぶ。

③ Select typeでbioproject（特に最近のものを検索する場合）を選択する。（古いデータを見たい場合はsra-studyも見た方がよい場合もある）

④ Select organismで生物種を絞り込める。
今回はHomo sapiensを選択している。
この欄の下に期間で絞り込む欄もある。

⑤ 右側に絞り込まれた結果が表示される。
bioprojectはNGS以外にも使われているが、
実際にNGSデータをダウンロードできるものについては
「sra-run:36」のようなバッジがついている。
興味のあるデータをクリックすると詳細を確認できる。

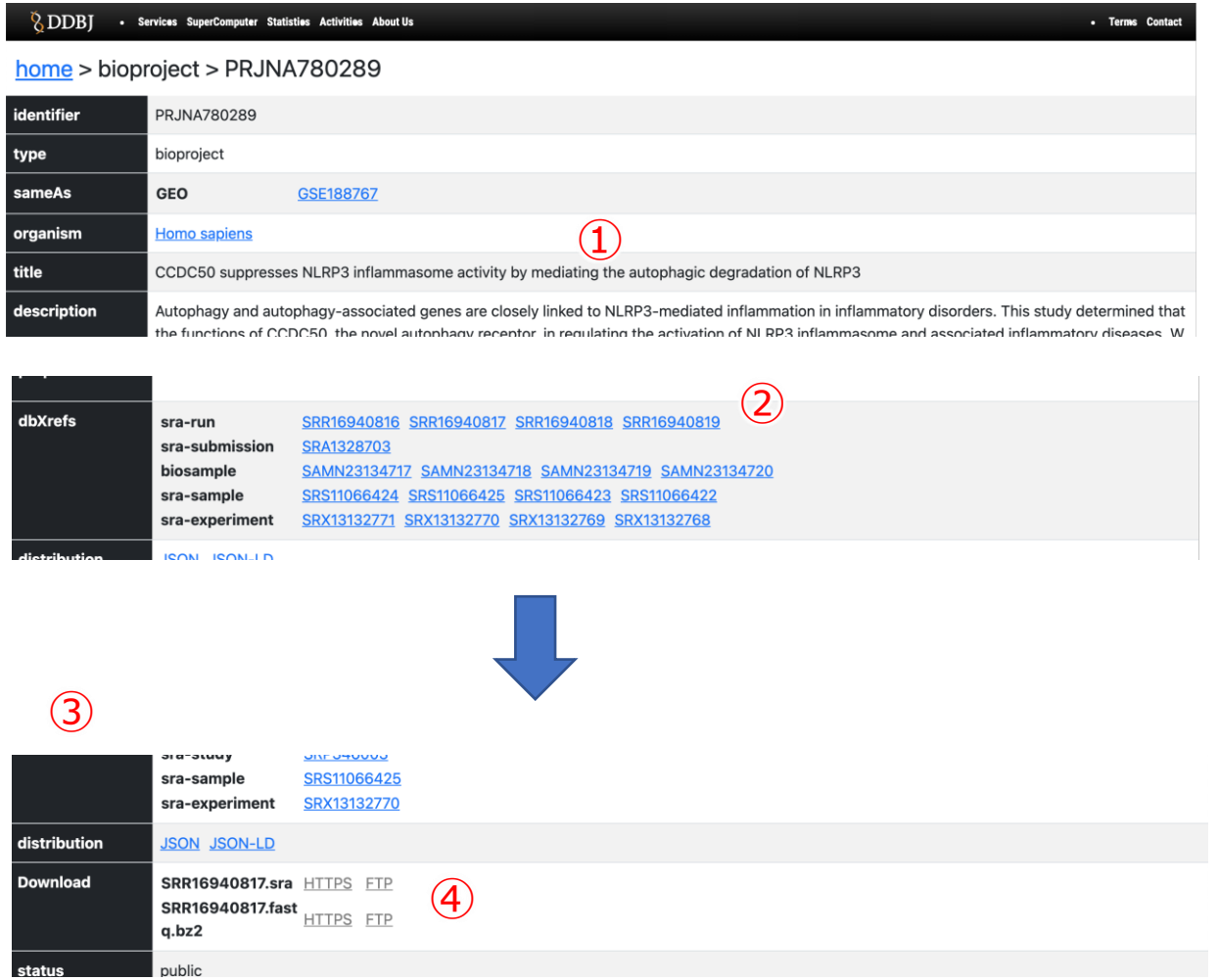
DDBJ Searchの検索画面



DDBJ SearchでのNGSデータの検索（詳細）（2）

DDBJ Searchにより公開NGSデータを検索できる。

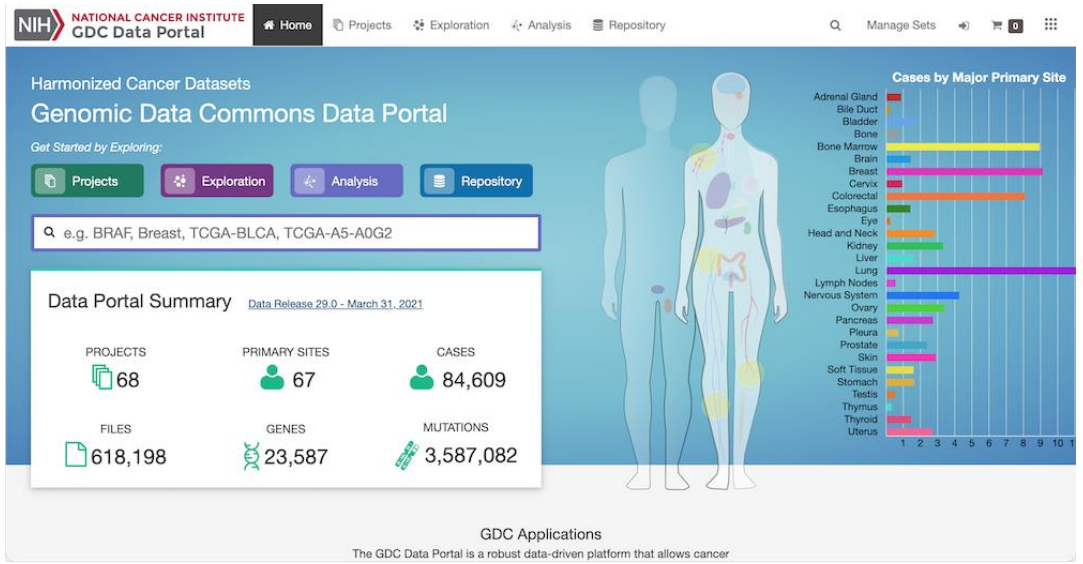
- DDBJ（DRA）での実際の検索手順（つづき）
  - ① 詳細画面ではプロジェクトの内容などが確認できる。
  - ② dbXrefs欄にある「SRR+数字」（もしくはERRかDRR+数字）を選択する。
  - ③ Runの画面に遷移する。
  - ④ Download欄にある.sraや.fastq.bz2のファイルをダウンロードできる。
    （まだファイルが用意されていなかったり、権利関係からダウンロードできないものもある）
  - ⑤ .sra ファイルはSRA Toolkit を用いて、.fastqファイルに変換する。
    また、SRA toolkitを通じて.sraファイルをダウンロードすることもできる。
    https://github.com/ncbi/sra-tools/wiki/



がん実験データ（TCGA）の活用

TCGA から公開されているがん実験データを活用することができる。

- **TCGA (The Cancer Genome Atlas) とは**
  - ✓ 米国国立がん研究所（NCI）と米国国立ヒトゲノム研究所（NHGRI）が中心となって行われたがんゲノム解析プログラム
    - ➢ プロジェクト： https://www.cancer.gov/about-nci/organization/ccg/research/structural-genomics/tcga
    - ➢ データベース： https://portal.gdc.cancer.gov/
  - ✓ 34の がん種、腫瘍組織と正常組織の合計20,000サンプル以上
  - ✓ TCGAはGDC(Genomic Data Commons)の主要なデータセット
  - ✓ ゲノム、エピゲノム、トランスクリプトーム、プロテオーム
- TCGA公開データでできること
  - ✓ がんのゲノム、エピゲノム、トランスクリプトームデータを検索・ダウンロードする
  - ✓ プロジェクトごとに関連遺伝子の重なりや生存曲線のコホートの比較を行う



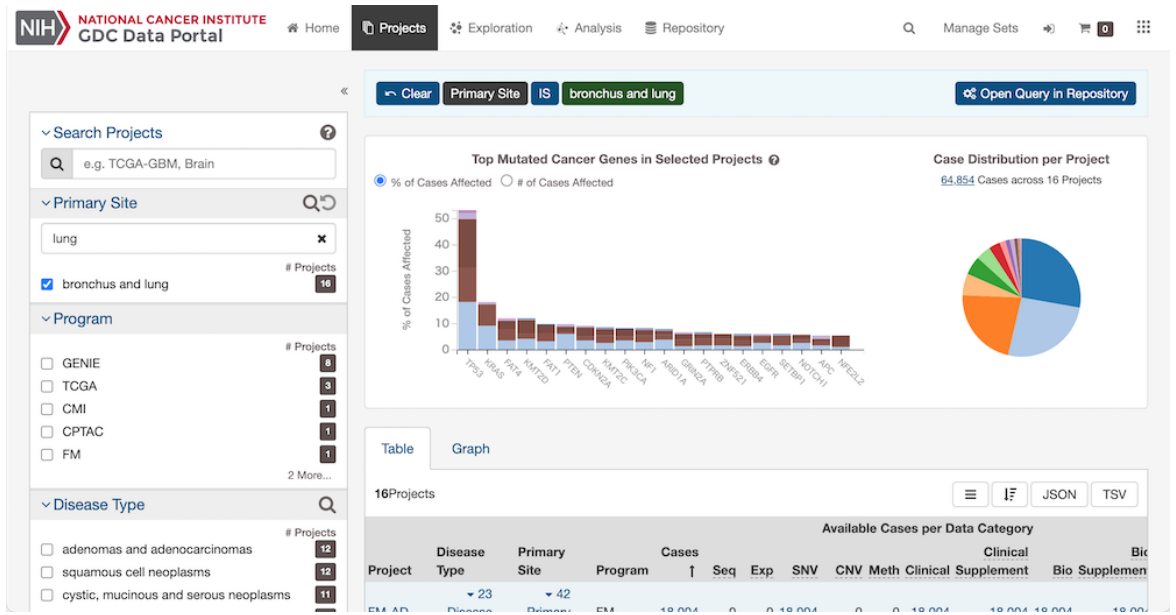
[参考]

- 統合TVによるチュートリアルムービー（ただし一部 UI が古い）
  - ✓ The Cancer Genome Atlas (TCGA) を使って各癌種の公開データを検索・ダウンロードする https://togotv.dbcls.jp/20171210.html
  - ✓ The Cancer Genome Atlas (TCGA) を使って各癌種の公開データを解析する https://togotv.dbcls.jp/20171214.html

TCGAの検索：実験データの検索（1）

TCGA でがん関連の実験データを検索したりデータの入手やウェブ上で解析する方法を説明する。

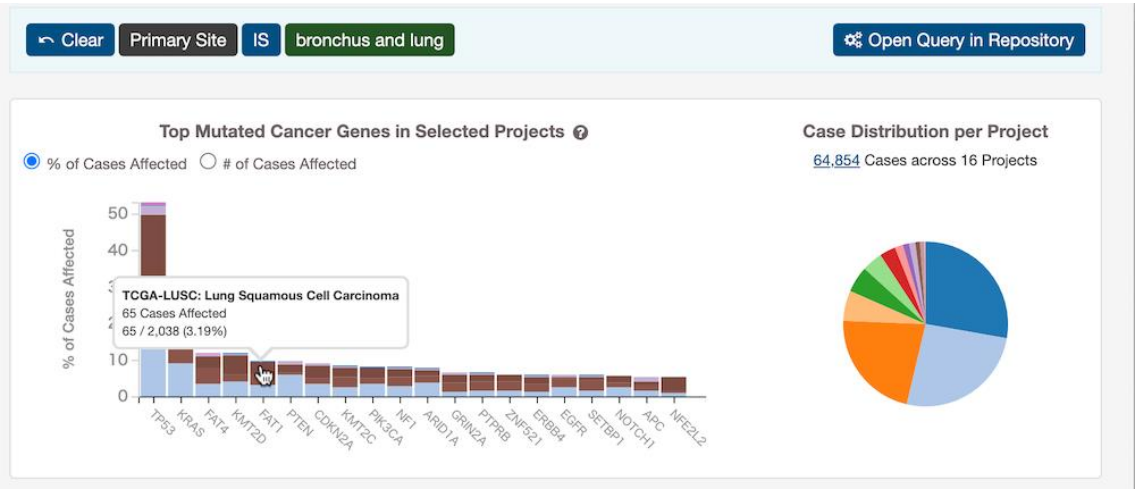
- プロジェクトのウェブサイトの場合は Access TCGA Data からデータベースサイトにアクセス
- （最初にサイトにアクセスするとデータの利用に関する Warning が出る）
- 今回は肺がんの実験データを検索する
- Projects をクリック（①）
- 左側で条件を絞り込んで、右側に結果の概要とリストが出る
- Primary Site で臓器の絞り込みができる。
  リスト中から探すか、虫眼鏡マークをクリックすると現れる検索窓からキーワード検索してもよい。
  今回は “bronchus and lung” にチェックを入れる
- 他に Disease Type などで絞り込みができる
- 右上に選択されたプロジェクトでの変異のあった遺伝子のリスト、右下には関連プロジェクトのリストが表示される（次ページ）



TCGAの検索：実験データの検索（2）

（続き）TCGA で実験データを検索する方法を説明する。

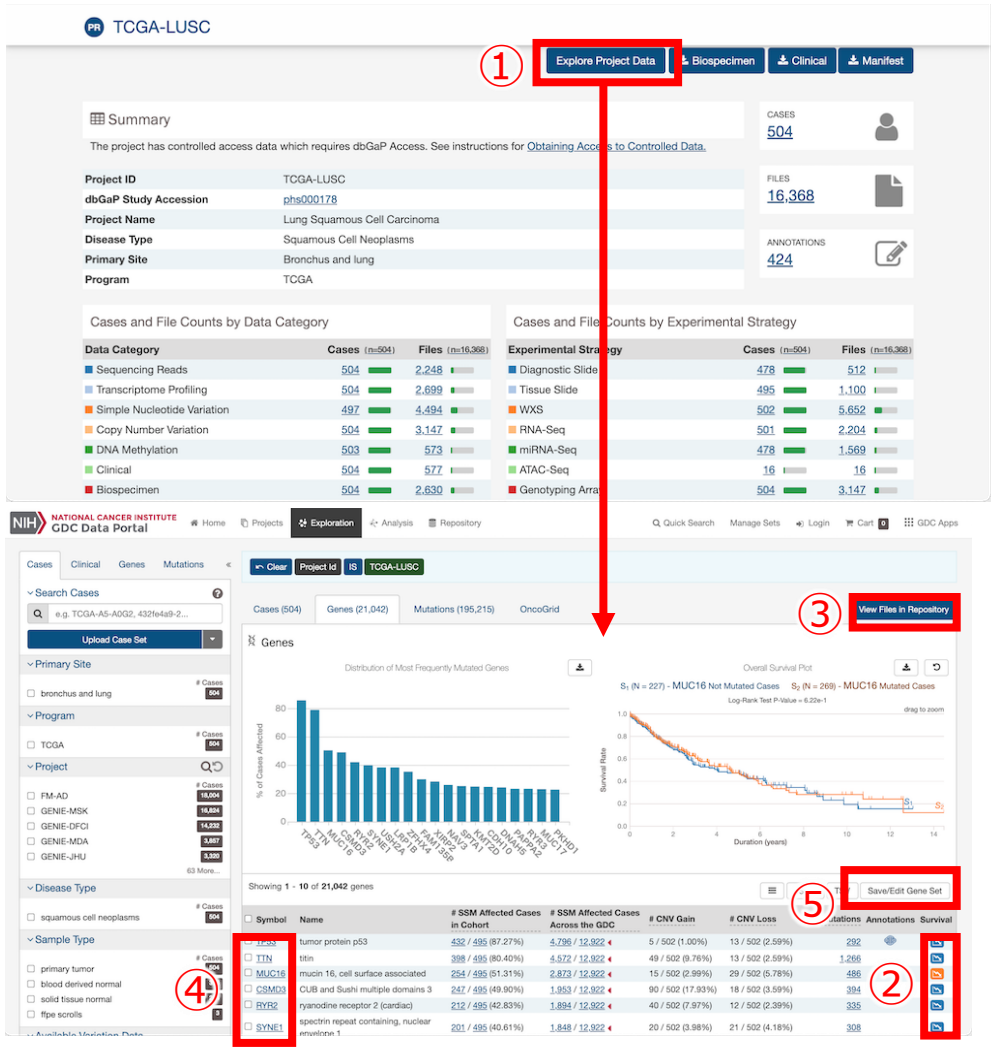
- 画面左側で条件を指定すると右側にプロジェクトのリストと、それらについての簡単な概要がグラフで表示される
- （右上）簡単な概要
  - ✓ 検索結果のプロジェクトで変異が多く見られた遺伝子ごとに、その変異を持つ症例の割合が棒グラフで表示される
  - ✓ マウスオーバーすると、当該部分のプロジェクトや症例数がバルーン表示される
  - ✓ 検索結果のプロジェクトでの患者数の割合が円グラフの形で表示される
- （右下）検索結果のプロジェクトリスト
  - ✓ Primary Siteや症例数、SNVの数などが一覧表示される
  - ✓ Disease TypeやPrimary Siteのデータをクリックすると、非表示になっていた詳細なデータが展開されて表示される
  - ✓ CasesやSNVのデータをクリックすると、個々の症例のデータの一覧表示にリンクされる
  - ✓ プロジェクト名などをクリックするとプロジェクトの詳細画面にリンクされる
    今回は、TCGA-LUSCをクリックする



TCGAの検索：プロジェクトの詳細

TCGA でプロジェクトの詳細を確認することができる。

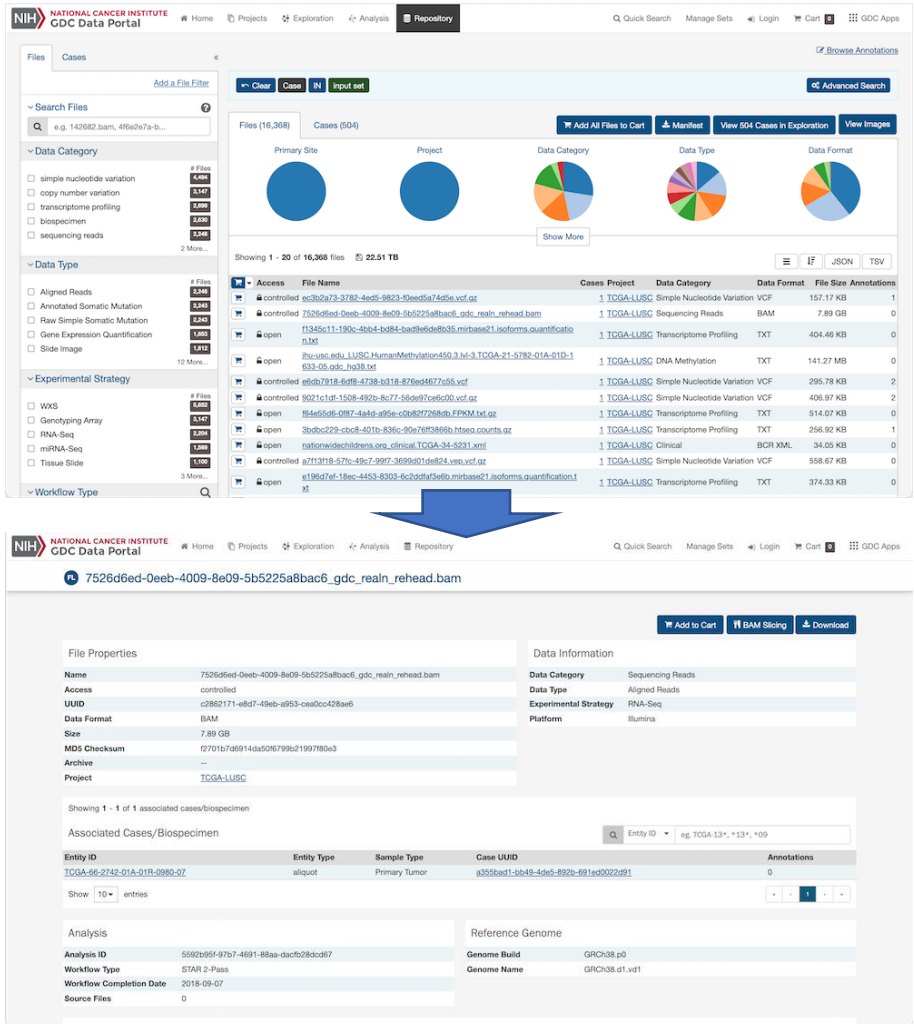
- プロジェクトの詳細画面
  - ✓ 当該プロジェクト下のデータの内容（遺伝子発現、SNV、CNVなど）や実験プラットフォームの数が表示される
- Explore Project Data ボタンをクリックすると（①）症例ごと、遺伝子ごと、変異ごとの統計情報が表示される
  - ✓ Geneタブを選択すると、変異が頻出する遺伝子リストとそれを可視化した棒グラフが表示されている。
  - ✓ また、右上には生存曲線が表示されている
  - ✓ リスト中の Survival のボタンをクリックすると（②）、その遺伝子の変異の有無での生存曲線の比較が行える。
- View File Repository をクリックすると（③）、データのダウンロードのためのファイル選択リスト画面が表示される（次ページ）
- 興味のある遺伝子や症例を選択し（④）、Save / Edit Gene Set（もしくは Case Set ）ボタンを押すことで（⑤）Gene Set / Case Setとして保存し解析することもできる
  （後述：TCGA の検索：症例/遺伝子の比較解析）



TCGAの検索：データのダウンロード

TCGA からデータをダウンロードすることができる。

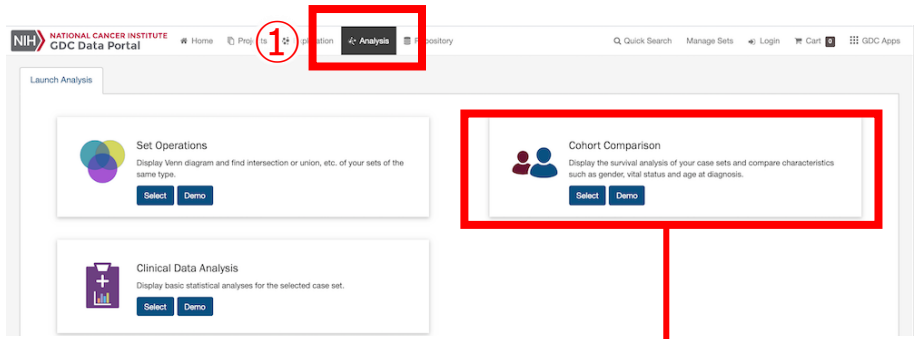
- データダウンロードのためのファイルリスト画面
- ファイルリストの絞り込み
  - ✓ 現状はすべてのファイルが表示されている状態なので画面左側のフィルターで絞り込みを行う。
    - ➢ Experimental Strategy で RNA-Seq
    - ➢ Data Format で bam や vcf を選択する　等
  - ✓ 選択すると、リストが即時に絞り込まれて結果が反映される
- ファイルリストからのダウンロードファイルの選択
  - ✓ ファイル名を見ると、bam や FPKM の文字が見えるので、中身が推定できる。
  - ✓ 通常 bamファイルは controlled access になっている
  - ✓ それでも、File Name をクリックしてファイルの詳細画面に行くと、Downstream Analysis Files として下流の解析（この場合、FPKM やcount データ）が（個人情報が丸められているので）open access データとしてダウンロードできるので、自分で解析することなく、結果を利用することができる。
  - ✓ Controlled access のファイルを利用したいときはデータ提供元である GDC に利用を電子申請することもできる。

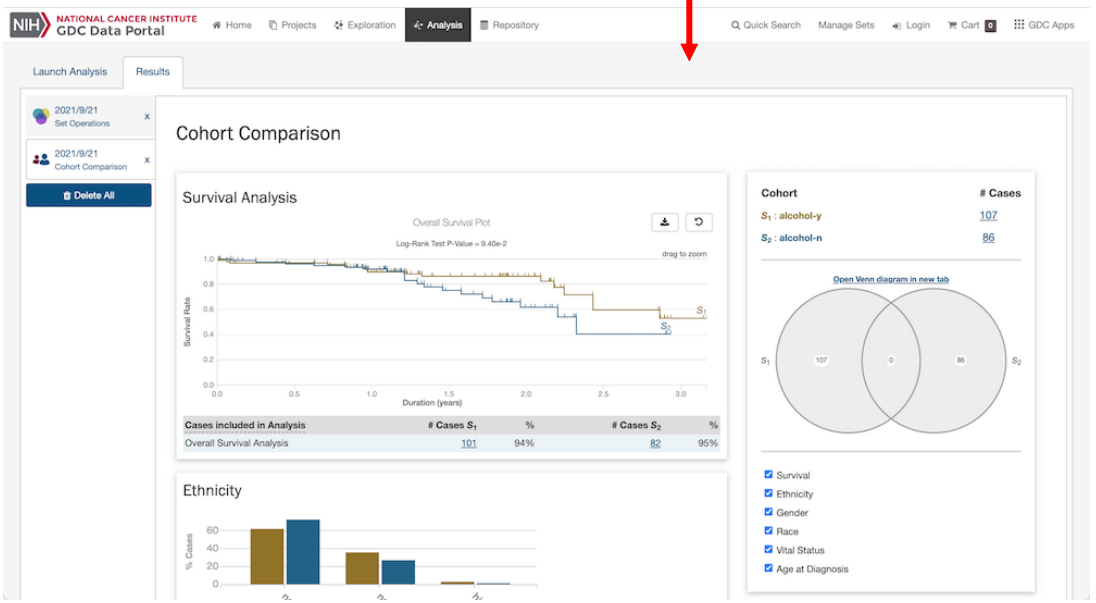


TCGAの検索：症例/遺伝子の比較解析

TCGA で症例/遺伝子の比較解析を行うことができる。

- Gene / Case Set を作成後、画面上部の Analysis をクリックすると（①）いくつかの比較解析が行える。
- 各解析で Demo があるので用意された症例等のセットを利用して試しに使ってみることもできる。
- Set Operations
  - ✓ 複数の Gene / Case Set に対して、含まれる遺伝子等の要素の重なりのベン図を表示する。
  - ✓ 共通部分や差分部分のリストも取得可能
  - ✓ セットは3つまで指定できる。
- Cohort Comparison（図）
  - ✓ 2つの症例のセットを選択して生存曲線のコホートを比較
  - ✓ その他、性別や生死なども棒グラフで比較可能
  - ✓ 今回はExploration画面での症例検索で、絞り込む際に画面左側で Cases タブでの絞り込みのほかに、Clinical タブで飲酒歴の有無で絞り込んで作成した症例セットの例を示している
- Clinical Data Analysis
  - ✓ 選択した症例セットについて、Gender や飲酒歴、喫煙歴などでのさらなる絞り込みや生存曲線の比較などが行える。

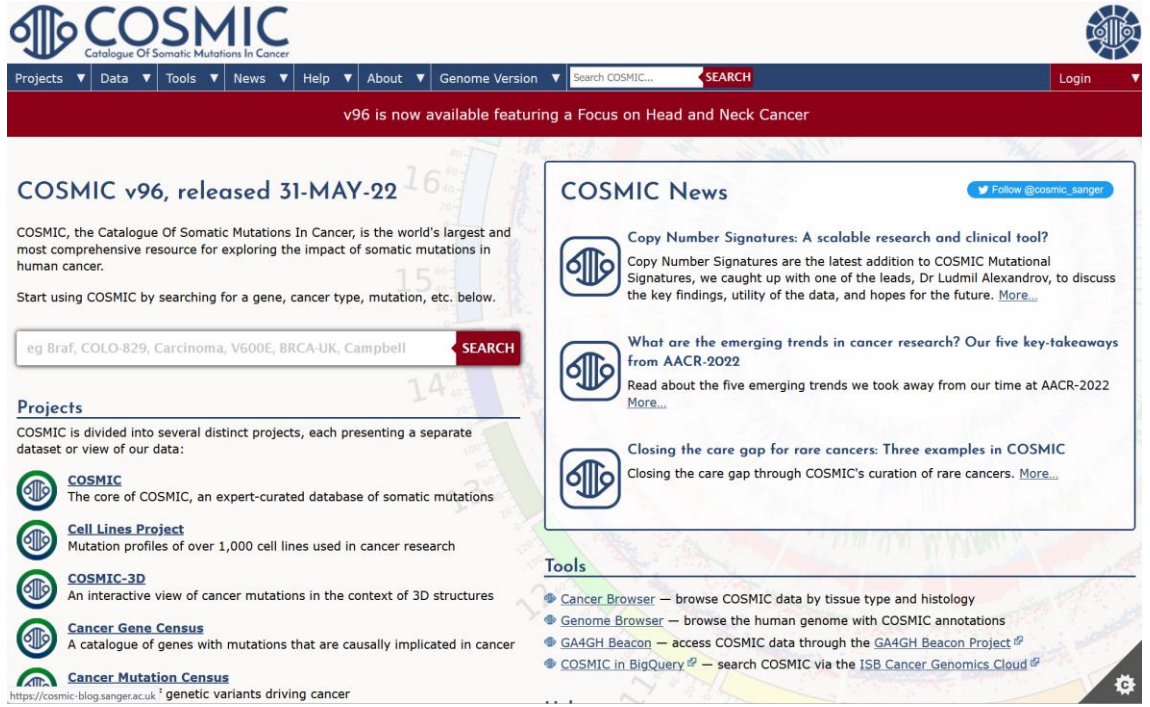




がん関連の知識DB（COSMIC）の活用

COSMIC からがん関連のナレッジデータ（既存の知見）を入手、活用することができる。

- **COSMIC（Catalogue Of Somatic Mutations In Cancer）とは**
  - ✓ がんにおける体細胞変異のカタログ
    - ➢ COSMIC：https://cancer.sanger.ac.uk/cosmic
- COSMIC でできること
  - ✓ 遺伝子の名称や変異、がんの部位で検索可能
  - ✓ 遺伝子の各種アノテーションについて閲覧可能

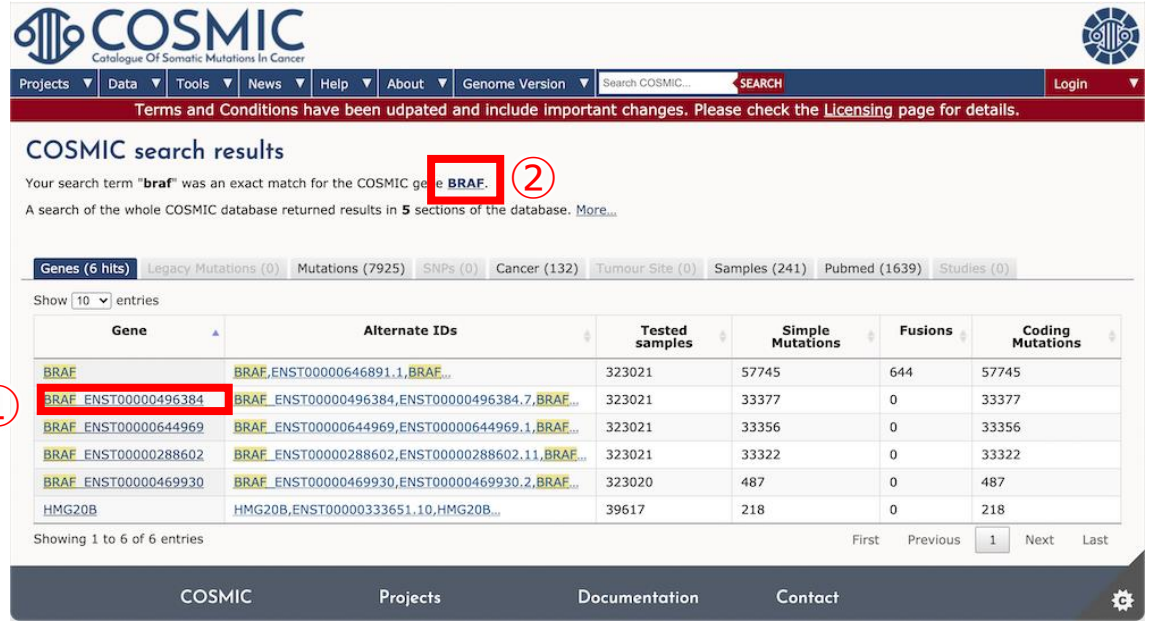


［参考］
- 統合TVによるチュートリアルムービー（ただし一部 UI が古い）
  - ✓ COSMIC でがん遺伝子の体細胞変異について調べる
    https://togotv.dbcls.jp/20180127.html

COSMICの検索

COSMIC からがん関連のナレッジデータ（既存の知見）を入手、活用することができる。

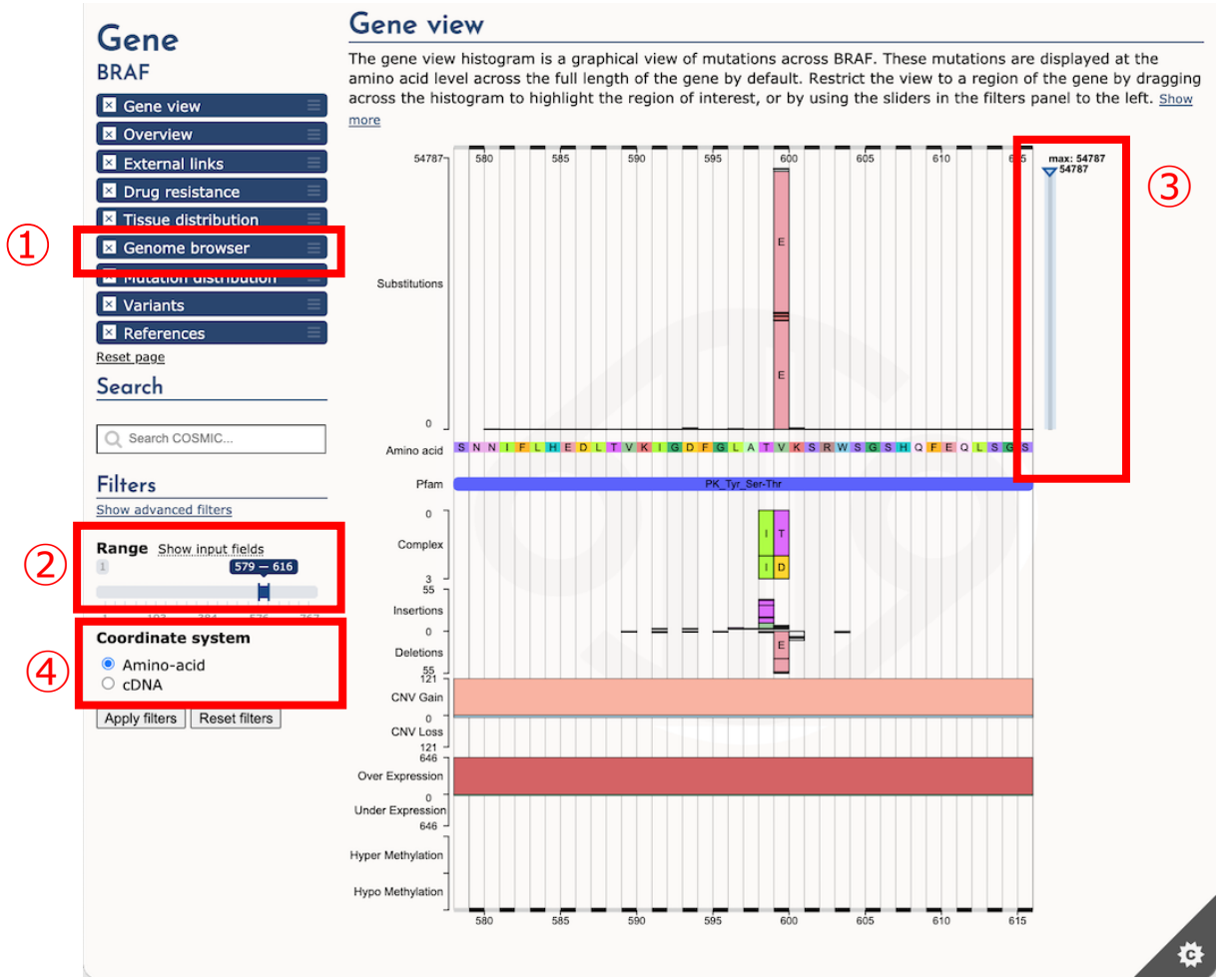
- COSMIC のサイトにアクセスする
  - ✓ シンプルに COSMIC と Google 検索すれば出てくるが、「COSMIC cancer」などと検索すると一番上に出てくる
- 今回は遺伝子から検索する例
- 検索フォームに *BRAF* と入力する
- いくつかの候補が出る
  - ✓ BRAF_ENST00000496384 など出るが（①）、これは Ensembl 由来のデータ。今回は *BRAF* を選択する
  - ✓ 基本的な情報の多くついているものを選べばよい
  - ✓ よく見ると、表の上に exact match として *BRAF* にヒットした旨の記述があるので、これをクリックしてもよい（②）



COSMICでの遺伝子情報：Gene view

COSMICのGene viewを使うと、遺伝子情報を確認できる。

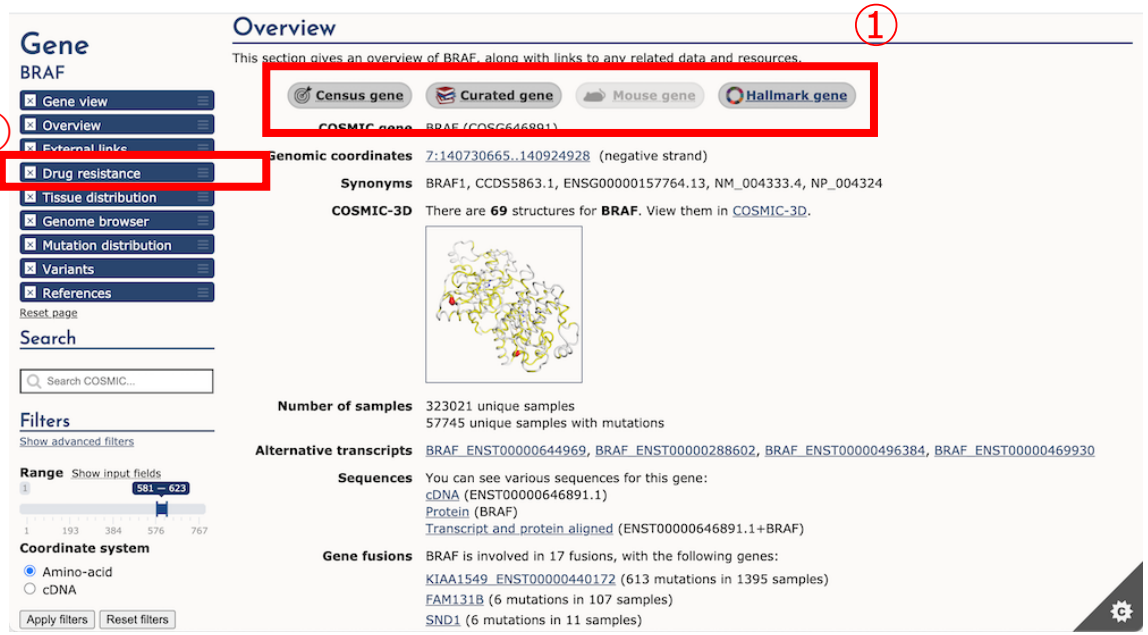
- Gene view
- 当該遺伝子のゲノムブラウザ表示
  - ✓ ゲノム全体の領域でのゲノムブラウザ表示も下の方にある（後述）（①）
- 塩基置換、INDEL 、CNV 、メチル化、タンパク質ドメインなどが表示される
- 領域をドラッグすることでその部分を拡大表示できる
  - ✓ 画面左側のFilters内で、Range が当該領域に切り替わる（②）
  - ✓ この Range を変更（その後、Apply Filters をクリック）することにより描画領域を変更してもよい（拡大しすぎた際などの領域のズームアウトにもこの方法を用いる）
  - ✓ 拡大すると塩基置換の詳細が描画されるようになる
  - ✓ 棒グラフにマウスオーバーすると、各置換内容など（何から何に置換されたか、件数）が表示される
  - ✓ 棒グラフの高さが低くて見えない場合は、表示の最大値のスライダ（③）を動かすことでグラフの高さを変更することができる
- Filters 内の Coordinate system で Amino-acid と cDNA を切り替えることで（④）右側の表示をアミノ酸配列表示か塩基配列表示か変更することができる



COSMICでの遺伝子情報：Overview

COSMIC の Overview を使うと、遺伝子のアノテーション情報を確認できる。

- Overview
- 遺伝子のさまざまなアノテーション情報が表示される
- がん遺伝子に関するタグ（①）により既知のがん遺伝子か確認できる
  - ✓ Census gene：Cancer Gene Census に登録されたがん遺伝子
  - ✓ Curated gene：COSMIC のキュレーターによる確認済のがん遺伝子
  - ✓ Mouse gene：マウス実験でがん化が確認されたがん遺伝子
  - ✓ Hallmark gene：D. Hanahan らの論文で定義されたがん特性のある遺伝子*
- Gene fusions でがんが見つかった融合遺伝子を確認できる
- Drug sensitivity data で変異により感受性が変わる薬剤が確認できる
  - ✓ 別セクションで Drug resistance の欄もある（②）
    この遺伝子に変異を含む腫瘍の治療に使われたデータのある薬剤がリストされる

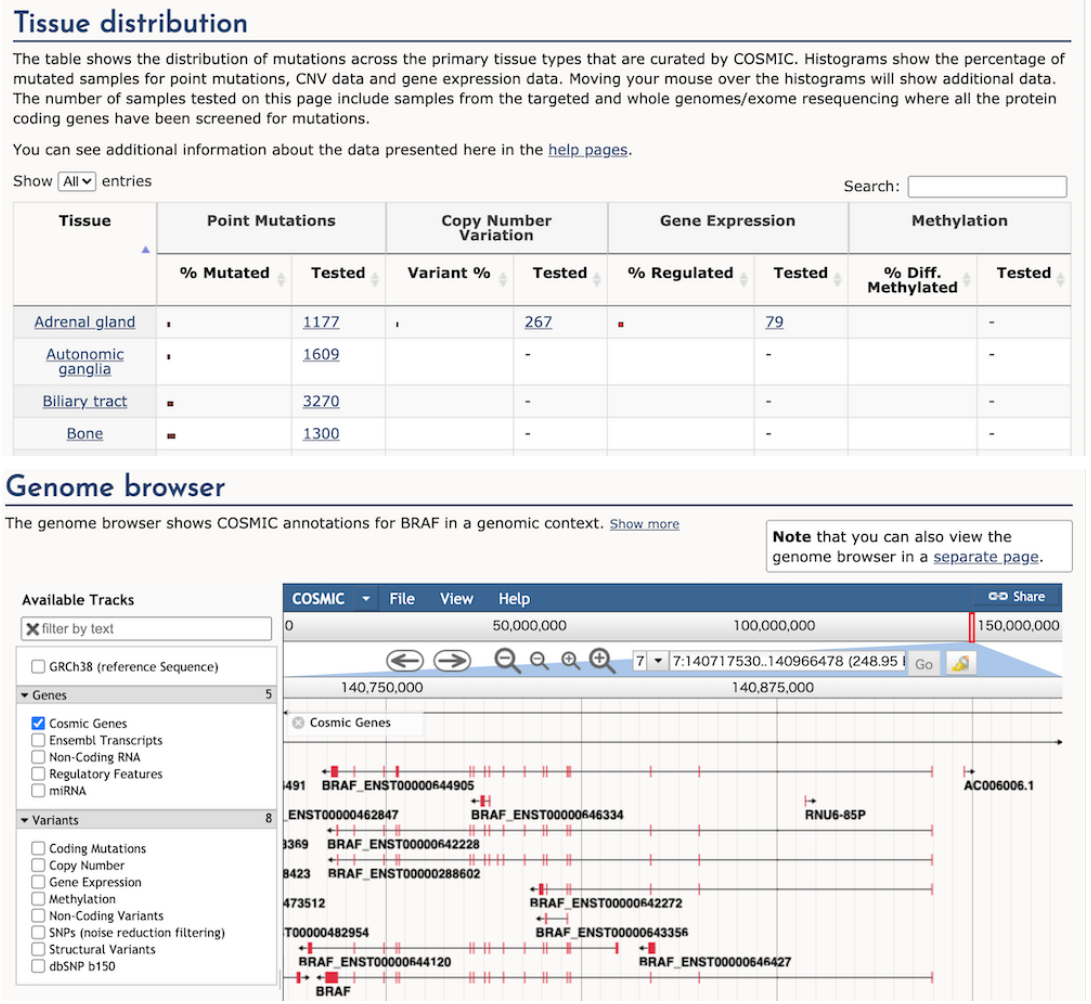


* Hallmarks of cancer: the next generation.
Hanahan D, Weinberg RA.
Cell. 2011 Mar 4;144(5):646-74. doi: 10.1016/j.cell.2011.02.013.
PMID: 21376230

COSMICでの遺伝子情報：Tissue distribution/Genome browser

COSMIC には他にも Tissue distribution / Genome browser 等の機能がある。

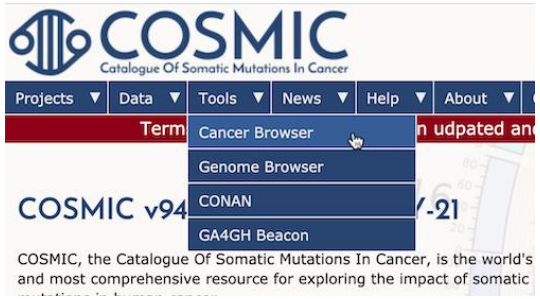
- Tissue distribution
  - ✓ 各組織におけるサンプルタイプごとの割合が表示される
  - ✓ 棒グラフが長いほど、今回ならば *BRAF* の遺伝子変異が見られたサンプル数が多いことを示している
  - ✓ 表内の値をクリックすると、各サンプルの点変異や CNV のデータが見られる。TSV や CSV でデータのダウンロードができるがログインが必要
- Genome browser
  - ✓ （この遺伝子に限らずゲノム全体での）ゲノムブラウザ表示
  - ✓ ゲノム上での他の遺伝子の位置などがゲノムブラウザ上で確認できる
  - ✓ Available Tracks 欄を設定することにより、さまざまな遺伝子情報や変異の情報をトラックとして画面上に追加することができる。

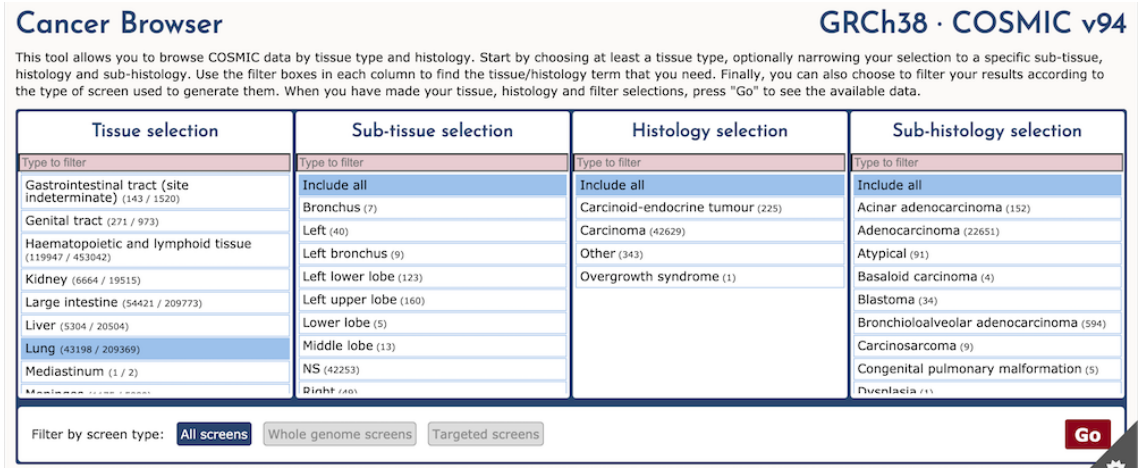


COSMIC：Cancer Browser

COSMIC の Cancer Browser で組織や腫瘍ごとの変異などの情報を確認できる。

- COSMIC のトップ画面で Tools > Cancer Browser
- 組織などで表示するデータの絞り込み条件を選択できる
  - ✓ Tissue selections 以外は、全件を表示する場合は Include all を選択する
  - ✓ リストから選ばずに自分でキーワードを入力して用語を探すこともできる
- Go ボタンをクリックすると下に結果が表示される（次ページ）

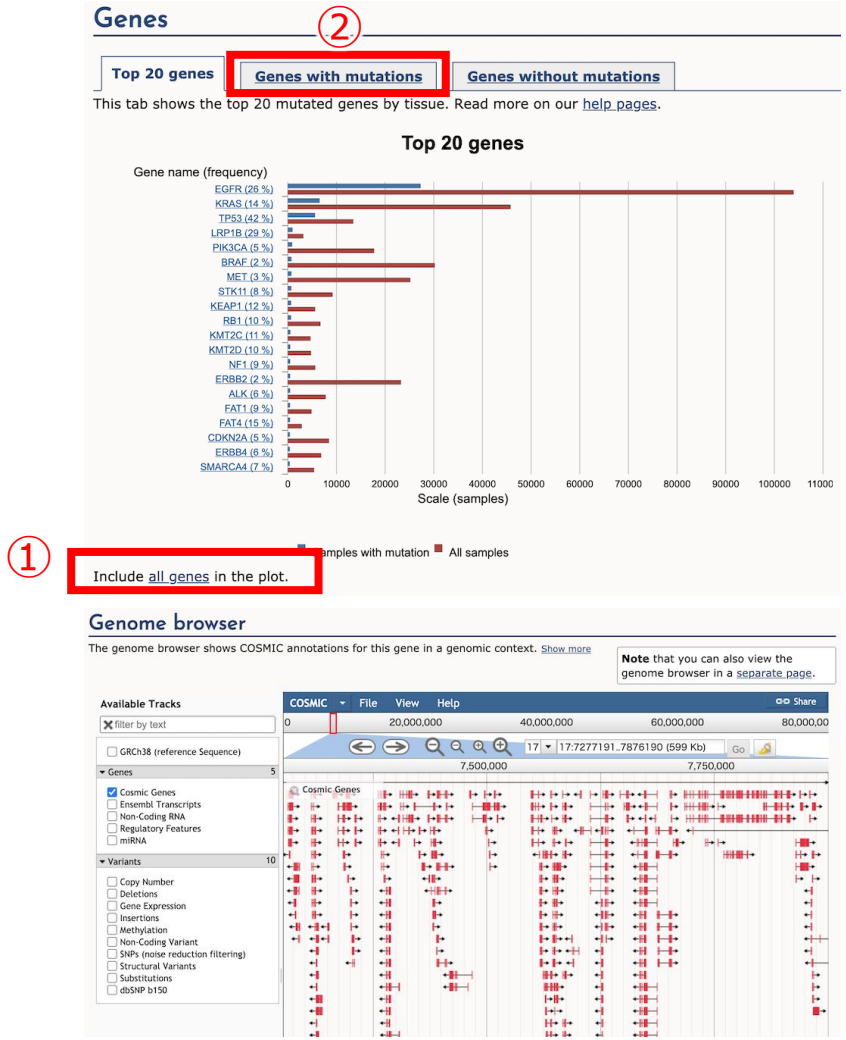




COSMIC：Cancer Browser結果

COSMIC の Cancer Browser で組織や腫瘍ごとの変異などの情報を確認できる。

- Genes
  - ✓ 選択した組織の腫瘍で変異が頻出する top 20 遺伝子について サンプルの割合を棒グラフで表す
  - ✓ デフォルトでは census 遺伝子のみだが、全遺伝子を対象にすることも可能（①）
  - ✓ タブを切り替えると変異のあった遺伝子リストも表示できる（②）
- Genome browser
  - ✓ この腫瘍で報告された変異をゲノムブラウザ上で確認できる
- その他
  - ✓ Mutation matrix：このがんで変異の多い遺伝子とサンプルの相関
  - ✓ Distribution：変異の分布（変異タイプや何塩基の indel かなど）



第 Ⅲ 章
データ解析基礎
（fastqファイルからはじめる点変異、構造変異、コピー数異常の探索とVCF作成）

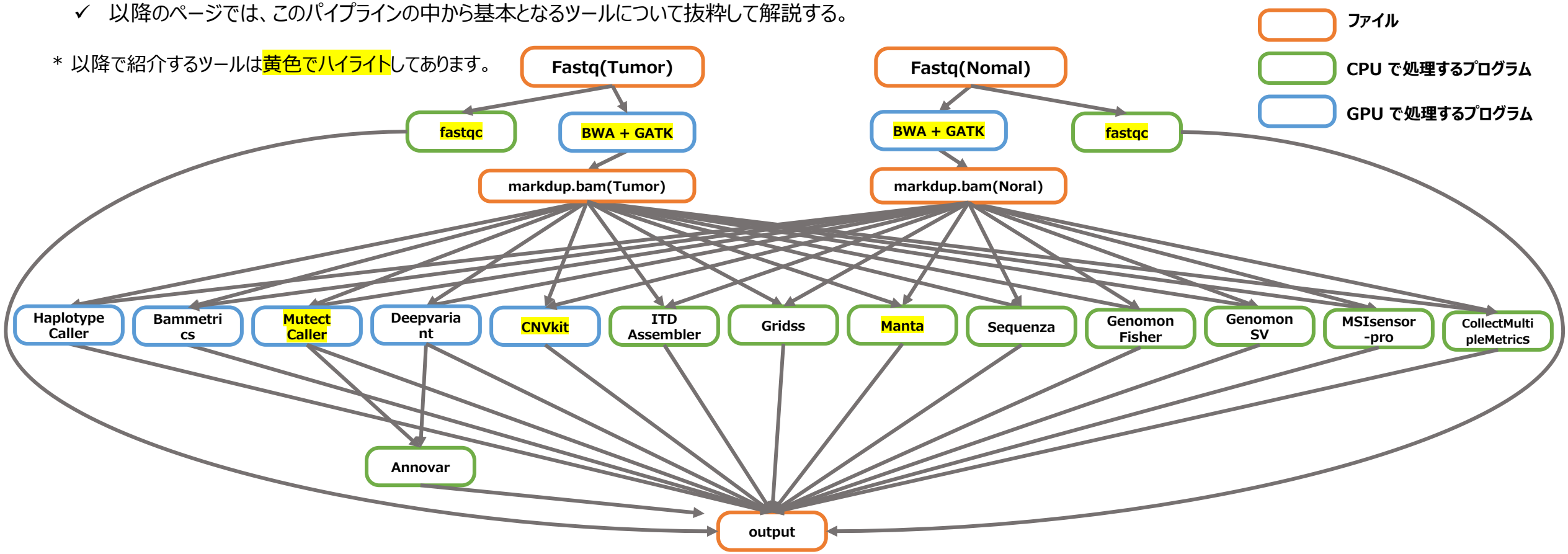
東京大学医科学研究所　ヒトゲノム解析センター　シークエンスデータ情報処理分野　准教授
片山　琴絵
愛知県がんセンター研究所システム解析学分野　分野長
山口　類
東京大学医科学研究所　ヒトゲノム解析センター　健康医療インテリジェンス分野　教授
井元　清哉

がんのデータ解析のパイプライン

パイプラインとは複数のツールを組合せた一連のワークフロー。各工程でどのツールを選ぶかは解析目的や好みなどによって異なる。

- 国家プロジェクトである令和3年度「革新的がん医療実用化研究事業」では、がんゲノム医療に資する情報解析基盤を構築するミッション遂行のため、fastqファイルから様々な解析を一連のワークフローとして実行するパイプラインが構築され、使用されている。https://www.hgc.jp/~kks_th/tmp/aO86CfPJcy6TogQW/docs/index.html#
  - ✓ 以降のページでは、このパイプラインの中から基本となるツールについて抜粋して解説する。

＊ 以降で紹介するツールは黄色でハイライトしてあります。



解析システム：計算機環境

全ゲノムシークエンスデータを解析するために必要な計算機環境

全ゲノムシークエンスデータはデータそのものが大きい。30XのWGSのfastqファイルは圧縮された状態でR1,R2合わせて60GB程度。また計算に必要なメモリ量も多くなる。このためノートパソコンや、デスクトップパソコンなど通常の計算機では扱うのはほぼ不可能

- スーパーコンピュータ（オンプレミス）
  - ✓ 東京大学医科学研究所ヒトゲノム解析センター：SHIROKANE
  - ✓ 国立遺伝学研究所：遺伝研スーパーコンピュータシステム
- クラウド
  - ✓ Amazon AWS
  - ✓ Google Cloud
  - ✓ Microsoft Azure
  - ✓ Illmina DRAGEN

東大医科研 HGC スーパーコンピュータ SHIROKANE

生命・医学系専用の運用をしている
国内最大規模のスパコンです

2.0 Peta FLOPS

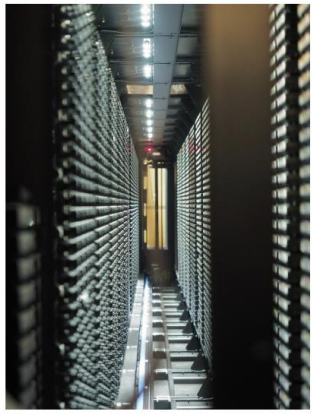
http://www.hgc.jp/~ayumu/705/P4207532.jpg
SHIROKANE を、主力の計算機「計算ノード Thin」側から見たものShirokane5 と Shirokane6 の理論演算性能は計 2.0 Peta FLOPS
GPU NVIDIA V100, A100
約 2 万の CPUコア

24 PB高速ﾃﾞｨｽｸｱﾚｲ

http://www.hgc.jp/~ayumu/705/P4207534.jpg
Shirokane を、「共有ディスク」側から見たもの。全ての計算ノードから同時に高速にアクセスできるディスクが 24 ペタバイトある

巨大ストレージ

max100PBｱｰｶｲﾌﾞ



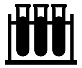
IBM社製テープアーカイブ
＋ 1 PB ニアラインディスク

データについて

NGS の大まかな流れとゲノム解析で取り扱う様々なデータファイル

- **fastqファイル**
  - ✓ シークエンサーから出力される結果ファイル（WGS の場合巨大なテキストファイル）。
  - ✓ バイオインフォマティクスで解析のスタートとなる基本の入力ファイル。
  - ✓ シークエンサーで読み取られた塩基配列（シークエンス）情報と付随する塩基ごとの計測クオリティの情報で構成。
  - ✓ シークエンサーで読み取られた塩基配列データを、リード（read）データと呼ぶ。
  - ✓ 各リードは150bp (塩基対)程度の長さが典型的。
  - ✓ リードデータを見ただけでは、それがゲノム上のどこから得られたものなのかわからない。
    - ➢ データ解析（アライメント / マッピング）が必要。
  - ✓ 論文投稿の際には公開データベースへの登録を求められることが多い。
  - ✓ 日本ではDDBJが運営するDRAデータベースなどが主な登録先（詳しくは入門編第 Ⅶ 章）
- **bamファイル**
  - ✓ fastqファイルに含まれる各リードデータを標準ゲノム配列（リファレンス配列：GRCh38 等）の対応する場所にアライメントすることによって得られるファイル
  - ✓ 後述する IGV に入力ファイルとして与えることで可視化も可能
- **vcfファイル**
  - ✓ 変異検出（コール）を行った結果得られるファイル

    例：1番染色体の 33,329,422番目の塩基が標準配列では**T**だが、
    患者Xさんの、がん細胞の DNA では **C** に変わっている、等の情報が記載されている。
  - ✓ TCGA など公開データベースで公表されることもある。



検体

シークエンサー

イルミナ社 NovaSeq 6000　等

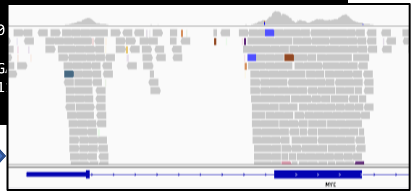
fastqファイル （リードデータ）

```
@HWI-D00672:79:C976TANXX:2:1101:1485:2078 1:N:0:ATAGCGAC
CNAAGGGGAAATGAATCCCTGCTTGTGTTAGTTCACTAAGCAGAGTAAAGAAGAAAGTCAAAACAAAATGAAGCCAGTCAACCAACCATAGAGTGTGGGTT
+
A#<<:F0E/F>FDCDG1GGGGGGGCDGGGGGGC1>GGGGFDF:1EGGGGGGGGCFGGF@G@GGGBGFGCDGCBFGGBGCFCGGGGG>BC>D>G;;CFG>EE
@HWI-D00672:79:C976TANXX:2:1101:1442:2148 1:N:0:ATAGCGAC
TGAAGCAAGTTAATTACTTACAGATAGGCAGCAAGAAACAAAACAAAAGAAGCCTGTGATCCATTGTGAGTAGGTTCCCCAAGGTTCGAGAAAGCTGCCTC
+
CCCCCGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGEFFCDGGGGGGFEGGGGG
…
```

bamファイル (リードのアライメント情報)

```
HWI-D00672:79:C976TANXX:2:1214:19608:24880      147     1       10001   1       101M    =       10042   -60
TAACCCTAGCCCTAACCCCAACCCTCACCCCAACCCTAACCCTAACCCTAGCCCTAACCCTAACCCTAACCCTAGCCCTAACCCTAACCCTAACCCTAACC   .80<0=
000;0=<.;<.;/0/FB?0/9E<//11GF>01=GC>01E01:<11C1>011:FCE>1FF?0110FE>11GE1=1:0E=01;@F>01CB>1?:3:3  NM:i:6
MD:Z:8A9T6A4T19A23A26   AS:i:71 XS:i:68
HWI-D00672:79:C976TANXX:2:1106:21143:8866       145     1       10003   0
0
AACCCTCCCCCTCACCCTATCCCTAACCCTAACACTAACCCCAACCCAAACCCAAACCCTAACCCTACCCCCG
....:...88/....../.;//0/9E000//:////;9/<///9///9FE0?0:>E/0E9</<<//////<111
MD:Z:6A13C7T5T5T13      AS:i:29 XS:i:29 SA:Z:20,62918125,+,24M7 0,0;
```

可視化



vcfファイル（検出された変異の情報；位置や変異パターン、depth など）

```
#CHROM  POS     ID      REF     ALT     QUAL    FILTER  INFO    FORMAT  data
1       33329422        .       T       C       100     PASS
DP=83;AC=17;AF=0.205;AN=66;VF_Normal=0.000;depth=83;depth_tumor=83;variantNum_tumor=17;depth_normal=103;varian
tNum_normal=0;bases_tumor=31,6,52,11;bases_normal=49,0,54,0;A_C_G_T_tumor=0,17,0,66;A_C_G_T_normal=1,0,0,102;m
isRate_tumor=0.205;strandRatio_tumor=0.353;misRate_normal=0.000;strandRatio_normal=---;P-
value(fisher)=6.352;readPairNum_tumor=66;variantPairNum_tumor=17;otherPairNum_tumor=0;readPairNum_normal=101;v
ariantPairNum_normal=0;otherPairNum_normal=1;P-
value(fisher_realignment)=6.311;indel_mismatch_count=0;indel_mismatch_rate=0;bp_mismatch_count=0;distance_from
_breakpoint=0;simple_repeat_pos=---;simple_repeat_seq=---       GT:GQ:DP:AD:VF  0/1:63.52:83:66,17:0.205
```

データについて：fastqファイル

シークエンサーから出力される。バイオインフォマティクス解析のスタートとなる基本ファイル

- **シークエンスの方法により異なるfastqファイルの数**
  - ✓ シングルエンドシークエンスの場合はシークエンサーから出力されるファイルの数は1つ。
  - ✓ ペアエンドシークエンスの場合は、出力されるファイルの数は2つで、リード1であるかリード2であるかの判別がつくファイル名になっている場合が多い。

    例：サンプルA：SampleA_R1.fastq　SampleA_R2.fastq
    　　サンプルB：SampleB_1.fq　　　SampleB_2.fq
  - ✓ 近年のNGS解析ではペアエンドシークエンスである場合が多い。
  - ✓ ヒトWGS30X の場合で1ファイル25～30GB（.gzで圧縮されている場合）
  - ✓ ペアエンドシークエンスの場合、出力される2つのファイルの行数は必ず同じであることに注意する。
    - ➢ 行数が異なる場合、bamファイル作成を行うことができない。

4行で1リード

```
@HWI-D00672:79:C976TANXX:2:1101:1485:2078 1:N:0:ATAGCGAC
CNAAGGGGAAATGAATCCCTGCTTGTGTTAGTTCACTAAGCAGAGTAAAGAAGAAAGTCAAAACAAAATGAAGCCAGTCAACCAACCATAGAGTGTGGGTT
+
A#<<:F0E/F>FDCDG1GGGGGGGCDGGGGGGC1>GGGGFDF:1EGGGGGGGGCFGGF@G@GGGBGFGCDGCBFGGBGCFCGGGGG>BC>D>G;;CFG>EE
@HWI-D00672:79:C976TANXX:2:1101:1442:2148 1:N:0:ATAGCGAC
TGAAGCAAGTTAATTACTTACAGATAGGCAGCAAGAAACAAAACAAAAGAAGCCTGTGATCCATTGTGAGTAGGTTCCCCAAGGTTCGAGAAAGCTGCCTC
+
CCCCCGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGEFFCDGGGGGGFEGGGGG
@HWI-D00672:79:C976TANXX:2:1101:1439:2171 1:N:0:ATAGCGAC
AGTTAAGCTTTGGCCACTTGGTTTCAGTTAAAAGGATTGTGTGGAAGGAAGGATAAAAGGCTCTTGCACGCAGACCAGCTATTGCGTCTGAGGACTAAGAA
+
CCCCCGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGFGGGGCGGGGGGGGGGGGGF
```

1行目：リード名
2行目：塩基配列
3行目：記号
4行目：クオリティスコア
（2行目と文字数同じ）

- **fastqファイルの中身**
  - ✓ 1つのリードの情報は4行で書かれる。

    1行目：@から始まる。リード名と付加情報（フローセルIDやインデックス配列など）

    2行目：塩基配列

    3行目：「+」記号。場合によって「+」のうしろに付加情報

    4行目：各塩基に対するクオリティスコア（Qスコア、Phred クオリティスコアとも呼ばれる）
  - ✓ 2行目の文字数と4行目の文字数は同じ。
  - ✓ 4行目クオリティスコア：各塩基のベースコールの確からしさを示す指標。数値や記号1文字で表す*。なお、ベースコールとは読み取られたリードのヌクレオチド（ATGC）を識別すること。Q30以上を一般的には信頼することが多い。

    Q10：90%の確率でそのベースコールは確からしい

    Q20：99%の確率でそのベースコールは確からしい

    Q30：99.9%の確率でそのベースコールは確からしい

    Q40：99.99%の確率でそのベースコールは確からしい

＊ 参考URL：
https://support.illumina.com/help/BaseSpace_OLH_009008/Content/Source/Informatics/BS/QualityScoreEncoding_swBS.htm

データについて：bamファイル（1）

BAM(Binary Alignment Mapfastq)は、fastqファイルをリファレンス配列にアライメントすることで得られるファイル形式

- **bamファイルは独自の形式で圧縮されている。**
  - ✓ Linuxコマンドの `cat` , `less` , `head` などでは開けない。
  - ✓ メモ帳などでも開けない。
  - ✓ bamファイルの中身を見るためには `samtools` コマンドを使う。
- **bamファイルと同じ情報を持つ圧縮前のファイルはsamファイル**
  - ✓ samファイルはテキストファイルだがbamファイルの4~5倍の容量
- **samファイルは「ヘッダー」部分と「アライメント」部分で構成される。**
  - ✓ ヘッダー部分は@から始まる。
    @HD：ファイル1行目に記載されるバージョン情報など
    @SQ：リファレンス配列の情報
    @RG：リードグループの情報。ライブラリやシークエンサーの情報
    @PG：bamファイル作成時に実行されたプログラムの内容

ヘッダー部分

```
[gc067hgc0951:output$] samtools view -h sample_out.bam |less
@HD     VN:1.5  SO:coordinate
@SQ     SN:chr1 LN:248956422
@SQ     SN:chr2 LN:242193529
@SQ     SN:chr3 LN:198295559
@SQ     SN:chr4 LN:190214555
@SQ     SN:chr5 LN:181538259
@SQ     SN:chr6 LN:170805979
 <中略>
@RG     ID:HK3TJBCX2.1  LB:lib1 PL:bar  SM:sample       PU:HK3TJBCX2.1
@PG     ID:pbrun_fq2bam PN:pbrun_fq2bam VN:v3.0.0.6     CL:pbrun_fq2bam --tmp-dir /work --re
```

✓ アライメント部分 ：1行に1リードの情報

タブ区切りで左から順に以下基本11項目＋α(付加情報)で構成される。

①QNAME（リード名）
②FLAG（フラグ）
③RNAME（染色体名）
④POS（リードのスタートポジション）
⑤MAPQ（マッピングクオリティスコア）
⑥CIGAR（マッピング状況を表す文字）
⑦RNEXT（ペアエンドの場合、もう一方のリードの染色体名）
⑧PNEXT（ペアエンドの場合、もう一方のリードのポジション）
⑨TLEN（ペアエンドの場合、インサートの長さ）
⑩SEQ（リードの塩基配列）
⑪QUAL（リードのクオリティスコア）

詳しい情報： https://samtools.github.io/hts-specs/SAMv1.pdf

アライメント部分 実際は1行で1つのアライメントを表し、タブ区切りとなっている

```
  [gc067hgc0951:output$] samtools view sample_out.bam |head
①HWI-D00127:570:HK3TJBCX2:1:1202:9643:76055 ②99 ③chr1
④10027 ⑤26 ⑥24M5I86M ⑦= ⑧10178 ⑨231
⑩CCCTAACCCTAACCCTAACCCGACCCCGACCCCGACCCAAACCCAAACCCTAACCCT
  AACCCTAACCCTAACCCTAACCCTAACCCTACCCTAACCCTAACCCTAACCCTAACC
⑪DDDDDHGHIIIIIHIIHHIHHHIHIIIIIIHDHHIHHHIHIHIIIIFHIEHHIIHHI
  IIIEHIIIIHHIHIIICHE@1FHH?1GEFE1111D11<FH11<FD11<<FFE111<11
以降、付加情報↓
  MD:Z:22T5T0A4T5T41A27    PG:Z:MarkDuplicates
  RG:Z:HK3TJBCX2.1          NM:i:11 AS:i:69 XS:i:72
```

※⑦が“＝”の場合、もう一方のリードの染色体名も③と同じことを意味する

データについて：bamファイル（2）

bamファイルはツールを用いて可視化できる。

- **bamファイルを可視化する場合に使う代表的なツールは IGV**
  無料でダウンロードして使うことができる。

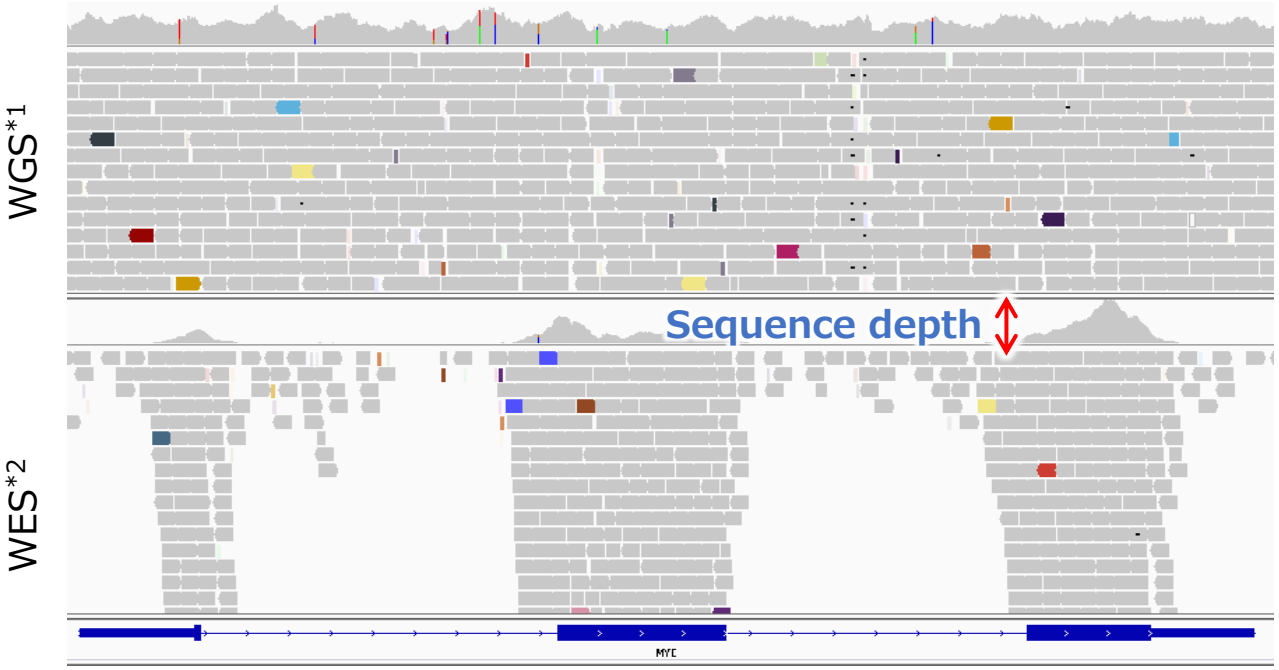
ダウンロードサイト：https://software.broadinstitute.org/software/igv/

- ✓ IGVで可視化するときには bamファイルに加えてインデックスファイルと呼ばれる baiファイル が必要。これは IGVなどのツールで大量のデータを素早く扱うための辞書のようなもの。bamファイルを作成した後、baiファイルをセットで作成することが多い。
- ✓ bamファイルを可視化する目的
  - ➢ 変異コールの確認：

    変異としてコールされているかどうかを確認する。見るべき項目としては周辺リードの情報や、アライメント状況、シークエンスエラーの有無など。
  - ➢ Hotspotの確認：

    変異コールの結果としては挙がってこなかったが、よく知られた Hotspot などでの状況を確認する。

IGVの画面



*1 WGS（Whole Genome Sequence, 全ゲノムシークエンス）
*2 WES（Whole Exome Sequence, 全エクソームシークエンス）

データについて：vcfファイル

VCF（Variant Call Format）は、遺伝子配列のバリエーションを保存するためのテキストファイルの形式

- **変異コールを行うと、一般的に出力される結果ファイル**
  - ✓ Linuxコマンドの `cat` , `less` , `head` などで開くことができる。
  - ✓ テキストファイルなのでメモ帳やエクセルなどでも開くことができる。
  - ✓ 1行に1変異の情報がタブ区切りで記述される。

ヘッダー部分

```
##fileformat=VCFv4.2
##FORMAT=<ID=AD,Number=R,Type=Integer,Description="Allelic depths for the ref and alt alleles in the order listed">
##FORMAT=<ID=AF,Number=A,Type=Float,Description="Allele fractions of alternate alleles in the tumor">
##FORMAT=<ID=DP,Number=1,Type=Integer,Description="Approximate read depth (reads with MQ=255 or with bad mates are filtered)">
##FORMAT=<ID=F1R2,Number=R,Type=Integer,Description="Count of reads in F1R2 pair orientation supporting each allele">
＜中略＞
##MutectVersion=2.1
##contig=<ID=1,length=249250621>
##contig=<ID=2,length=243199373>
##normal_sample=5929_control
##source=Mutect2
##tumor_sample=5929_tumor
```

- **vcfファイルは「ヘッダー」部分と「変異情報」部分で構成される**
  - ✓ ヘッダー部分：＃から始まる。
    ファイルフォーマット、変異コールを行ったソフト、サンプル情報、変異情報部分で使用されるフィールド情報
  - ✓ 変異情報部分：タブ区切りで1行が1変異情報を表す。以下の基本8項目＋α（追加の付加情報）で構成される。

① #CHROM（染色体名）	⑤ ALT（検出された変異アレル）
② POS（変異ポジション）	⑥ QUAL（変異のクオリティスコア）
③ ID（rs番号など）	⑦ FILTER（変異コールソフトが定める変異コールの確からしさなどに対する閾値を超えたかどうかの情報）
④ REF（リファレンスゲノムでのアリル）	⑧ INFO（付加情報）

詳しい情報：https://samtools.github.io/hts-specs/VCFv4.2.pdf

変異情報部分

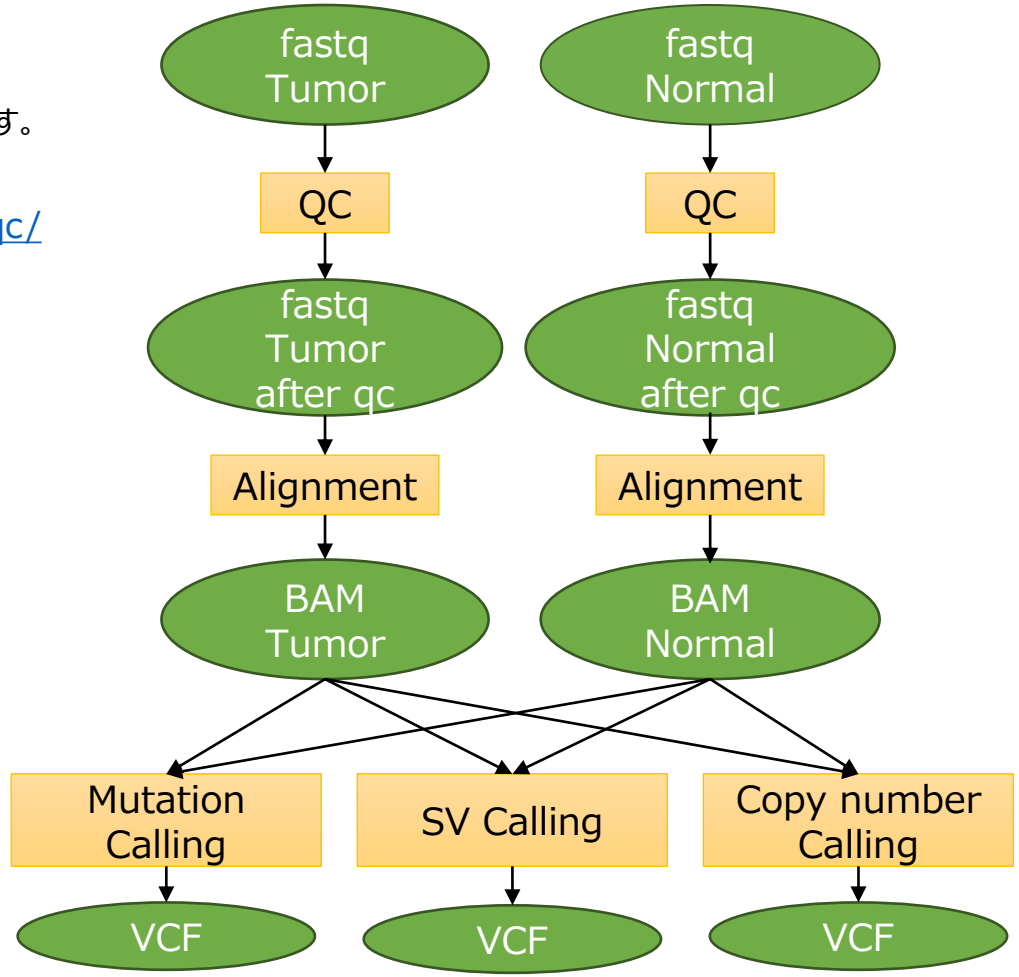
```
#CHROM  POS      ID    REF   ALT   QUAL  FILTER  INFO    FORMAT  5929_control    5929_tumor
chr1    978697   .     G     T     .     .       DP=146;ECNT=1;MBQ=37,35;MFRL=178,180;MMQ=60,60;MPOS=20;NALOD=1.87;NLOD=21.58;POPAF=6.00;SAAF=0.283,0.293,0.308;SAPP=0.015,0.021,0.964;TLOD=55.12    GT:AD:AF:DP:F1R2:F2R1   0/0:72,0:0.013:72:36,0:36,0     0/1:45,20:0.308:65:21,11:24,9
cbr1    6186793  .     C     A     .     .       DP=217;ECNT=1;MBQ=38,37;MFRL=181,184;MMQ=60,60;MPOS=21;NALOD=2.04;NLOD=31.90;POPAF=6.00;SAAF=0.535,0.556,0.568;SAPP=0.034,0.014,0.952;TLOD=187.24   GT:AD:AF:DP:F1R2:F2R1   0/0:106,0:8.981e-03:106:55,0:51,0       0/1:41,54:0.567:95:25,27:16,27
①       ②        ③     ④     ⑤     ⑥     ⑦       ⑧                                                                                                                          ↑以降、追加の付加情報
```

解析システム: 解析パイプライン

パイプラインとは複数のツールを組み合わせた一連のワークフローのこと。
各工程でどのツールを選ぶかは解析目的や好みなどによって異なる。

- **各工程にて使用される代表的なツール**　＊ 以降で紹介するツールは黄色でハイライトしてあります。
  - ✓ **QC**：fastqファイルのクオリティをチェックするツール
    - FASTQC＊：https://www.bioinformatics.babraham.ac.uk/projects/fastqc/
    - preseq：http://smithlabresearch.org/manuals/preseqmanual.pdf
  - ✓ **Alignment**:ゲノム配列データのアライメント
    - BWA*：https://github.com/lh3/bwa
    - Bowtie2：http://bowtie-bio.sourceforge.net/bowtie2/index.shtml
  - ✓ **Mutation Calling**
    - Mutect2*
    - VarScan
    - Deepvariant
  - ✓ **SV Calling**
    - Manta*
    - GRIDSS
  - ✓ **Copynumber Calling**
    - CNVkit*
    - Control-FREEC
- **各工程がパッケージ化されたパイプラインGenomon2**

  Genomon2：https://genomon.readthedocs.io/ja/latest/#



体細胞変異コールの一般的なパイプライン

解析: FASTQC（1）

FASTQCでは、リードのクオリティチェックを行う。

目的：計測したリードデータのクオリティチェック（QC）

理由：NGS解析の出発点となるfastqファイルに含まれるリードデータのクオリティがそもそも悪ければ以降の解析からは意義のある結果を得ることが難しい。
事前にクオリティチェックを行うことによって、のちに行う変異コールなどがうまくいかないといった事象の原因が、リードのクオリティであるのか、変異コールなどの解析自体にあるのかの切り分けをすることが可能となる。

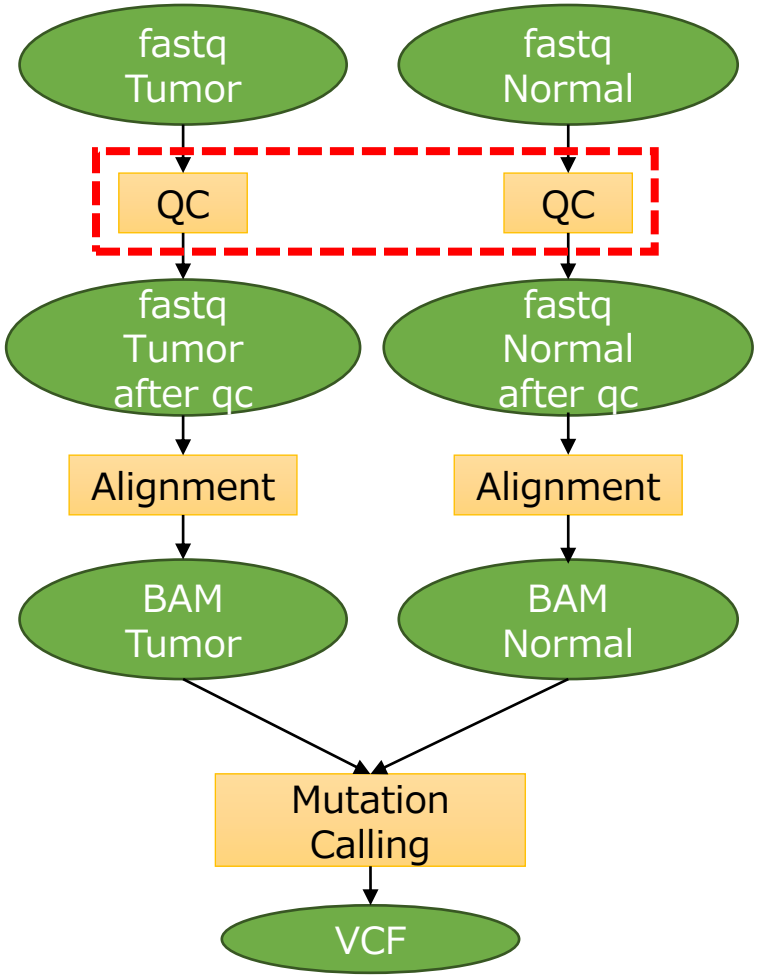
コマンド

```
fastqc –nogroup -o ~/fastqc_tumor_1 _output   ~/Data/sample _tumor_1.fq.gz
          ↑              ↑                        ↑
       オプション*    アウトプットディレクトリ          fastqファイル
```

2つのアウトプットファイル

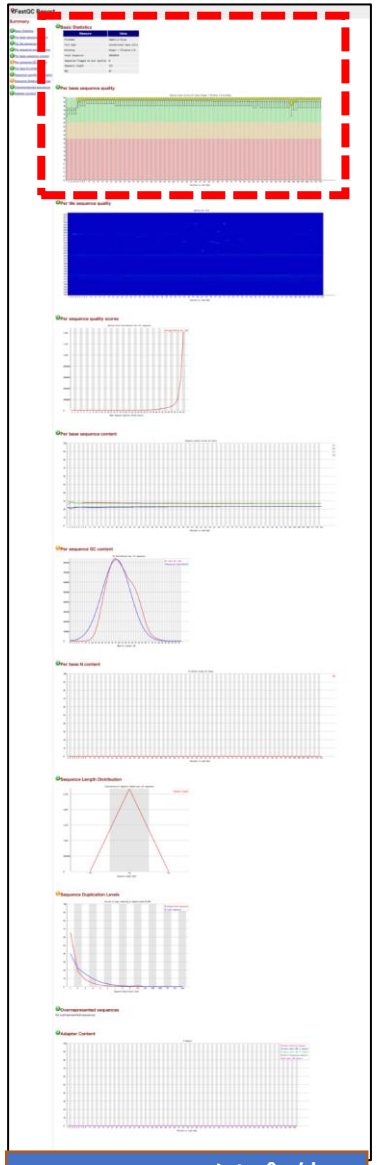
```
[fastqc_output$]ls
sample_1_fastqc.html   sample_1_fastqc.zip
```

＊ オプション「 -nogroup 」について:
リード長が50 bpよりも長い場合には3'側の塩基についての結果を1bpずつ評価するのではなく10bpごとなどまとめて行う。これを避けるためには「 -nogroup 」が必要。



体細胞変異コールの一般的なパイプライン

解析: FASTQC（2）



HTMLファイル全体

出力されたsample_1_fastqc.htmlをダブルクリックすることで、ブラウザ上で結果を見ることができる。

問題無し

注意

問題あり

FastQC Report

Summary

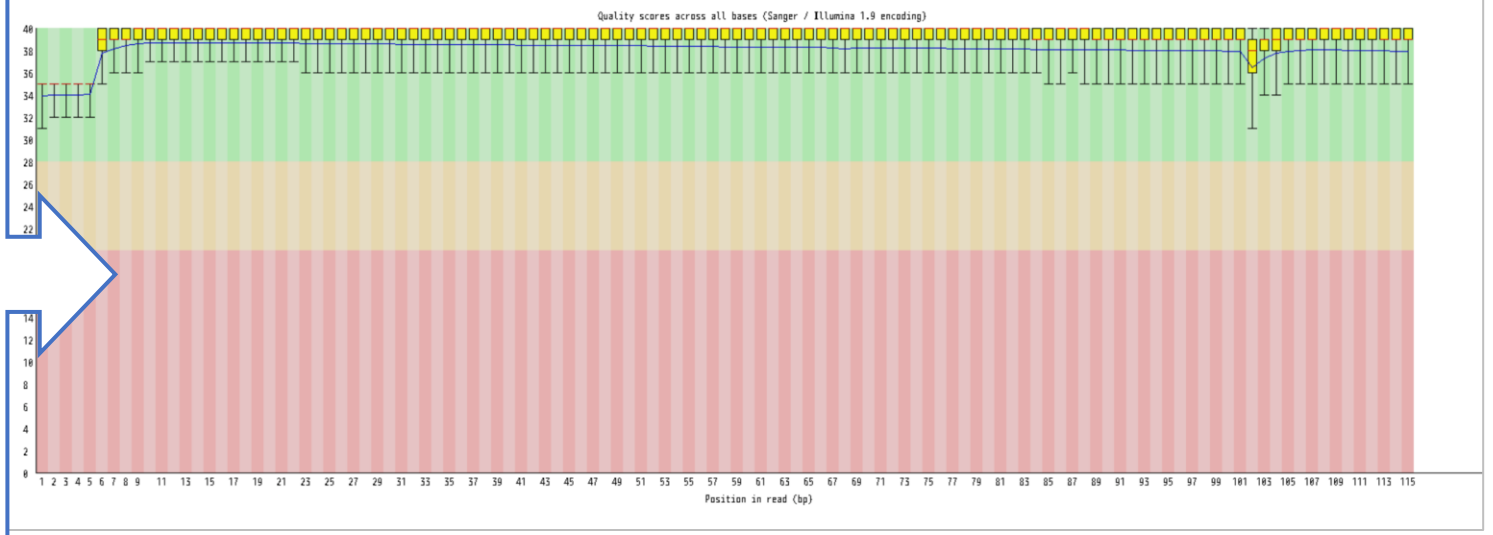
- Basic Statistics
- Per base sequence quality
- Per tile sequence quality
- Per sequence quality scores
- Per base sequence content
- Per sequence GC content
- Per base N content
- Sequence Length Distribution
- Sequence Duplication Levels
- Overrepresented sequences

Basic Statistics

Measure	Value
Filename	sample_1.fq.gz
File type	Conventional base calls
Encoding	Sanger / Illumina 1.9
Total Sequences	26658919
Sequences flagged as poor quality	0
Sequence length	115
%GC	45

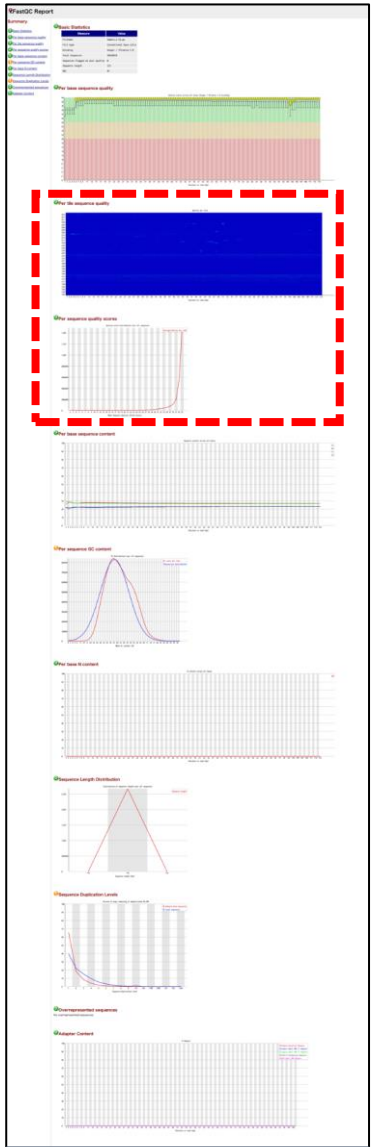
基本情報
- ファイル名
- リード数
- リード長　など

Per base sequence quality



リードのポジションごとのクオリティの高さを示す。
　横軸：塩基位置
　縦軸：クオリティスコア
ボックスプロットが橙色や赤色にあるとクオリティが悪い。リードの最初と最後でクオリティが低下することが多く、リードが長ければ長いほど後半なるにつれ低下する。またリード1よりもリード2のほうが低い傾向にある。あまりに低下がひどい場合にはトリミングなどを検討する。

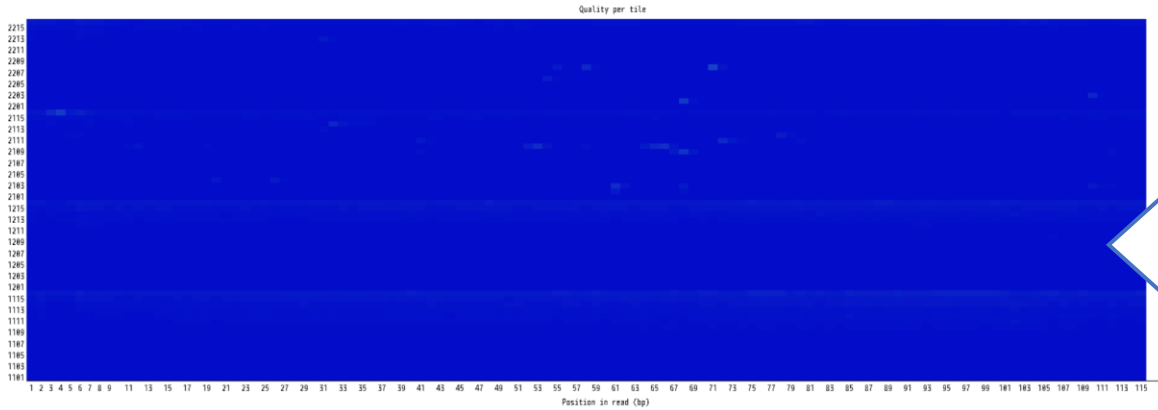
解析: FASTQC（3）



HTMLファイル全体

出力されたsample_1_fastqc.htmlをダブルクリックすることで、ブラウザ上で結果を見ることができる。

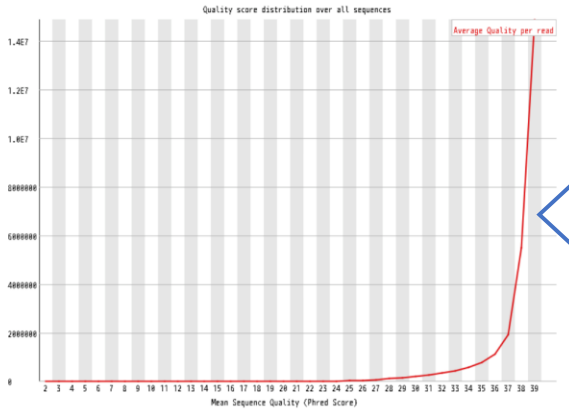
Per tile sequence quality



フローセルでの読み取り品質を示し、クオリティが悪ければ赤色となる。
横軸：塩基位置
縦軸：サイクル数

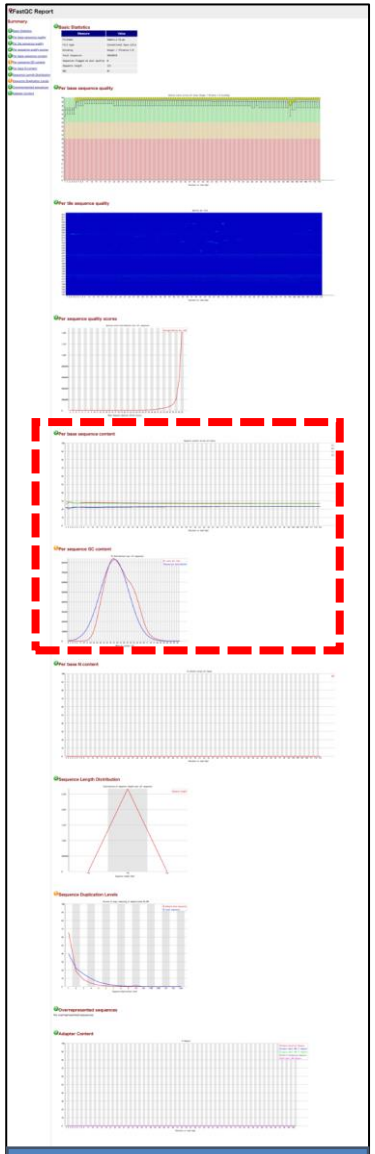
左図は問題がない例であり、赤色は見えない。

Per sequence quality scores



リードの平均クオリティスコアを示す。
横軸：クオリティスコア
縦軸：頻度

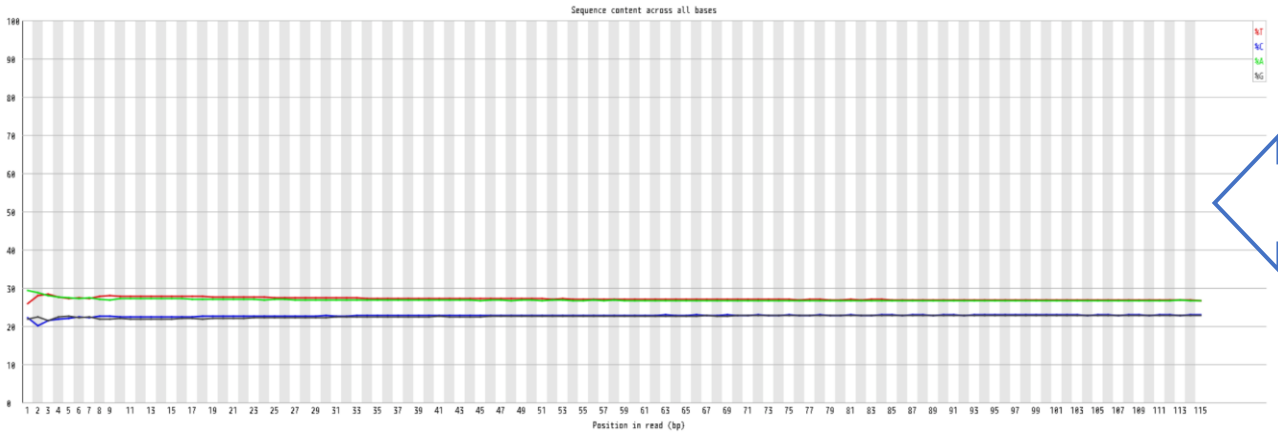
解析: FASTQC（4）



HTMLファイル全体

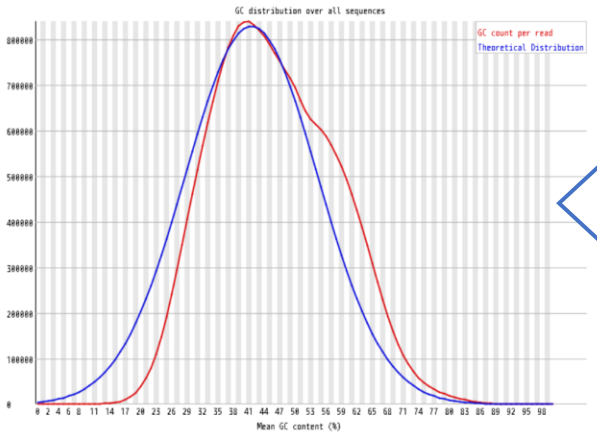
出力されたsample_1_fastqc.htmlをダブルクリックすることで、ブラウザ上で結果を見ることができる。

Per base sequence content



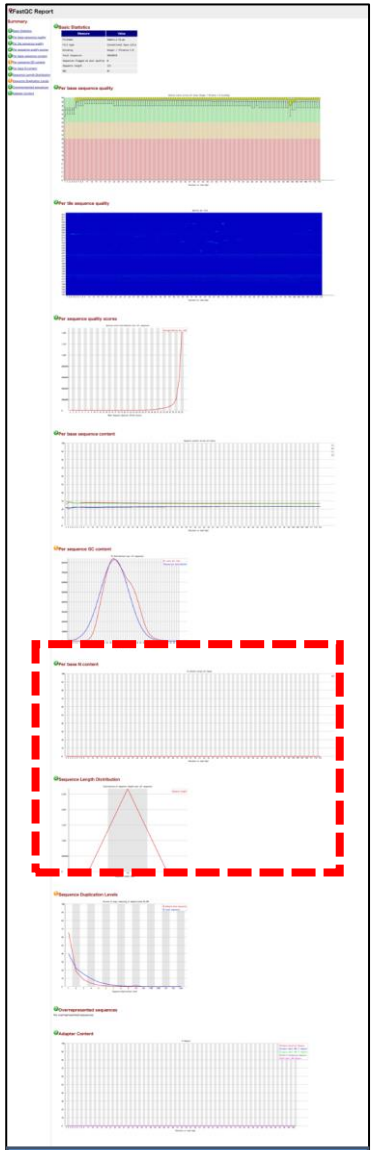
各リードのポジション別ATCG割合
横軸：塩基位置
縦軸：サイクル数

Per sequence GC content



各リードにおける平均 GC Contents（赤線）
理論上は正規分布（青線）となる。
横軸：GC含有率
縦軸：頻度

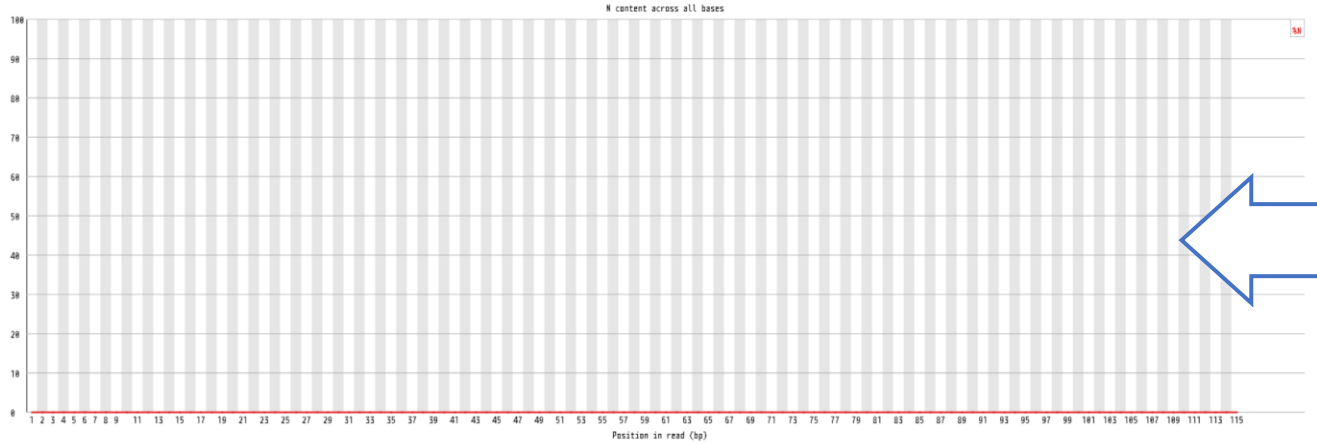
解析: FASTQC（5）



HTMLファイル全体

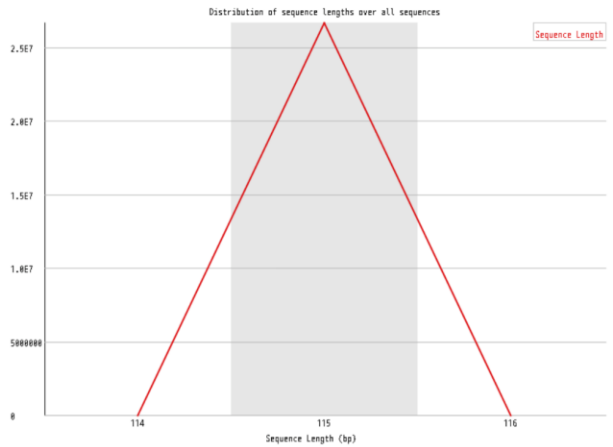
出力されたsample_1_fastqc.htmlをダブルクリックすることで、ブラウザ上で結果を見ることができる。

Per base N content



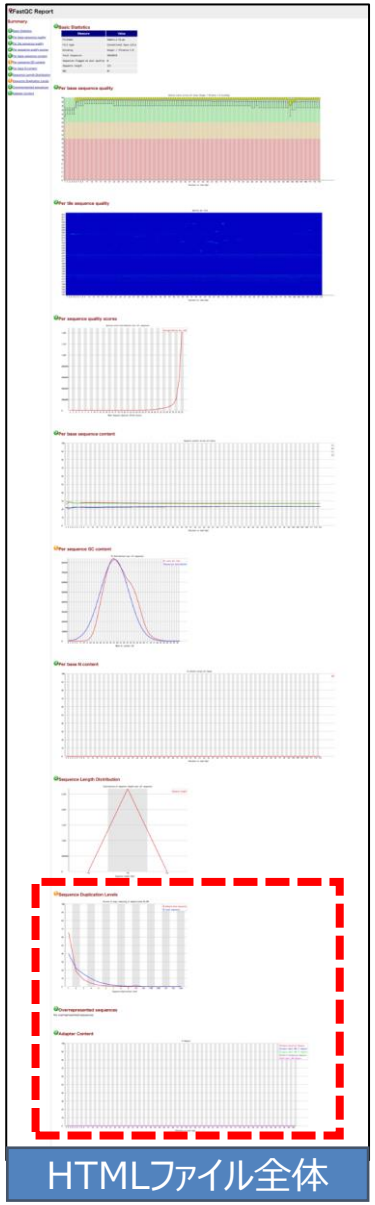
シークエンサーがベースコールの結果ATGCのいずれでもないと判断したリードの部位は「N」となる。その割合を示す。
　横軸：塩基位置
　縦軸：割合

Sequence Length Distribution



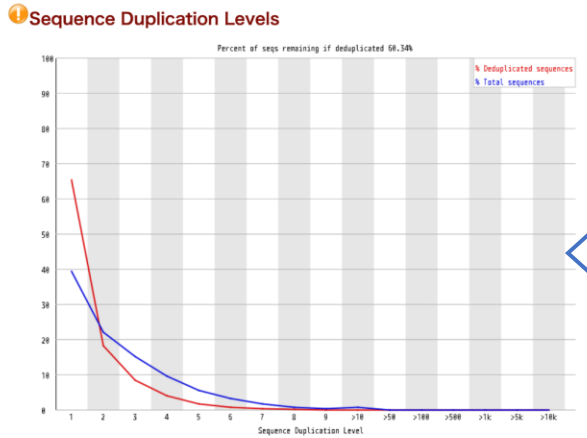
リードの長さの分布。
　横軸：リード長
　縦軸：頻度
シークエンスの設定によって、均一な長さのフラグメントを生成するものもあるが、長さが大きく異なるリードを含むものもある。

解析: FASTQC（6）

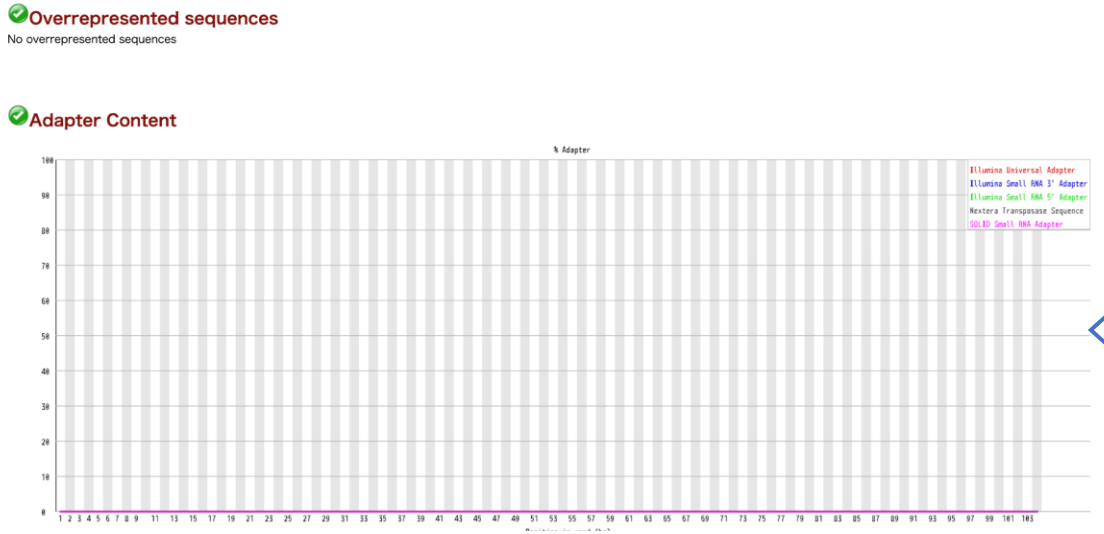


HTMLファイル全体

出力されたsample_1_fastqc.htmlをダブルクリックすることで、ブラウザ上で結果を見ることができる。



配列の重複度の相対的カウント。
適切に設計されたライブラリーでは、赤線と青線ともにこのプロットのように左端で高い割合を示す。
　横軸：重複塩基数
　縦軸：頻度



アダプター配列の有無を示す。
　横軸：塩基位置
　縦軸：頻度
アダプターが混入する場合には、これを除外するなどの処理を行う必要がある。

解析: bwa 事前準備

bwaで、mappingを行うためにリファレンス配列を入手し、indexファイルを準備をする。

- **事前準備：GRCh38 リファレンスファイルを入手。ダウンロードにはGoogleアカウントが必要。**

  ダウンロード先：https://console.cloud.google.com/storage/browser/genomics-publicdata/resources/broad/hg38/v0;tab=objects?prefix=&forceOnObjectsSortingFiltering=false

  ✓ **DL対象ファイルは以下。これを同じディレクトリに格納する。**

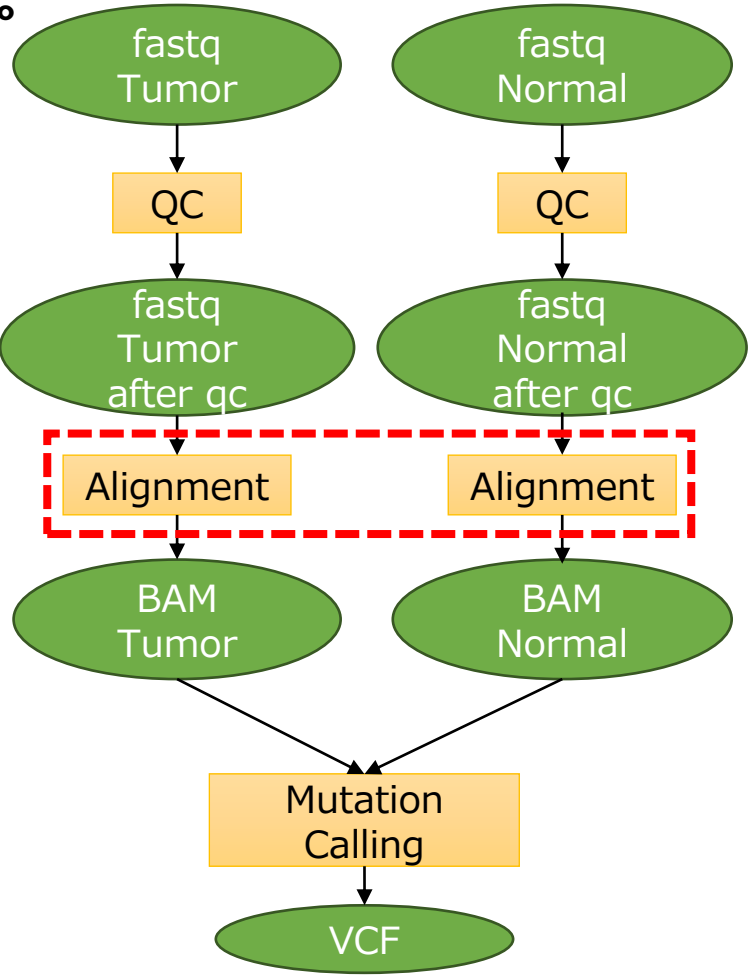
  Homo_sapiens_assembly38.fasta ←これがリファレンスファイル。その他はindexファイル*。
  Homo_sapiens_assembly38.fasta.fai
  Homo_sapiens_assembly38.fasta.64.alt
  Homo_sapiens_assembly38.fasta.64.amb
  Homo_sapiens_assembly38.fasta.64.ann
  Homo_sapiens_assembly38.fasta.64.bwt
  Homo_sapiens_assembly38.fasta.64.pac
  Homo_sapiens_assembly38.fasta.64.sa
  Homo_sapiens_assembly38.fasta.dict ←GATK用indexファイル
  Homo_sapiens_assembly38.known_indels.vcf.gz ←GATK用
  Homo_sapiens_assembly38.known_indels.vcf.gz.tbi ←GATK用

  * indexファイルとはリファレンスファイルに対して付加情報を与えたり、計算を高速化するための辞書的なファイル

  ✓ **bwaをかける前にリファレンスファイルから `bwa index` コマンドでindexファイルを作成することも可能。**

コマンド

```
bwa index -6  ~/Ref/Homo_sapiens_assembly38.fasta
          ↑                ↑
       オプション      リファレンスファイル
```



体細胞変異コールの一般的なパイプライン

解析: bwa bamファイルの作成

fastqファイルのリファレンス配列へのアライメントを実行する。 ※Step1以外はNormalサンプルのコードは書いていないので注意。

Step1：fastqファイルをリファレンスファイルにアライメント 出力ファイルはBAM形式ではなくSAM形式であることに注意。
このsamファイルは並び替え（sort）されていない。

Step1：Tumorサンプルのアライメント

```
bwa mem -t 20 ¥ ←並列数
-R ‘@RG¥tID:sample_tumor¥tLB:lib1¥tPL:illumina¥tSM:sample_tumor¥tPU:sample_tumor’ ¥  ←リードグループ情報
~/Ref/Homo_sapiens_assembly38.fasta ¥  ←リファレンスファイル
~/Data/sample_tumor_1.fq.gz ~/Data/sample_tumor_2.fq.gz ¥  ←2つのfastqファイル
> ~/Data/sample_tumor.sam ←出力するsamファイルの名前
```

Step1：Normalサンプルのアライメント

```
bwa mem -t 20 ¥ ←並列数
-R ‘@RG¥tID:sample_normal¥tLB:lib1¥tPL:illumina¥tSM:sample_normal¥tPU:sample_normal’ ¥  ←リードグループ情報。ここでつけた情報をStep7で利用する
~/Ref/Homo_sapiens_assembly38.fasta ¥  ←リファレンスファイル
~/Data/sample_normal_1.fq.gz ~/Data/sample_normal_2.fq.gz ¥  ←2つのfastqファイル
> ~/Data/sample_normal.sam ←出力するsamファイルの名前
```

Step2：sortとbamファイルへの変換 並び替えを行いながら、samファイルをbamファイルへ変換する。
（以下はTumorサンプルのコードです。Normalサンプルについても同じように実施しましょう。）

Step2：sortとbamファイルへの変換

```
samtools sort  -@ 4 ¥ ←並列数
./Data/sample_tumor.sam ¥ ←入力するsamファイルの名前
-o ./Data/sample_tumor.bam  ←出力するファイル名。これは並べ替えた後のものになる
```

解析: bamファイルへの情報付加と補正

fastqファイルのリファレンス配列へのアライメントを実行する。

Step3 : Markduplicates 通常の変異コールでは重複リードがあった場合、重複を無視し1リードとして扱う。そのため変異コールをする前に重複リードを同定し、付加情報としてbamファイルに与える。重複リードは PCR の duplication などを起因として生じる

Step3 : Markduplicates

```
gatk MarkDuplicates ¥
--java-options -Xmx30g  ¥ ←メモリ要求量
-I ~/Data/sample.bam ¥ ←input ファイル
-O ~/Data/sample_tumor_markdup.bam ¥ ←output ファイル
-M ~/Data/sample_tumor_metrics.txt ← duplication readと判定されたリード数などの統計情報
```

Step4 : BaseRecalibrator 各リードのクオリティスコアの補正を行うためのテーブルを作成する。よく知られた variants がある領域の情報を known-sites として与えることで、Step5でクオリティスコアを補正する。

Step4 : BaseRecalibrator

```
gatk BaseRecalibrator --java-options -Xmx30g ¥ ←メモリ要求量
--input ~/Data/sample_tumor_markdup.bam ¥ ←input ファイル
--output ~/Data/sample_tumor_recal.txt ¥ ←output ファイル
--known-sites ~/Ref/Homo_sapiens_assembly38.known_indels.vcf.gz ¥ ← known-sitesが書かれたファイル
--reference ~/Ref/Homo_sapiens_assembly38.fasta  ←リファレンスファイル
```

解析: bamのクオリティスコア補正

クオリティスコアを補正したbamファイルの作成を行う。

Step5 : ApplyBQSR Step5の出力ファイルを使いクオリティスコアを補正したbamを作成する。

Step5 : ApplyBQSR

```
gatk ApplyBQSR --java-options -Xmx30g ¥ ←メモリ要求量
-R ~/Ref/Homo_sapiens_assembly38.fasta ¥ ←リファレンスファイル
-I ~/Data/sample_tumor_markdup.bam ¥ ←input ファイル
--bqsr-recal-file sample_tumor_recal.txt ¥ ←step4.BaseRecalibratorでのアウトプット
-O ~/Data/sample_tumor_markdup_updated.bam  ←outputファイル
```

Step6 : bam indexファイルの作成 bam に対するindexファイルを作成する

Step6 : bam indexファイルの作成

```
samtools index ~/Data/sample_tumor_markdup_updated.bam
```

Step2~Step6をNormalサンプルに対しても行い、sample_normal_markdup_updated.bam を準備しましょう。

解析: 変異コール

Mutect2によって変異コールを実施する。

Step7 : Mutect2 Normal と Tumor それぞれについてのbamファイルができたら、いよいよ変異コール

Step7 : Mutect2

```
gatk Mutect2 -R ~/Ref/Homo_sapiens_assembly38.fasta   ¥ ←リファレンスファイル
-I ~/Data/sample_tumor_markdup_updated.bam   ¥   ←Tumorのbamファイル
-I ~/Data/sample_normal_markdup_updated.bam ¥   ←Normalのbamファイル
--normal-sample   sample_normal   ¥   ←Normalのサンプル名
--output result.vcf    ←結果vcfファイル
```

Step1 : mapping でつけたリードグループ情報のうち、Normalサンプルの SMで指定したのと同じものにする。

```
-R ‘@RG¥tID:sample_normal¥tLB:lib1¥tPL:illumina¥tSM:sample_normal¥tPU:sample_normal’ ¥   ←リードグループ情報。ここでつけた情報をStep7で利用する
```

Step7の結果、VCFを入手できた。

```
##fileformat=VCFv4.2
##FILTER=<ID=LowQual,Description="Low quality">
##FORMAT=<ID=AD,Number=R,Type=Integer,Description="Allelic depths for the ref and alt alleles in the order listed">
##FORMAT=<ID=DP,Number=1,Type=Integer,Description="Approximate read depth (reads with MQ=255 or with bad mates are filtered)">
##FORMAT=<ID=GQ,Number=1,Type=Integer,Description="Genotype Quality">
<中略>
#CHROM  POS     ID      REF     ALT     QUAL    FILTER  INFO
chr1    978697  .       G       T       .       .       DP=146;ECNT=1;MBQ=37,35;MFRL=178,180;MMQ=60,60;MPOS=20;NALOD=1.87;NLOD=21.58;POPAF=6.00;SAAF=0.283,0.293,0.308;SAPP=0.015,0.021,0.964;TLOD=55.12       GT:AD:AF:DP:F1R2:F2R1   0/0:72,0:0.013:72:36,0:36,0     0/1:45,20:0.308:65:21,11:24,9
chr1    6186793 .       C       A       .       .       DP=217;ECNT=1;MBQ=38,37;MFRL=181,184;MMQ=60,60;MPOS=21;NALOD=2.04;NLOD=31.90;POPAF=6.00;SAAF=0.535,0.556,0.568;SAPP=0.034,0.014,0.952;TLOD=187.24       GT:AD:AF:DP:F1R2:F2R1   0/0:106,0:8.981e-03:106:55,0:51,0       0/1:41,54:0.567:95:25,27:16,27
…
```

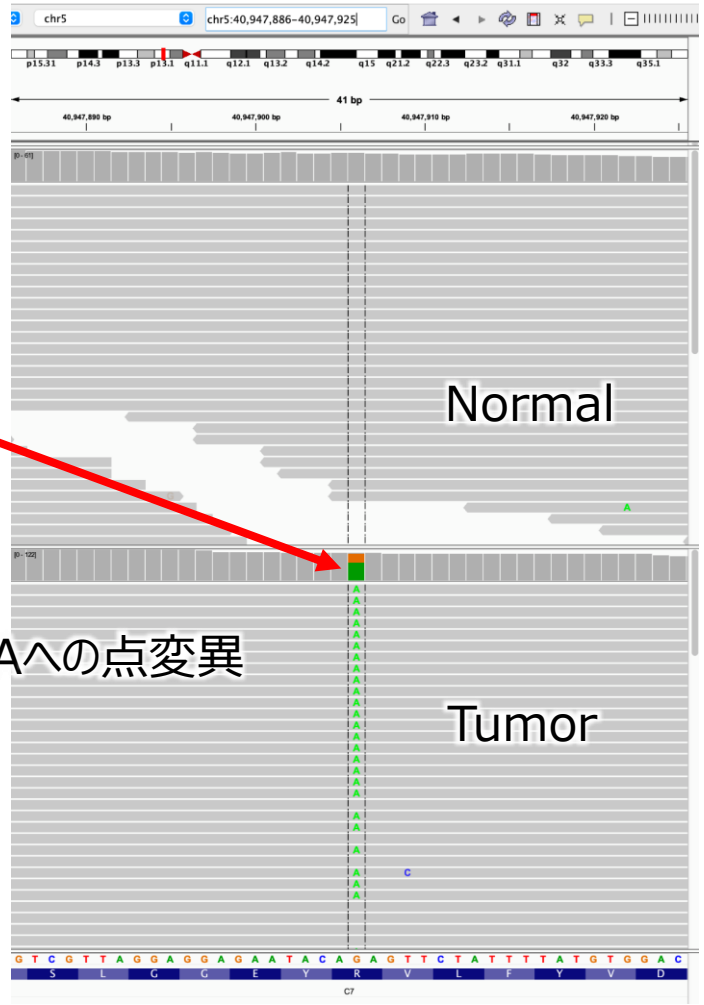
※上記例では、改行位置がわかりやすいようにフォントサイズを一部小さく設定して同じ行を1行で表示

解析結果例

Mutect2によって得られたVCFの結果から得られた点変異をIGVで確認する。

VCF一部抜粋

chr4	80990696	.	T	A	100	PASS
chr4	160264237	.	A	G	100	PASS
chr4	160271416	.	T	C	100	PASS
chr5	10947906	.	G	A	100	PASS
chr5	55034736	.	C	T	100	PASS
chr5	147790186	.	C	G	100	PASS

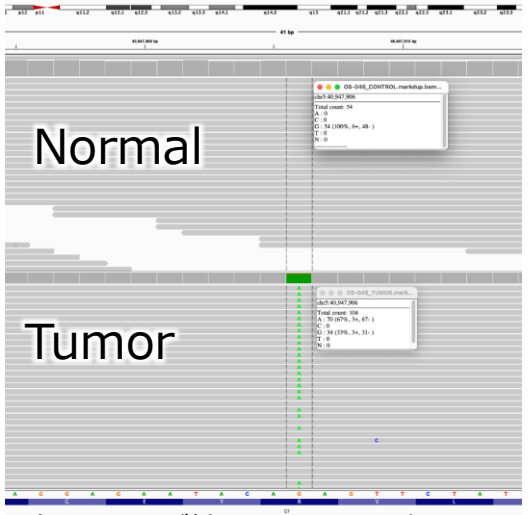


chr5:10947906でG＞Aへの点変異

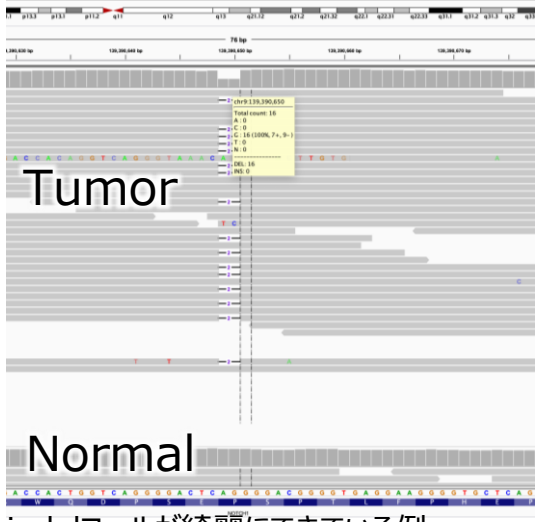
解析: Mutect2の出力VCFに対する検討

Step7までの手順で変異のリストであるVCFを手に入れることができた。しかし、変異コールにはfastqファイルそのものの品質や同じ配列が繰り返されるリピート領域でのマッピング自体の難しさ等、様々な困難がある。VCFファイルにはこれらに起因する誤りが含まれる場合もあり、得られた結果が真に意味のある変異であるかを吟味する必要がある。この吟味は、例えば変異のリスト全体のVariant Allele Frequency(VAF)に対してDepthがある閾値以上の場合のみを対象にする等、一括で行う場合もあり、これを「フィルタを行う」という。
フィルタの方法は様々あるが、第 Ⅳ 章で学ぶ。また、変異コールの結果の精度を確認するために IGVを使うことが多々あり、VCFだけでは分からなかった様々な情報を視覚的に理解することができる。

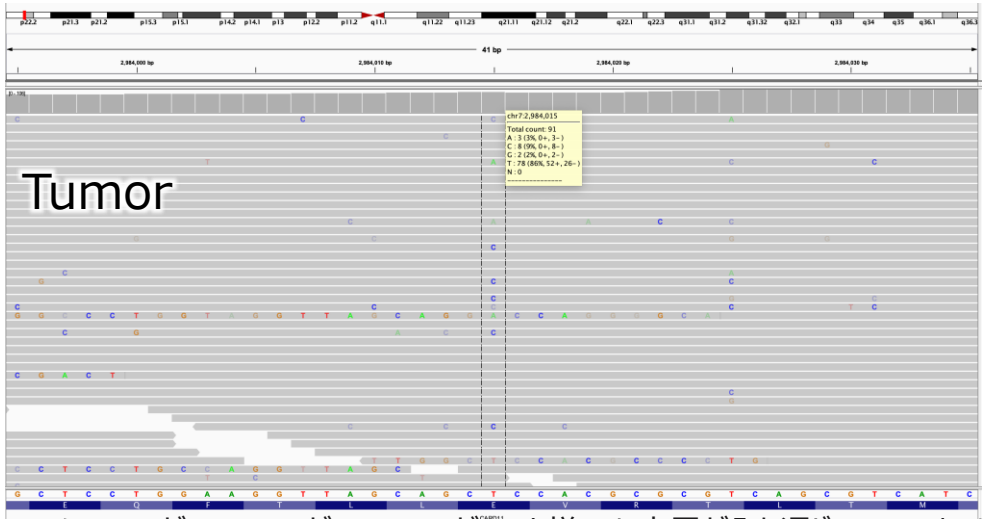
- vcfファイルの結果を全て鵜呑みにしてはいけない
  - ✓ 変異としてコールされたポジションにリードは全部でいくつあるか？リードの数があまりにも少ない場合には信頼性が低い。
  - ✓ リードはソフトクリップされていないか？ソフトクリップされているということは、そもそもリードのマッピングが正確に行われていないことを示します。
  - ✓ VAFはどの程度か？周辺のマッピングの状況はどうか？



変異コールが綺麗にできている例。
Normal も Tumor も depth が十分で周辺にソフトクリップもない。



indelコールが綺麗にできている例。
2ベースの Deletion が Tumor のみで見られる。



chr7:2984015 にてT>Cが9%、T>Aが3%、T>Gが2%と様々な変異が入り混じっているように見える。VCFファイルでは chr7:2984015T>Cがコールされていたが、周辺にエラーとなった変異として検出されたC が多く、シークエンスエラーが考えられる。確証を得るためには実験を行い確認するなどが必要。

解析: 構造変異の検出（1）

mantaを用いて構造変異の検出を行う。

✓ step6までに作成したbamファイルを用いて構造変異の検出をおこなう。ここでは manta を使う。事前準備と本解析の2段階で行う

Step8：manta その1 事前準備 manta を使うための事前準備を行う。

Step8：manta その1 事前準備

```
configManta.py ¥
--normalBam ~/Data/sample_normal_markdup_updated.bam ¥ ←Step6までに作成したNormalのbamファイル
--tumorBam  ~/Data/sample_tumor_markdup_updated.bam ¥ ←Step6までに作成したTumorのbamファイル
--referenceFasta ~/Ref/Homo_sapiens_assembly38.fasta  ¥ ←リファレンスファイル
--runDir ~/Data/manta ←アウトプットディレクトリ（あらかじめ作っておく）
```

Step8のアウトプットファイル群

```
[~$]tree ~/Data/manta
 .
├── results
│   ├── evidence
│   ├── stats
│   └── variants
├── runWorkflow.py ←Step9で使うファイルができている
├── runWorkflow.py.config.pickle
└── workspace
```

解析: 構造変異の検出（2）

mantaを用いて構造変異の検出を行う。

Step9：manta　その2　本番　manta を使って本解析を行う。

Step9：manta　その2　本番

```
python ~/Data/manta/runWorkflow.py ←Step8の結果入手したファイル
```

Step9のアウトプットファイル群

```
[~$]tree -L 3 ~/Data/manta
├── results
│   ├── evidence
│   ├── stats
│   │   ├── alignmentStatsSummary.txt
│   │   ├── svCandidateGenerationStats.tsv
│   │   ├── svCandidateGenerationStats.xml
│   │   └── svLocusGraphStats.tsv
│   └── variants ←このディレクトリにあるものが主たる結果
│       ├── candidateSV.vcf.gz   ←SVおよびインデル候補
│       ├── candidateSV.vcf.gz.tbi   ←上のindexファイル
│       ├── candidateSmallIndels.vcf.gz   ←candidateSV.vcf.gzファイルからの抜粋、デフォルトでは50以下のスコアであるもの。
│       ├── candidateSmallIndels.vcf.gz.tbi   ←上のindexファイル
│       ├── diploidSV.vcf.gz   ←正常サンプルに対してdiploidモデルの下でコールされたSVおよびindel
│       ├── diploidSV.vcf.gz.tbi   ←上のindexファイル
│       ├── somaticSV.vcf.gz ←normal-tumorペアの元でのSVおよびindel
│       └── somaticSV.vcf.gz.tbi   ←上のindexファイル
├── runWorkflow.py
├── runWorkflow.py.config.pickle
├── workflow.error.log.txt
```

続き

```
├── workflow.exitcode.txt
├── workflow.warning.log.txt
└── workspace
    ├── alignmentStats.xml
├── chromDepth.txt
├── edgeRuntimeLog.txt
├── pyflow.data
│   ├── logs
│   └── state
├── svHyGen
│   ├── candidateSV.0000.vcf
│   ├── diploidSV.0000.vcf
│   ├── edgeRuntimeLog.0000.txt
│   ├── list.edgeRuntimeLog.txt
│   ├── list.sortCandidateSV.txt
│   ├── list.sortDiploidSV.txt
│   ├── list.sortSomaticSV.txt
│   └── somaticSV.0000.vcf
└── svLocusGraph.bin
```

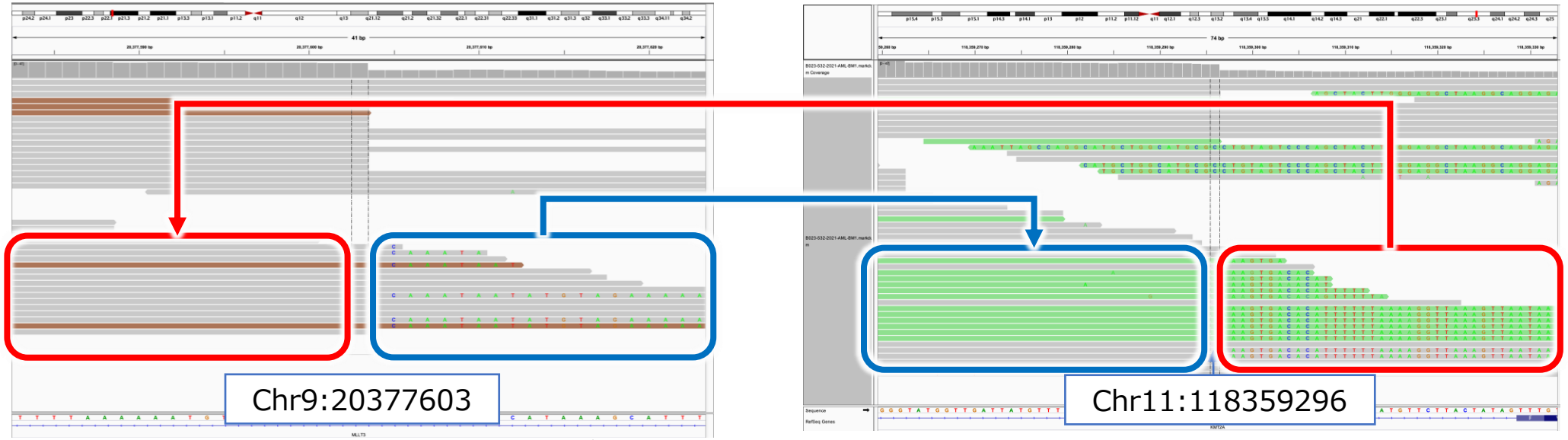
詳細な見方はこちら：https://github.com/Illumina/manta/blob/master/docs/userGuide/README.md

解析: mantaの出力VCFに対する検討

Step9までの手順でSVコールが終わり、VCFを手に入れることができた。

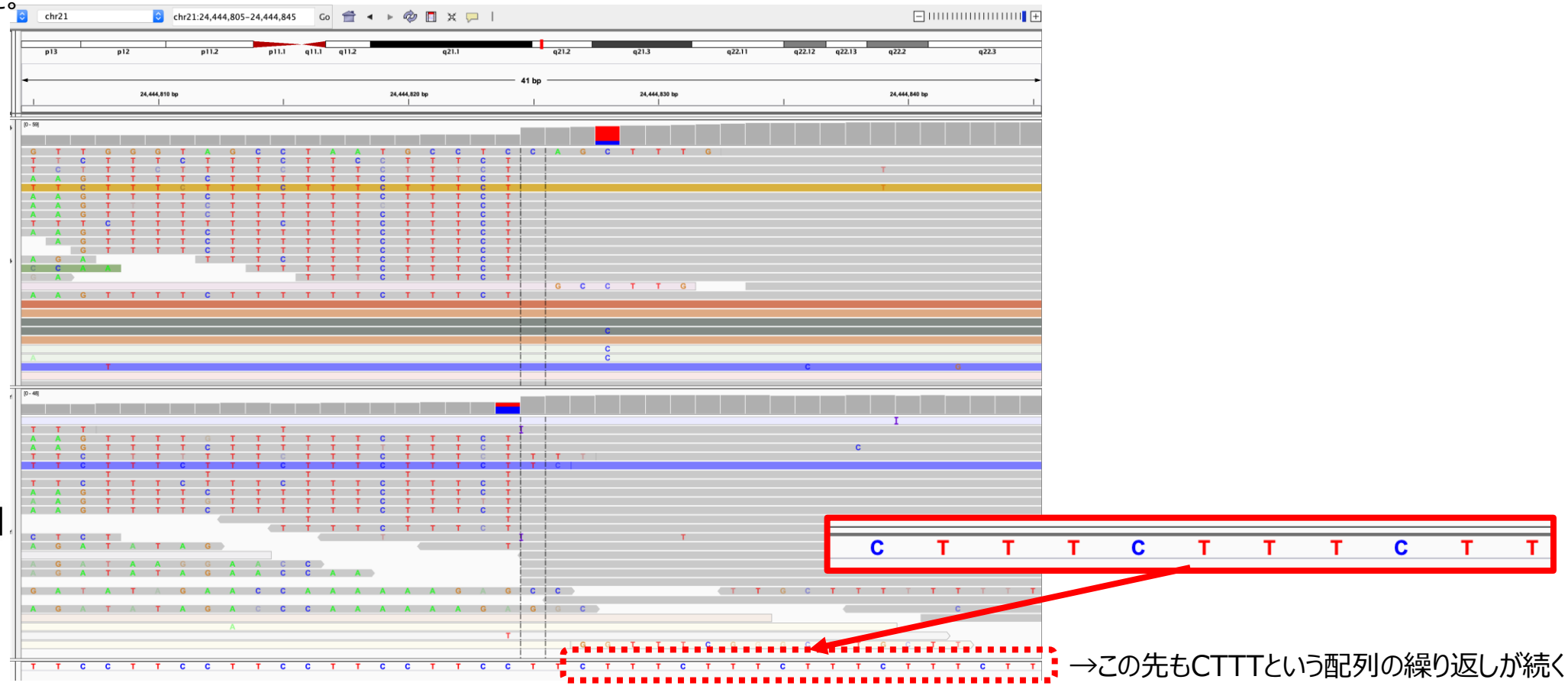
ここでも mutect2 の時と同じように、結果には誤りが含まれている可能性があるため、VCFファイルを精査する必要がある。

- vcfファイルの結果を全て鵜呑みにしてはいけない。
  - ✓ SVとしてコールされた結果は本当に構造変異を起こしているのか？必要に応じてIGVで確認
  - ✓ 十分な数のサポートリードがあるか？またその質はどうであるか？周辺のマッピングの状況はどうか？
    - ➢ Chr9:20377603 と Chr11:118359296 を break point とする融合遺伝子がうまく見つかった例
      - Chr9 の青で囲まれたソフトクリップされた配列は Chr11 の青で囲まれた部分のリードと同じ配列
      - Chr11 の赤で囲まれたソフトクリップされた配列は Chr9 の赤で囲まれた部分のリードと同じ配列



解析: mantaの出力VCFに対する検討

- Chr21:24444825 と Chr4:118372915 の融合遺伝子として検出されたが間違っていた例（上段Tumor,下段Normal）
  - Normal にも Tumor にも同じソフトクリップされた配列がある。この配列は「CTTT」
  - 本来、Chr21 でもう少し右にスライドして mapping されるべきリードが間違ってアライメントされたためソフトクリップとなり、これがSVとして検出された。



解析:Copy number 解析

Step6までに作成したbamファイルを用いてコピー数変異の検出をおこなう

Step10 : cnvkit.py コピー数変異の検出を行う。

Step10 : cnvkit.py

```
cnvkit.py batch ~/Data/sample_tumor_markdup_updated.bam ¥ ←Step6までに作成したTumorのbamファイル
 --normal ~/Data/sample_normal_markdup_updated.bam ¥ ←Step6までに作成したNormalのbamファイル
 -m wgs ¥ ←WGSであることを明示
 --fasta ~/Ref/Homo_sapiens_assembly38.fasta ¥ ←リファレンスファイル
 --output-reference ~/Data/cnvkit/my_reference.cnn --output-dir ~/Data/cnvkit ¥ ←アウトプットディレクトリの指定
--diagram --scatter ←作図を行うことを明示
```

Step10のアウトプットファイル群

```
[~$]tree ~/Data/cnvkit
.
├── GRCh38.antitarget.bed
├── GRCh38.target.bed
├── sample_normal_markdup_updated.antitargetcoverage.cnn
├── sample_normal_markdup_updated.targetcoverage.cnn
├── sample_tumor_markdup_updated-diagram.pdf
├── sample_tumor_markdup_updated-scatter.png
├── sample_tumor_markdup_updated.antitargetcoverage.cnn
├── sample_tumor_markdup_updated.bintest.cns
├── sample_tumor_markdup_updated.call.cns
├── sample_tumor_markdup_updated.cnr
├── sample_tumor_markdup_updated.cns
├── sample_tumor_markdup_updated.targetcoverage.cnn
└── my_reference.cnn
```

pdfファイルやpngファイルはコピーナンバー変換を図示してあり直感的にわかりやすい。
必要に応じてcnr, cnsファイルをRなどで加工する。

詳細な見方はこちら：https://cnvkit.readthedocs.io/en/stable/index.html

解析:Copy number 解析 の出力

sample_tumor.markdup.call.cns

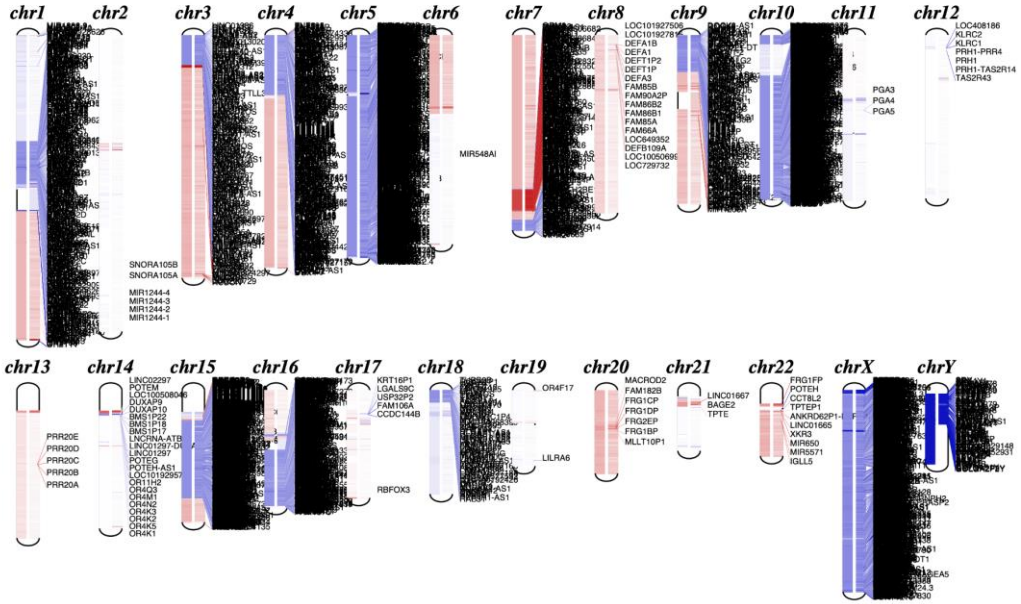
詳細な見方はこちら：https://cnvkit.readthedocs.io/en/stable/index.html

```
chromosome      start   end     gene    log2    cn      depth   p_ttest probes  weight
chr1    11220   350409  DDX11L1,DDX11L1,WASH7P,WASH7P,WASH7P,MIR6859-1,MIR6859-2,MIR6859-3,MIR6859-4,MIR1302-2,MIR1302-9,MIR1302-10,MIR1302-
11,FAM138A,FAM138F,FAM138C,OR4F5,LOC729737,DDX11L17,MIR6859-1,MIR6859-2,MIR6859-3,MIR6859-4,FAM138D,-       0.293864        3       150.522 1.08713e-25     157     130.173
chr1    350409  500581  OR4F3,OR4F16,OR4F29,LOC100132287,LOC100132062   -0.43736        1       79.976  4.83085e-31     118     104.449
chr1    500581  535988  -       0.261567        3       134.72  2.84432e-08     29      27.1026
```

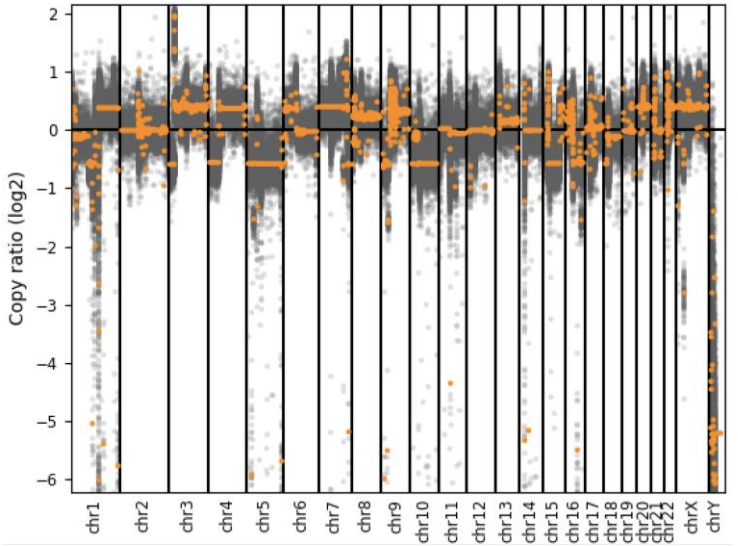
sample_tumor.markdup.call.cnr

```
chromosome      start   end     gene    log2    depth   probes  weight  ci_lo   ci_hi
chr1    11220   258887  DDX11L1,DDX11L1,WASH7P,WASH7P,WASH7P,MIR6859-1,MIR6859-2,MIR6859-3,MIR6859-4,MIR1302-2,MIR1302-9,MIR1302-10,MIR1302-
11,FAM138A,FAM138F,FAM138C,OR4F5,LOC729737,DDX11L17,MIR6859-1,MIR6859-2,MIR6859-3,MIR6859-4,FAM138D  0.25539 137.546 124     103.219 0.239523        0.268743
chr1    258887  288197  -       0.699529        234.17  23      17.1224 0.667727        0.734319
chr1    288197  350409  -       0.254547        141.066 10      9.83144 0.225468        0.288438
chr1    350409  500581  OR4F3,OR4F16,OR4F29,LOC100132287,LOC100132062   -0.417477       79.976  118     104.449 -0.436742       -0.39817
chr1    500581  535988  -       0.28145 134.72  29      27.1026 0.261794        0.298822
```

sample_tumor_markdup_updated-diagram.pdf



sample_tumor_markdup_updated-scatter.png



まとめ

- NGS解析で得られたデータをどのソフトを用いて解析を行うかについては多くの選択肢があります。また日進月歩で新しいソフトやツールが登場しており、これを行えば絶対に間違いがない、というものではありません。データが生成された実験条件や解析目的によってツールを選ぶ必要があります。また多くのソフトウェアは開発者が丁寧な使い方やアウトプットに関する説明をHPで公開しています。ぜひソフトウェア開発者の説明を読んでみましょう。
- NGS解析では、実験条件、シークエンスエラー、マッピングエラー、ソフトウエアの種類、変異コールなどを行う際に決める閾値など、様々な要因で結果が異なってきます。得られたVCFを盲信せず、データに立ち戻り確認を行うことが大切です。
- 近年のNGS解析では出発点となるfastqファイルは巨大です。例えばWGSの30Xのfastqファイルは圧縮された状態でもリード1、リード2を合わせて40～60GBほどあります。そしてこれらを扱うためには十分なデータの保存領域や多くの計算リソースが必要です。スーパーコンピュータやクラウドの利用をぜひ検討してください。

option: SHIROKANEユーザー向け

- 実際には入力ファイルのサイズ（どれくらいの Depth の WGS であるかなどによって変わる）によって、メモリ要求量を上げて実行をすることを考えたりする必要があります。 WGS データは巨大であるため、インタラクティブジョブと呼ばれる会話的なジョブ実行は時間がかかりすぎて現実的ではありません。前項まででは、わかりやすさを重視し、インタラクティブジョブでの実行例のようにコマンドを書きましたが、実務では大規模・長時間実行を前提としたbatchジョブの導入を強く勧めます。

- 以降の option ページは SHIROKANE で実行を行いたい時に、どのように解析ツールを呼び出すかを書いたものです。
  ＊batchジョブの方法を記載したものではありませんので注意してください。

option:SHIROKANEでFASTQCを実行するために

SHIROKANE にはソフトウェア FASTQC がインストールされています。

これを使うためには qlogin をしたのち、以下のコマンドを実行します。

コマンド

```
module use /usr/local/package/modulefiles/
module load fastqc/0.11.8
```

option: SHIROKANEでStep1,2を行うために

SHIROKANE にはソフトウェア bwa がインストールされています。また、マッピングの後処理で使う samtools もインストールされています。

これらを使うためには qlogin をしたのち、以下のコマンドを実行します。

コマンド

```
module use /usr/local/package/modulefiles/
module load bwa/0.7.17
module load samtools/1.9
```

option: SHIROKANEでStep3-7を行うために

SHIROKANE にはソフトウェア gatk はインストールされていません。

これを使うためには qlogin をしたのち、conda環境で gatk を使えるように準備する必要があります。

- ✓ まず miniconda が使える環境の整備を以下のURLに従い構築してください。

  https://supcom.hgc.jp/internal/mediawiki/Bioconda_%E4%BD%BF%E7%94%A8%E6%96%B9%E6%B3%95
- ✓ 次に Bioconda を使用できる環境を構築してください。

  https://supcom.hgc.jp/internal/mediawiki/Bioconda_%E4%BD%BF%E7%94%A8%E6%96%B9%E6%B3%95
- ✓ またgatkが使用可能なconda環境を立ち上げたのち、 step7で使う samtools を使用可能な環境にしておきます。

コマンド

```
eval "$(~/miniconda3/bin/conda shell.bash hook)"
conda create -n gatk gatk4
conda activate gatk

module use /usr/local/package/modulefiles/
module load samtools/1.9
```

option：SHIROKANEでStep8, 9を行うために

SHIROKANE にはソフトウェア manta はインストールされていません。

これを使うためには qlogin をしたのち、conda環境で manta を使えるように準備する必要があります。

コマンド

```
eval "$(~/miniconda3/bin/conda shell.bash hook)”
conda create -n sv_detect manta
conda activate sv_detect
```

option：SHIROKANEでStep10を行うために

SHIROKANE にはソフトウェア cnvkit はインストールされていません。

これを使うためには qlogin をしたのち、conda環境で cnvkit を使えるように準備する必要があります。

コマンド

```
eval "$(~/miniconda3/bin/conda shell.bash hook)"
conda create -n cnvkit cnvkit
conda activate cnvkit
```

第 Ⅳ 章
データ解析応用
vcfファイルへのアノテーション、公開データベースとの連携やフィルタリング

東京大学　医科学研究所附属病院　血液腫瘍内科　助教
横山和明
東京大学医科学研究所　ヒトゲノム解析センター　健康医療インテリジェンス分野
清水英悟

初めに: 本章で学ぶ事

応用編　第Ⅲ章では1次解析を学び、その結果、参照（対照）配列と異なるSNVのバリアントリスト（vcfファイル) を得ました。

本章では、次のSTEPとして、

- 注釈付きバリアントリストを作成する操作（アノテーション)
- 臨床的意義により絞り込む操作（フィルタリング)　　についての概要を学びます。

入門編　第 Ⅵ 章も参考になりますので、ご確認ください。

Annotation & Filtering workflow

1. HGC SHIROKANE へのログイン (実習)

2. SnpEff & SnpSift 解析環境の設定*

- 2.1 ソフトのインストール
- 2.2 SnpEff アノテーション用 GRCh38.RefSeq のダウンロード
- 2.3 SnpSiftアノテーション用 カスタムデータベースのダウンロード
- 2.4 カスタムデータベースの GRCh38 への座標変換

3. SnpEff & SnpSift によるアノテーション (実習)

*今回の実習では既に実施済みのためユーザーが行う必要はない

ドライバー変異（候補）の絞り込み

4. SnpSift を用いたVCFのフィルタリング(実習)

4.1 技術的フィルタリング
目的:エラーの可能性が低い変異を選択
filter条件: 1+2+3
1.DP: Tumor, normalでdepth ≧ 25
2.MBQ (base quality) ≧ 20
3.AF: TumorのVAF ≧ 0.08
NormalのVAF ≦ 0.04

4.2 生物学的フィルタリング①:
目的:病原性ではない変異の除外
filter条件:1または2の変異を除外
1.ToMMoSNPデータベースでMAFが0.01以上のSNP
2.ClinVarでbenign, likely benignと判断される変異

4.3 生物学的フィルタリング②:
目的: がんにおいて病原性を有すると考えられる変異の選択
filter条件: 1+2 または 3
1.がん関連遺伝子:
-CancerGeneCensus で ”Tier1”
2.病原性有りと予測される変異:
-SnpEffのAnnotation Impactが “High”か”Moderate”
3.変異の遺伝学的エビデンス:
-COSMIC報告サンプル数≧3

5. Filter済みVCF のマージ（実習）

6. VCFからTSVへの整形と 不要なアノテーション列の削除（実習）

7. ドライバー変異の最終絞り込み（実習）

8. 変異のIGVでの確認（実習）

1. HGC SHIROKANE へのログイン（1）

今回の実習ではヒトゲノム解析センター（HGC）のSHIROKANEへアクセスしてゲノム解析を行います。

- SHIROKANEスーパーコンピュータへのログイン
  SHIROKANE5 webサイト：https://gc.hgc.jp/, https://gc.hgc.jp/category/sys-const/shirokane5/
  - ✓ 応用編 Ⅳ 章ではヒトゲノム解析センターのSHIROKANEスーパーコンピュータ上で行います。
  - ✓ SSHクライアントを使用して SHIROKANE（現時点ではSHIROKANE5）にログインします。
  - ✓ *user name* をご自身のユーザー名に置き換えて、下記のコマンドを実行しログインしてください。

```
ssh user_name@slogin.hgc.jp
```

- 解析用のノードへのログイン
  - ✓ ログイン後、SHIROKANE に常備されているコマンド群を使用するには、qloginする必要があります。
  - ✓ 下記のコマンドだと、使用できるメモリは、2.9G となります。

```
qlogin
```

  - ✓ 今回は、JAVA program である SnpEff 、SnpSift を使用します。
  - ✓ 今回のテストdataを使用するに当たり必要なメモリ量を確保する必要があります(defaultで割り当てられる2.9Gのメモリでは足らない可能性があります)。
  - ✓ 下記コマンドを実行して、8Gのメモリを確保した環境の解析用のノードにログインしてください。

```
qlogin -l s_vmem=8G,mem_req=8G
```

1. HGC SHIROKANE へのログイン（2）

- 本実習で解析をスムーズに進めるためのディレクトリ構造の確認
  - ✓ ログイン後、共有フォルダに移動し、下記のコマンドを実行してください。

```
ls -l ~/jinzai4
```

1. 共有フォルダ下に実習でテキストと共に参照する README.txt * （黄色で表示）がある事を確認します。
2. 共有フォルダ下に実習用の bin、data、db のディレクトリがある事を確認します。
3. data の中に bam, sh ファイルがある事を確認します。
4. db の中に、以下のGRCh38 のデータベースがある事を確認します（黄色で表示）
   - dbSNP: “00-All.”
   - GRCh38.p13.RefSeq
   - ClinVar : “clinvar_20210814”
   - COSMIC: “CosmicCodingMuts”
   - ToMMo: “tommo-8.3kjpn”

```
├── README.txt
├── bin
│   ├── SnpSift.jar
│   ├── bgzip
│   ├── db
│   │   ├── 00-All.vcf.gz
│   │   ├── 00-All.vcf.gz.tbi
│   │   ├── JSH_AML.txt
│   │   ├── Census_2022_genes.txt
│   │   ├── Census_allMon_Jun_27_05_34_09_2022.tsv
│   │   ├── CosmicCodingMuts.vcf.gz
│   │   ├── CosmicCodingMuts.vcf.gz.tbi
│   │   ├── GRCh38.p13.RefSeq
│   │   ├── census.awk
│   │   ├── clinvar_20210814.vcf.gz
│   │   ├── clinvar_20210814.vcf.gz.tbi
│   │   ├── conv.sh
│   │   ├── download.sh
│   │   ├── tommo-8.3kjpn-20200831-af_snvall-autosome.vcf.gz
│   │   └── tommo-8.3kjpn-20200831-af_snvall-autosome.vcf.gz.tbi
│   ├── db_list.txt
│   ├── down.sh
│   ├── examples
│   ├── galaxy
│   ├── info_format.awk
│   ├── scripts
│   ├── snpEff.config
│   ├── snpEff.jar
│   └── tabix
├── data1
│   ├── JinzaiCOLO_BL.bam
│   ├── JinzaiCOLO_BL.bam.bai
│   ├── JinzaiCOLO_T.bam
│   ├── JinzaiCOLO_T.bam.bai
│   ├── anno.sh
│   ├── filter.sh
│   └── mutect.sh
└── data2
    ├── JinzaiAML_BM.bam
    ├── JinzaiAML_BM.bam.bai
    ├── JinzaiAML_SW.bam
    ├── JinzaiAML_SW.bam.bai
    ├── anno.sh
    ├── filter.sh
    └── mutect.sh
```

*このテキストでは実習で使うコマンドの概要を記載していますが、README.txtには、実際のファイル名等を反映した詳細な記載があり、本実習では、README.txtの内容をCOPY & PASTEして進めます。
*記述しているコマンドは、実習で使用するデータ（jinzai）がユーザーのホームディレクトリにあることを前提にしています。

2. SnpEff & SnpSift 解析環境の設定

SnpEff & SnpSift はSNVのアノテーション& フィルタリングで用いられる代表的なフリーソフトの1つです。

解析環境の設定:

ダウンロードサイト：http://pcingola.github.io/SnpEff/

- ✓ SnpEffは、ユーザが指定した SnpEff のデータベースで、 SNVのアノテーションを行うツールです。
- ✓ 一方、SnpSift は、ユーザが用意した任意のカスタムデータベースを利用したアノテーション追加や、フィルタリングに用いるツールです。
- ✓ 本章では、両ツールをセットで用います。
- ✓ SnpEff & SnpSift を用いてアノテーションやフィルタリングコマンドを実施すると、VCFのヘッダ行にソフトのversion情報、コマンド操作のログ情報が書き込まれます。また、INFO列にはアノテーション情報が書き込まれます。

vcfのヘッダ行に付与されるコマンドログの例:

```
##SnpSiftCmd="SnpSift Annotate /home/user/jinzai4/bin/db/tommo-8.3kjpn-20200831-af_snvall-autosome.vcf.gz
/home/user/jinzai4/data1/test.GRCh38.vcf"
```

SNVのINFO列に付与されるアノテーション例:

```
ANN=T|missense_variant|MODERATE|……"
```

- ✓ ANN=が SnpEff & SnpSift がSNVに付与したアノテーションです。| （パイプ）区切りで書かれた情報については、ヘッダ情報の”INFO=<ID=”以下に書かれています。
- ✓ アノテーションにはどの遺伝子、あるいは転写産物上のSNVか、SNVによってどんなアミノ酸変化が生じるか、機能的インパクトなどが出力されます。

2.1 ソフトのインストール

SnpEff & SnpSift を使用するための準備をします。

- SnpEff は下記のサイトからダウンロードする必要があります。

  SnpEff ダウンロードサイト： https://pcingola.github.io/SnpEff/download/

  - ✓ SnpEff は、JAVA programで作成されています。
  - ✓ JAVA program で作成されている SnpEff を使用するに当たり、SnpEff 操作に必要なメモリの量を指定します。
  - ✓ 下記のコマンドを実行して JAVA heap memory を最大6G、最小2Gに設定してください。

```
export JAVA_TOOL_OPTIONS='-XX:+UseSerialGC -Xmx6g -Xms2g'
```

2.2 & 2.3 SnpEff/SnpSift アノテーションデータのダウンロード

SnpEff / SnpSift を使用するためにアノテーションデータのダウンロードをする必要があります。

- SnpEff アノテーションデータベースの確認
  - ✓ SnpEff でアノテーションできるデータベースは、SnpEff でダウンロードして使用します。
  - ✓ 使用できるデータベースは下記のコマンドで確認することができます。

```
java -jar ~/jinzai4/bin/snpEff.jar databases
```

  - ✓ Homo_sapiens 用のデータベースは下記のコマンドで確認することができます。

```
java -jar ~/jinzai4/bin/snpEff.jar databases | grep "Human¥|Homo_sapiens"
```

- GRCh38.RefSeq のダウンロード*
  - ✓ GRCh38.RefSeq をダウンロードするには、SnpEff を使用します。

```
java -jar ~/jinzai4/bin/snpEff.jar download GRCh38.p13.RefSeq
```

  - ✓ ダウンロードした GRCh38.RefSeq でアノテーションするコマンドは下記になります。

```
java -jar ~/jinzai4/bin/snpEff.jar GRCh38.p13.RefSeq input.vcf > output.vcf
```

*今回の実習ではGRCh38データベースはダウンロードされ、設定されていますので実行する必要はありません。
*記述しているコマンドは、実習で使用するデータ（jinzai）がユーザーのホームディレクトリにあることを前提にしています。

2.3 SnpSiftアノテーション用 カスタムデータベースのダウンロード

SnpSift でアノテーションをするためにはデータベースをダウンロードする必要があります* 。

- SnpSift でのアノテーション:

  今回使用するデータベースは下記になります* 。
  - 東北メディカルメガバンク機構 ToMMo 8.3KJPN Allele Frequency Panel (v200831)
  - ClinVar
  - COSMIC
  - CancerCensus

✓ コマンドラインで `curl` でダウンロードする場合は、接続している SHIROKANE 上に 直接ダウンロードできます（実習では先4つ全てダウンロード済み）。

```
curl -O [ データベース URL]
```

✓ ウェブブラウザからダウンロードする場合は、ダウンロードしたファイルを SHIROKANE にコピーする必要があります。

```
scp [ダウンロードしたファイル] [username]@slogin.hgc.jp:[スパコン上でのフォルダー]
```

*今回の実習では上記データベースは全てダウンロードされ、設定されています。
自習で環境設定を行う場合は応用編テキスト巻末にあるAppendixを参照してください。

3. SnpSiftによるアノテーション（1）

コマンド入力で SnpSift を使ったアノテーションを実行できます。

- SnpSift を使ったアノテーションのコマンド
  - ✓ vcfファイル形式のデータベースを使って、変異のvcfファイルにアノテーションを付けるには、下記のコマンドを使用します* 。

```
java -jar [SnpEffフォルダーの絶対パス]/SnpSift.jar annotate annotation_db.vcf.gz input.vcf > output.vcf
```

  - ✓ 今回の実習の環境だと、~/jinzai/bin に SnpEff がインストールされているので、下記のコマンドで実行できます* 。

```
java -jar ~/jinzai4/bin/SnpSift.jar annotate annotation_db.vcf.gz input.vcf > output.vcf
```

  - ✓ データベースとして使用するファイルは、vcfファイルを bgzip で圧縮し、tabix で index ファイルを作成したものになります。

    tabix、bgzip は、~/jinzai4/bin にあります。

    コマンド bgzip 、tabix については下記を参照してください。
    https://www.htslib.org/doc/tabix.html

```
bgzip annotation_db.vcf
tabix -p vcf annotation_db.vcf.gz
```

*コマンド中のfile名の表記は実際のものとは異なります。実習では、README.txtの内容をCOPY & PASTEして使用してください。

3. SnpSiftによるアノテーション（2）

SnpSift を用いて ToMMo、Clinvar、COSMICのアノテーションを順に付与してみましょう

- 東北メディカルメガバンク機構 ToMMo 8.3KJPN Allele Frequency Panel (v200831)
  - ✓ ダウンロード後、**GRCh38リファレンスに変換***したファイルを使って、SnpSift でアノテーションします*2。

```
java -jar [SnpEffフォルダーの絶対パス]/SnpSift.jar annotate tommo-8.3kjpn-20200831-af_snvall-autosome.vcf.gz input.vcf > output.vcf
```

- ClinVar
  - ✓ ダウンロードしたファイルを使って、SnpSift でアノテーションします*2 。

```
java -jar [SnpEffフォルダーの絶対パス]/SnpSift.jar annotate clinvar_20210814.vcf.gz input.vcf > output.vcf
```

- COSMIC (Catalogue of Somatic Mutation in Cancer)
  - ✓ ダウンロードしたファイルを使って、SnpSift でアノテーションします*2 。

```
java -jar [SnpEffフォルダーの絶対パス]/SnpSift.jar annotate CosmicCodingMuts.vcf.gz input.vcf > output.vcf
```

*2. コマンド中のfile名の表記は実際のものとは異なります。実習では、README.txtの内容をCOPY & PASTEして使用してください。

4. SnpSiftを用いたVCFのフィルタリング（1）

- アノテーション結果を確認してみます。

```
cd data1
ls
```

  - ✓ アノテーションの結果、3 つのファイルが出力されています。
    1. バリアントに関する要約統計を含むサマリーHTMLファイル
    2. アノテーション済みvcfファイル
    3. 遺伝子ごとのバリアントタイプ数をまとめたテキストファイル

- アノテーション済みvcfファイルを見てみましょう* 。

* 実習では、README.txtに記載されたコマンドに従ってください。

```
less test.GRCh38.p13.RefSeq.vcf
```

  - ✓ vcfファイルのヘッダに以下のような情報が書き込まれています。

```
##INFO=<ID=ANN,Number=.,Type=String,Description="Functional annotations: 'Allele | Annotation
| Annotation_Impact | Gene_Name | Gene_ID | Feature_Type | Feature_ID | Transcript_BioType |
Rank | HGVS.c | HGVS.p | cDNA.pos / cDNA.length | CDS.pos / CDS.length | AA.pos / AA.length |
Distance | ERRORS / WARNINGS / INFO' ">
```

  - ✓ 「q」で閲覧を終了します。
  - ✓ 変異のINFO列に下記のような情報が書き込まれます。

```
ANN=T|missense_variant|MODERATE|BRAF|BRAF|transcript|NM_001354609.2|protein_coding|15/19|c.17
99T>A|p.Val600Glu|2025/9687|1799/2304|600/767||,...
```

  - ✓ ANN=が付与されたアノテーションです。 | （パイプ）区切りで書かれた情報については、ヘッダ情報の”INFO=<ID=”以下に書かれています。

4. SnpSiftを用いたVCFのフィルタリング（2）

次にhtml summary(snpEff_summary.html)を見てみましょう。

- “Number of effects by type and region”のカラムを見てみます.
  変異のタイプと、変異が観察されたゲノム領域(割合)がわかります。

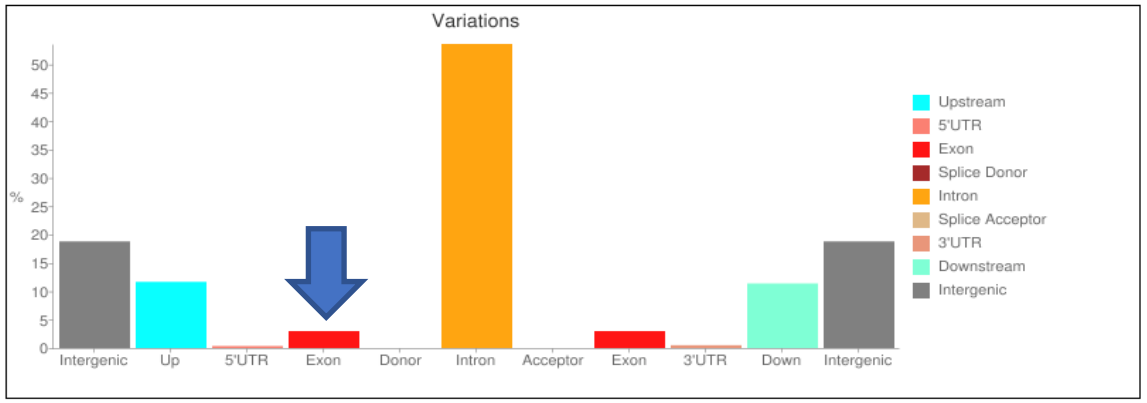
変異のタイプ　　Number of effects by type and region　　**ゲノム領域**

Type

Type (alphabetical order)	Count	Percent
3_prime_UTR_variant	33	0.502%
5_prime_UTR_variant	23	0.35%
disruptive_inframe_deletion	4	0.061%
downstream_gene_variant	748	11.389%
intergenic_region	1,234	18.788%
intragenic_variant	45	0.685%
intron_variant	3,521	53.608%
missense_variant	62	0.944%
non_coding_transcript_exon_variant	113	1.72%
splice_region_variant	2	0.03%
synonymous_variant	17	0.259%
upstream_gene_variant	766	11.663%

Region

Type (alphabetical order)	Count	Percent
DOWNSTREAM	748	11.392%
EXON	196	2.985%
INTERGENIC	1,234	18.794%
INTRON	3,519	53.594%
SPLICE_SITE_REGION	2	0.03%
TRANSCRIPT	45	0.685%
UPSTREAM	766	11.666%
UTR_3_PRIME	33	0.503%
UTR_5_PRIME	23	0.35%



Exon領域の変異は全体のごく僅か(矢印)です。

- “Number of effects by impact”のカラムを見てみます.
  変異のインパクト予測結果の集計があります。

- SnpEffでは、変異が与えるインパクトを、Sequence Ontology*1 で定めた規則を元にインパクトが大きい順に HIGH 、 MODERATE、 LOW、MODIFIER の4段階*2に分類します。

*1. Sequence Ontology : http://www.sequenceontology.org/
*2. https://pcingola.github.io/SnpEff/adds/VCFannotationformatv1.0.pdf

Number of effects by impact

Type (alphabetical order)	Count	Percent
LOW	19	0.289%
MODERATE	66	1.005%
MODIFIER	6,481	98.705%

インパクトによる変異の機能予測結果は殆どがMODIFIER（赤枠）である事がわかります。

4. SnpSiftを用いたVCFのフィルタリング（3）

Filteringの基本構文:

SnpSiftを利用して、アノテーションがついたvcfファイルのデータをフィルタリングできます。

SnpSiftのフィルタリングは以下のコマンドで実行できます。

```
cat [vcf file] | java -jar SnpSift.jar filter "( フィルタリング条件 )" > [出力先vcf file]
```

SET option:

SnpSiftのSET optionを利用して、参照fileを別途指定した、vcfファイルのフィルタリングができます*。

-s|--set <file> ：
遺伝子名の格納されたファイル<file>を指定して、
vcfファイルから該当遺伝子 変異を取り出す事ができます。

以下はCensus_2022_genes.txtfileを指定して、 test.anno.1.2.3.4.vcfから該当遺伝子変異を取り出すコマンド例です。

```
＃Set option でがん関連遺伝子の変異を取り出す
java -jar ~/jinzai4/bin/SnpSift.jar filter -s ~/jinzai4/bin/db/Census_2022_genes.txt "ANN[*].GENE in SET[0]"
~/jinzai4/data2/test.anno.1.2.3.4.vcf > ~/jinzai4/data2/test.anno.1.2.3.4.5.vcf
```

*https://pcingola.github.io/SnpEff/ss_filter/#snpeff-ann-fields

4.1 技術的フィルタリング（1）

SnpSiftを利用して、vcfファイルの技術的フィルタリングをまず行なってみましょう。

- 技術的フィルタリング（Technical Filter）とは

  vcfファイルには、シークエンスエラー、変異コールのエラーなどの技術的エラー*による偽陽性のバリアントが含まれているため、これらの偽陽性のバリアントをfilter outして、エラーの可能性が低いバリアントを選択する操作が必要となります。この操作を技術的フィルタリングと呼びます。

  * 技術的エラー: シークエンスエラーと変異コールエラーの事。シークエンスエラーには、検体の種類（FFPE検体を含むか）や、ライブラリ調整等の前調整課程（PCR増幅の有無）、シークエンサーの機種とシークエンス原理、シークエンス試薬の種類などが関与します。変異コールエラーには、pipelineの検出原理や、referenceゲノム、パラメータの設定などが影響します。ユーザは、解析するvcf fileについて、これらの背景を考慮しながら、カットオフ値やフィルタリングに使用する項目を毎回指定する必要があります。

- 以下はエラーの可能性が低いバリアントを選択する技術的フィルタリングの例です。
  フィルタリング条件：
  - 1. Control, tumor, normalでのカバレッジ（DP） 25以上
  - 2. Control, tumor,normalでのmedian base quality(MBQ)*がそれぞれ20以上(つまりエラー率1%以下)
  - 3. Control “GEN[0]”のVAF が0.04以下で、Tumor “GEN[1]”のVAFが0.08以上

  の変異を選択。

  * MBQ：https://sites.google.com/a/broadinstitute.org/legacy-gatk-forum-discussions/2020-01-07-2019-07-10/24425-how-to-tell-whether-a-variant-is-a-sequence-error-or-real

上記の技術的フィルタリング（条件1かつ条件2かつ条件3)をコマンドで表すと“(フィルタリング条件)“は以下の通りです。

```
"(GEN[*].DP >= 25 ) & (exists MBQ[*] ) & ( MBQ[*] >= 20 ) & ( GEN[0].AF <= 0.04 ) & ( GEN[1].AF >= 0.08 )"
```

条件1 **条件2** **条件3**

1番サンプルならGEN[0] 、全サンプルならGEN[*]で表せます。 & で（条件A）and（条件B）を表せます。

4.1 技術的フィルタリング (2)

技術的filteringを実行してみます

```
#技術的filtering
cat ~/jinzai4/data1/test.anno.vcf | java -jar ~/jinzai4/bin/SnpSift.jar filter "( GEN[1].DP >= 25 ) & (exists MBQ[*] ) & ( MBQ[*] >= 20 )
&( GEN[0].AF <= 0.04 ) & ( GEN[1].AF >= 0.08 )"  > ~/jinzai4/data1/test.anno.1.vcf
```

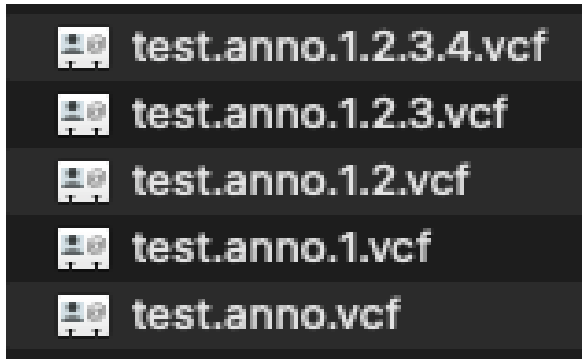
変異の個数を確認してみます。

```
# 変異の個数をcount
grep -v '^#' ~/jinzai4/data1/test.anno.1.vcf | sort -u | wc -l
```

変異が減っているのが確認できました

この実習では、filtering実行毎に、test.anno.* vcf file (*は任意の文字)を作成します(図1)。
毎回上記の構文を実行して、filtering実行毎に、変異のカウント操作を実行するのは面倒です。
以下の様にFor構文を使ったスクリプトを使えば、変異の個数カウント操作を一気に実行できます(図2)。

図1. 出力される vcf file (例)



input

スクリプト

```
# 変異の個数を繰り返しcount
#
cd ~/jinzai4/data1
files=`ls -v test.anno.vcf test.anno.*.vcf`
for file in $files
do
    count=`grep -v ^# $file | wc -l`
    echo $file $count
done
```

output

図2. カウント操作実行結果 (例)

```
#COLO
/home/eigosi/jinzai4/data1/test.anno.vcf 2402
"( GEN[1].DP >= 25 ) & (exists MBQ[*] ) & ( MBQ[*]
/home/eigosi/jinzai4/data1/test.anno.1.vcf 92
" (( exists AF) & (AF < 0.01) | ! (exists AF) )"
/home/eigosi/jinzai4/data1/test.anno.1.2.vcf 89
-n "(( CLNSIG has 'Likely_benign' ) | ( CLNSIG has
/home/eigosi/jinzai4/data1/test.anno.1.2.3.vcf 89
"(ANN[*].IMPACT == 'HIGH' | ANN[*].IMPACT == 'MODER
/home/eigosi/jinzai4/data1/test.anno.1.2.3.4.vcf 1
```

4.2 生物学的フィルタリング①（1）

SnpSiftを利用して、変異の生物学的機能に着目した生物学的フィルタリングを行います

SNPデータベースにより、一般的な変異（病原性ではないSNPなど）を除外することができます。

- SNPデータベースを用いて、病原性ではないSNPを除外する生物学的フィルタリング（Biological Filter）を行なってみましょう。
  - ✓ この実習では日本人の代表的なSNPデータベースのToMMo[*1]を使います。
    - ➢ 世界的に有名な多型 データベース収集プロジェクトとしては他に、1000genome[*2], ESP[*3], gnomAD[*4.] などがあり、横断的検索データベースとしてNCBI dbSNP[*5]等があります。
  - ✓ ToMMo を用いて、日本人でマイナーアレル頻度（MAF）1%以上を一般的な変異（病原性ではないSNP）と見做してFilter outします。
    - ➢ 先ほどの「3. SnpSiftによるアノテーション」STEPのカスタムデータベースアノテーションで、tommo:AF列が追加され、MAFが記載されていますのでこれを使います。

ToMMoでMAF1%未満の変異をpick upする為の“(フィルタリング条件)“は以下の通りです。

```
" (( exists AF) & (AF < 0.01) | ! (exists AF) )"
```

`!` でnotを表せます*6。

*1. ToMMo : https://jmorp.megabank.tohoku.ac.jp/202102/
*2. 1000genome : https://www.internationalgenome.org/data-portal/data-collection/grch38
*3. ESP : https://evs.gs.washington.edu/EVS/
*4. gnomAD : https://gnomad.broadinstitute.org
*5. NCBI dbSNP : https://www.ncbi.nlm.nih.gov/clinvar/docs/faq/
*6. `-n` を使って、`-n “ (exists AF) & (AF >= 0.01)”`とfilter条件を記載する事も可能です。

4.2 生物学的フィルタリング①（2）

- ✓ ClinVarで clinical significance が”benign”または”likely benign”とアノテーションされた病原性ではない変異をFilter outします。
  - ➢ ClinVarのカスタムデータベースアノテーションで、clinvar:CLNSIGの clinical significance 欄が追加されていますので、これを使います。

clinical significance が”benign”または”likely benign”のfilter条件とマッチしないものを取り出すための“(フィルタリング条件)“は以下の通りです。

```
 -n "( CLNSIG has 'Likely_benign’ ) | ( CLNSIG has 'benign’ )"
```

| （パイプ）で（条件A）or（条件B）を表せます。
Filter条件の前に -n を記載する事でfilter条件とマッチしないものを取り出せます。

4.2 生物学的フィルタリング①の実行

技術フィルタリング後のfile “test.anno.1.vcf” に生物学的フィルタリング①を実行してみましょう

```
#生物学的フィルタリング①_ToMMoでMAF1%以上の病原性ではないSNPを除外
cat ~/jinzai4/data1/test.anno.1.vcf | java -jar ~/jinzai4/bin/SnpSift.jar filter " (( exists AF) & (AF < 0.01) | ! (exists AF) )"  >
~/jinzai4/data1/test.anno.1.2.vcf

# 生物学的フィルタリング① _ClinVarで病原性ではない変異を除外
cat ~/jinzai4/data1/test.anno.1.2.vcf | java -jar ~/jinzai4/bin/SnpSift.jar filter -n "(( CLNSIG has 'Likely_benign' ) | ( CLNSIG has 'benign'))
"  > ~/jinzai4/data1/test.anno.1.2.3.vcf
```

変異の個数を確認してみます*。

```
# 変異の個数をcount
#
cd ~/jinzai4/data1
files=`ls -v test.anno.vcf test.anno.*.vcf`
for file in $files
do
    count=`grep -v ^# $file | wc -l`
    echo $file $count
done
```

少し変異が減っています。

4.3 生物学的フィルタリング②（１）変異の病原性

変異の病原性の有無の予測（機能予測アルゴリズム）に基づいた生物学的フィルタリング:

- 次に変異の病原性の有無の予測（機能予測アルゴリズム）に基づいた生物学的フィルタリングをしてみましょう。
  アミノ酸の変化のない同義変異、タンパク質をコードしない領域の変異などの機能的に病原性を有するとは考えにくい変異などを除外しましょう。

  ✓ この目的の為、SnpEff *1 のアノテーションとして付与されるAnnotation Impact をフィルタリングに使用します。
  ✓ 病原性を有すると予測される変異は HIGH、MODERATE を中心に考えます。
  ✓ SnpEff では、リファレンスとなるmRNA(ENST番号)毎にアノテーションがつくため、1つの変異に複数のアノテーションがつきます。
  ➢ そこでいずれかのアノテーションで Annotation_Impact がHIGHかMODERATEの変異を取り出します。

Annotation Impact がHIGHかMODERATEを取り出す“(フィルタリング条件)“は以下の通りです。

```
"(ANN[*].IMPACT == 'HIGH' ) | (ANN[*].IMPACT == 'MODERATE')"
```

| （パイプ）で（条件A）or（条件B）を表せます。
ANNに[*] とつけると「すべてのアノテーションの」という意味になります。

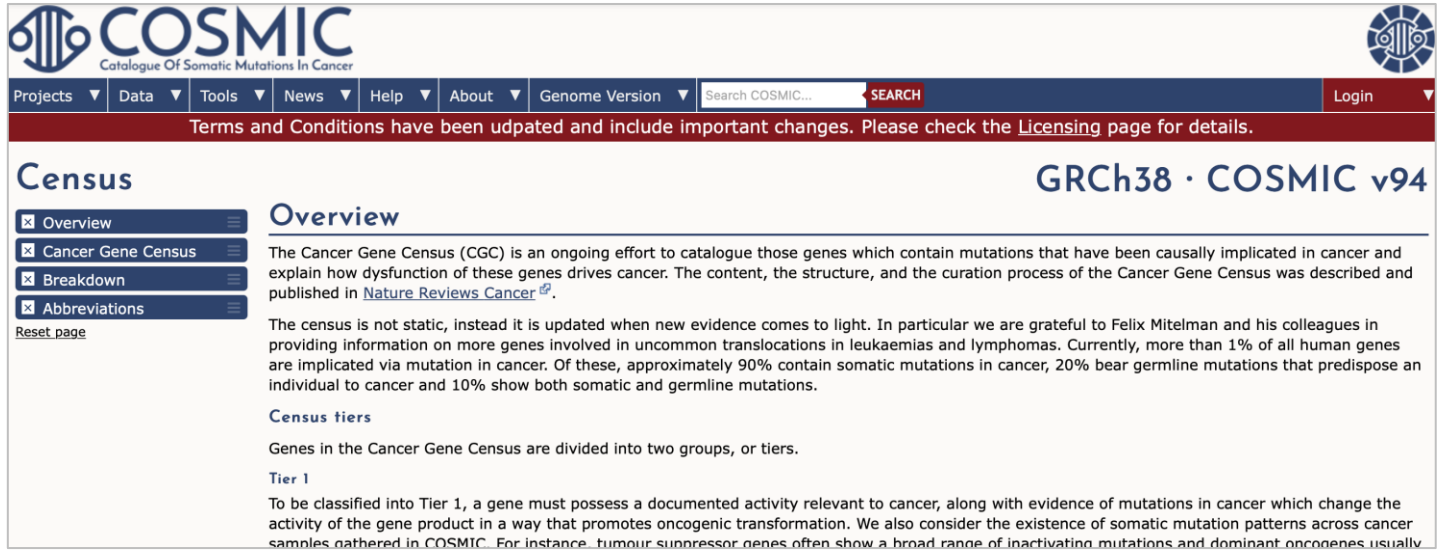
*1 他に変異によって起こるアミノ酸置換の影響がタンパク質の影響についてスコア化するソフトウェアとしては、SIFT, Polyphen2, Mutation taster, Proveanなどがあります。例えばSIFTは0に近いほど、Polyphenは1に近いほど生物学的にインパクトを与える変異とされます。

4.3 生物学的フィルタリング②（2）がん関連遺伝子

CGC Genes[*1][*2] （がん関連遺伝子）に注目した生物学的フィルタリング。

- 次にCancer Gene Census (CGC) というプロジェクトで挙げられた遺伝子に注目してみます
- CGCは、専門キュレーターにより、論文の情報から、がんの原因となる変異が報告されている遺伝子をがんの特性(がんのホールマーク)[*3]としてまとめたものです。特にTier1に分類されている遺伝子は、機能的かつ遺伝学的なエビデンスがあるため重要です。

https://cancer.sanger.ac.uk/census



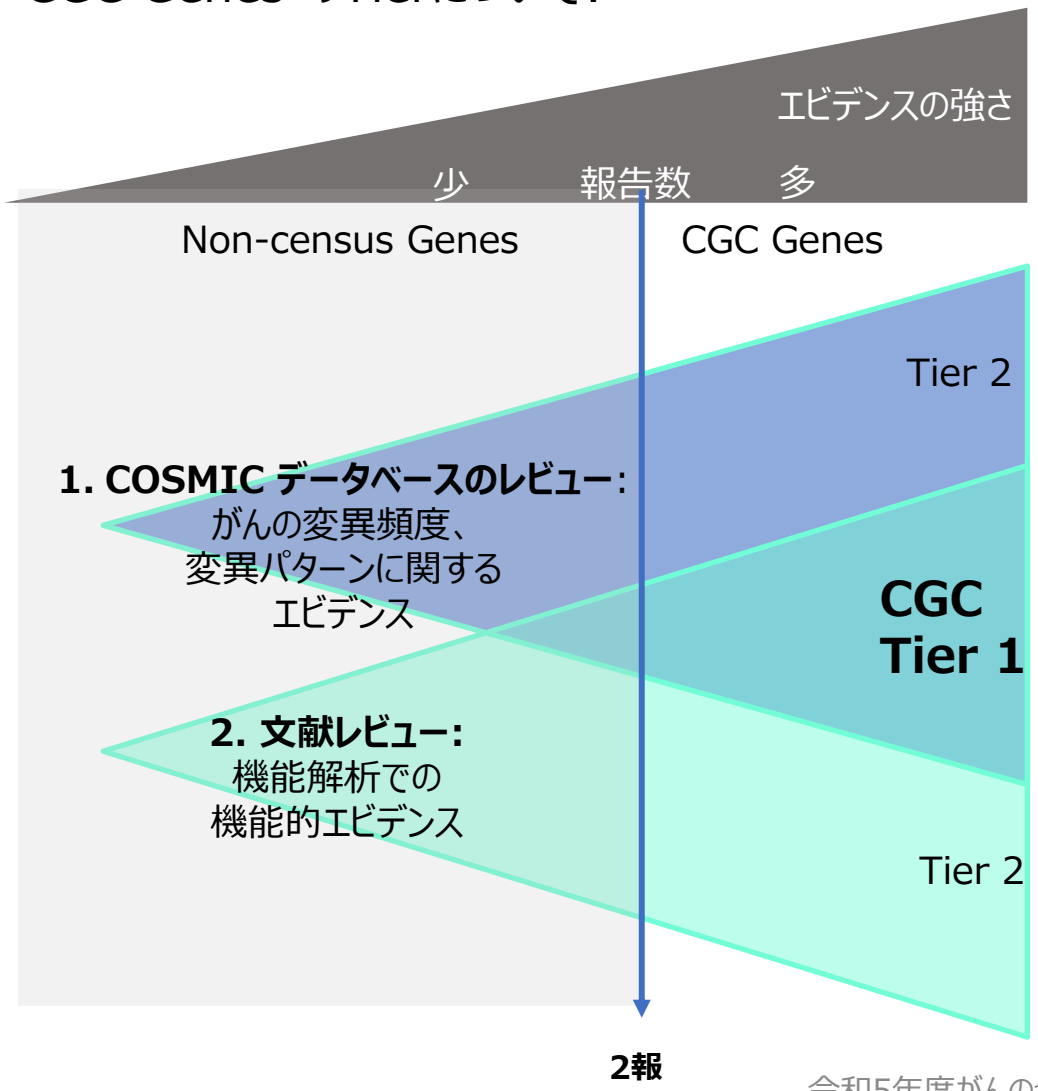
*1. CGC genes : https://cancer.sanger.ac.uk/census

*2. https://pcingola.github.io/SnpEff/adds/VCFannotationformat_v1.0.pdf

*3. Hanahan D, Weinberg Robert A. Hallmarks of Cancer: The Next Generation. *Cell*. 2011; 144:646– 674.

4.3 生物学的フィルタリング②（3）がん関連遺伝子

CGC Genes のTierについて:



Minimum evidence for CGC Genes [*1]

1. 少なくとも1種類のがんにおける変異頻度の増加や、特徴的な変異のパターン[*2]が、異なる研究グループからに2報以上報告がある事
2. がんの特性（hallmarks of cancer[*3]）に関わるという実験結果に基づいた機能的なエビデンスを示す論文が2報以上ある事

Tier1: 1 かつ 2 を満たす遺伝子
Tier2: いずれかのみ満たす遺伝子

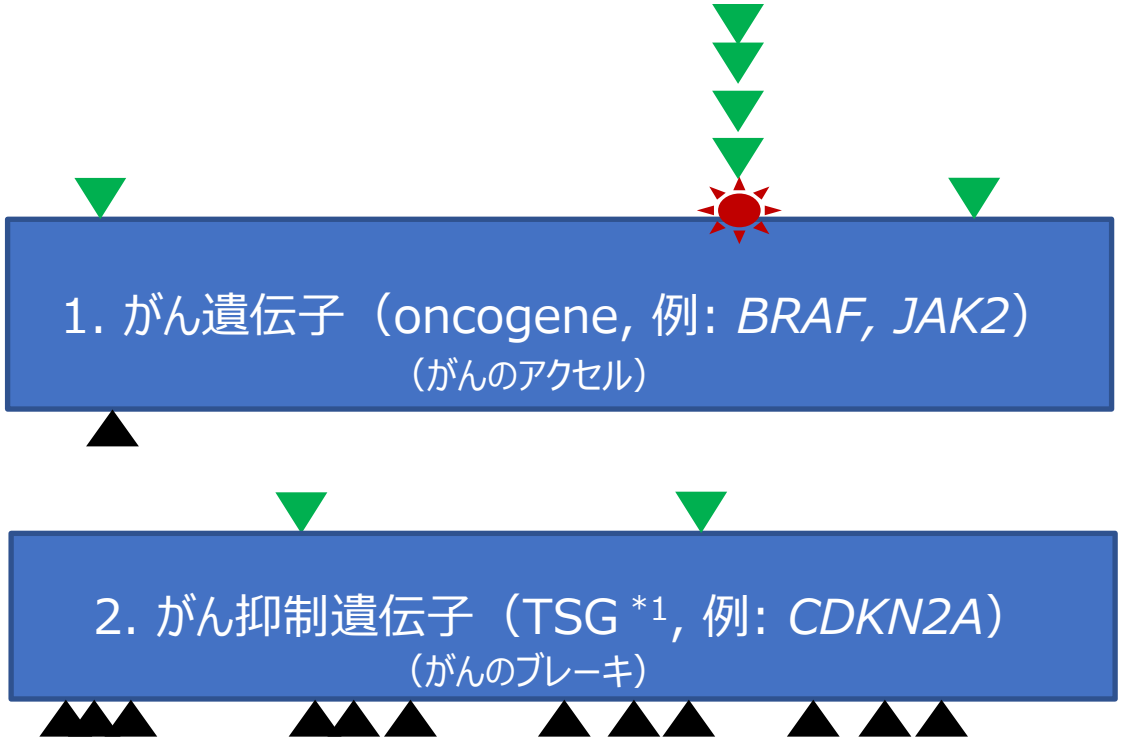
*1. Nat Rev Cancer. 2018 Nov; 18(11): 696–705. doi: 10.1038/s41568-018-0060-1

*2. がん遺伝子や、がん抑制遺伝子での典型的な変異分布のパターン等（応用編テキスト巻末Appendix参照）

*3. *Cell.* 2011; 144:646– 674.

4.3 生物学的フィルタリング②（4）がん関連遺伝子

CGC Genes で考慮される体細胞変異の分布パターン:



Hotspot有り
殆どがミスセンスかin-frame indel
（gain of function変異が想定される）

Hotspot無し
殆どがTruncatingな変異
（loss of function変異が想定される）

ミスセンス 変異、in-frame indel 変異
Truncating 変異*2 (ナンセンス、フレームシフト、スプライスサイト変異など)
Hot Spot

*1. TSG：Tumor Suppressor Gene，がん抑制遺伝子
*2. Truncating変異：タンパク質合成が途中で中断される変異

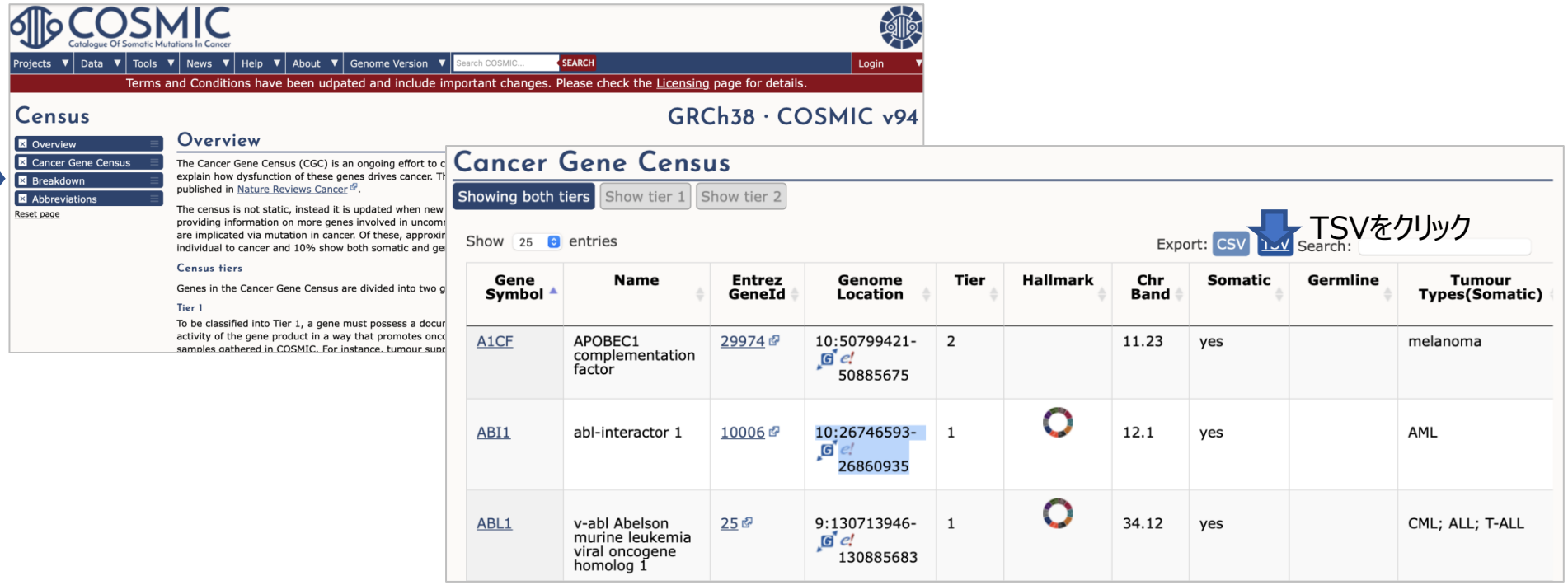
4.3 生物学的フィルタリング②（5）がん関連遺伝子

- CancerCensusのフィルタリング準備を行います

  下記のウェブサイトからファイルをダウンロードしてください。

https://cancer.sanger.ac.uk/census

Cancer Gene Census をクリック



Census_allTue Jun 28 09 59 01 2022.tsv(注: 2022年6月時点)をダウンロード （実習ではダウンロード済み）

4.3 生物学的フィルタリング②（6）がん関連遺伝子

Census_all Tue Jun 28 09 59 01 2022.tsvをExcelで開いてみます

CGCの遺伝子listが得られます

遺伝子名　Tier　生殖細胞系列　がん種　機能　変異タイプ

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	Gene Symbo	Name	Entrez Genel	Genome Loc	Tier	Hallmark	Chr Band	Somatic	Germline	Tumour Type	Tumour Type	Cancer Syndi	Tissue Type	Molecular Ge	Role in Canc	Mutation Typ	Translocatio	Other Germli	Other Syndro	Synonyms
2	A1CF	APOBEC1 co	29974	10:50799421	2		11.23	yes		melanoma			E		oncogene	Mis				29974,A1CF,
3	ABI1	abl-interacto	10006	10:26746593	1	Yes	12.1	yes		AML			L	Dom	TSG, fusion	T	KMT2A			10006,ABI-1,
4	ABL1	v-abl Abelsor	25	9:130713946	1	Yes	34.12	yes		CML, ALL, T-ALL			L	Dom	oncogene, fu	T, Mis	BCR, ETV6, NUP214			25,ABL,ABL1
5	ABL2	c-abl oncoge	27	1:179099327	1		25.2	yes		AML			L	Dom	oncogene, fu	T	ETV6			27,ABL2,ABL
6	ACKR3	atypical chen	57007	2:236569641	1	Yes	37.3	yes		lipoma			M	Dom	oncogene, fu	T	HMGA2			57007,ACKR3
7	ACSL3	acyl-CoA syn	2181	2:222860934	1	Yes	36.1	yes		prostate			E	Dom	fusion	T	ETV1			2181,ACS3,A
8	ACSL6	acyl-CoA syn	23305	5:131949973	2		31.1	yes		AML, AEL			L	Dom	fusion	T	ETV6			23305,ACS2,
9	ACVR1	activin A rece	90	2:157736444	1	Yes	24.1	yes		DIPG			O	Dom	oncogene	Mis		yes	Fibrodysplasi	90,ACVR1,AC
10	ACVR1B	Activin A Rec	91	12:51951701	1		13.13	yes		pancreatic cancer			E		TSG	D, F, Mis, N				91,ACVR1B,A
11	ACVR2A	activin A rece	92	2:147844517	1	Yes	23.1	yes		large intestine carcinoma, stomach carci			E	Rec	TSG	Mis, N, F				92,ACTRII,AC

遺伝子名の欄をコピー＆ペーストしてtext形式で保存します(Census_2022_genes.txt :実習では作成済み)。

Census_2022_genes.txt

```
1  A1CF
2  ABI1
3  ABL1
4  ABL2
5  ACKR3
6
```

(実習ではダウンロード済み)

4.3 生物学的フィルタリング②（7）

（続き）がん関連遺伝子で、病原性を有すると予測される変異を取り出す:

STEP4.1の技術的フィルタリングと、STEP4.2.と4.3.のフィルタリング条件（=CGCに登録のあるがん関連遺伝子について病原性を有すると考えられるバリアントを取り出す）コマンドは以下の通りです*。

*実習では、README.txtに記載されたコマンドに従ってください。

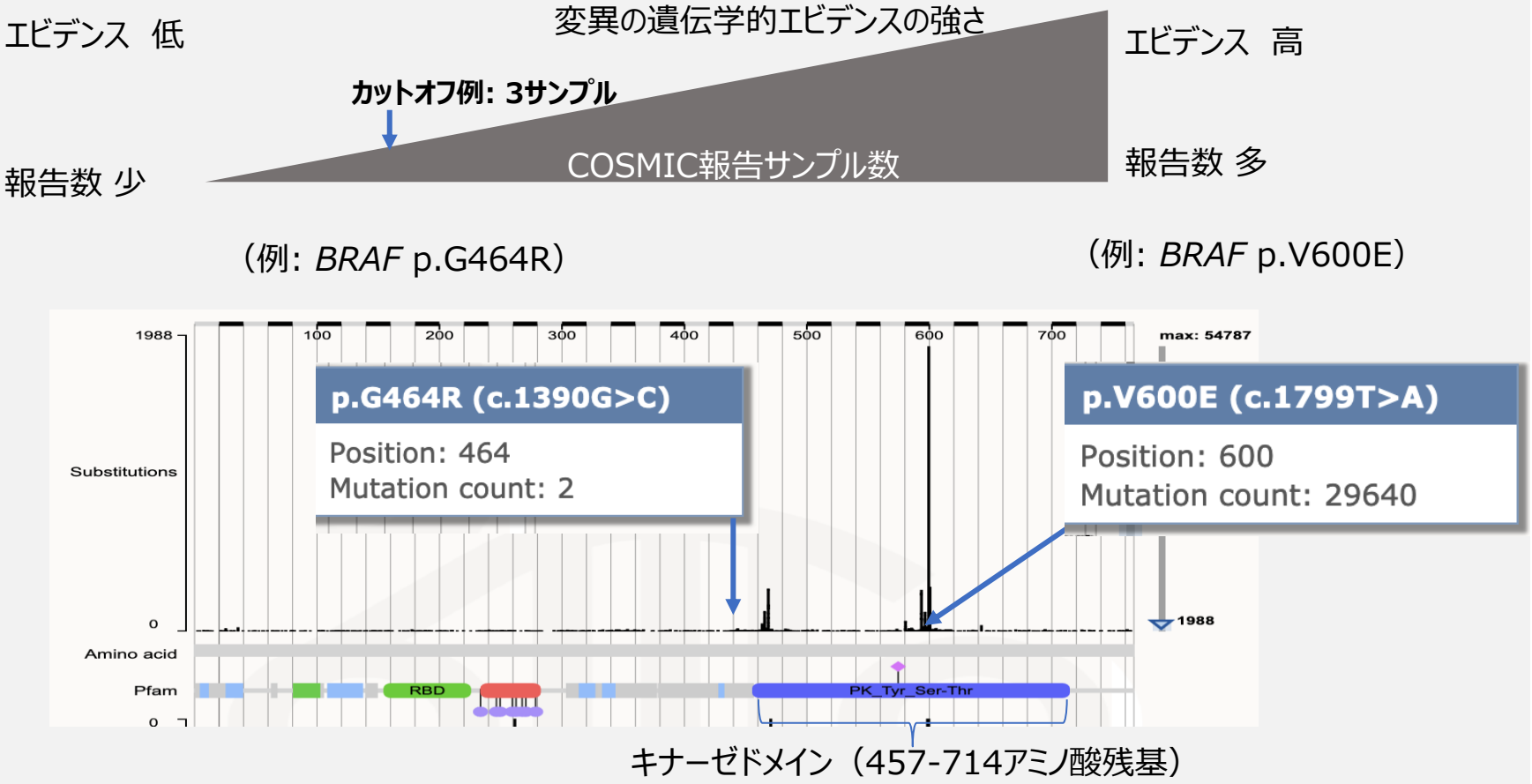
```
＃生物学的フィルタリング②_病原性を有すると予測される変異を取り出す
cat ~/jinzai4/data1/test.anno.1.2.3.vcf | java -jar ~/jinzai4/bin/SnpSift.jar filter
 "(ANN[*].IMPACT == 'HIGH' | ANN[*].IMPACT == 'MODERATE')" >
 ~/jinzai4/data1/test.anno.1.2.3.4.vcf

＃生物学的フィルタリング②_Set option でがん関連遺伝子のうち、病原性を有すると予測される変異を取り出す
cat ~/jinzai4/data1/test.anno.1.2.3.4.vcf | java -jar ~/jinzai4/bin/SnpSift.jar filter -s
 ~/jinzai4/bin/db/Census_2022_genes.txt "ANN[*].GENE in SET[0]" >
 ~/jinzai4/data1/test.anno.1.2.3.4.5.vcf
```

4.3 生物学的フィルタリング②（8）

COSMIC報告サンプル数と変異の遺伝学的エビデンスの強さについて（*BRAF*遺伝子を例に）：

BRAF p.V600Eは極めて多数の登録検体がある、機能的に重要なキナーゼドメインに位置するhotspot変異ですが、
p.G464Rはhotspotから離れた、2検体しか登録がない変異であり、遺伝学的なエビデンスが乏しい変異です。



4.3 生物学的フィルタリング②（9）

COSMICでエビデンスのある変異を取り出す:

前述のように、COSMICで登録がある＝当該バリアントにエビデンスがある、とは必ずしもいえません。
バリアントのエビデンスの高さは、COSMICでの登録サンプル数、遺伝子名、バリアントの局在場所（機能的に重要なタンパク質のドメインなのか、ホットスポットの近傍なのか）、バリアントの種類（サイレント、ミスセンス、truncating等）、
検体の内訳（細胞株か、臨床検体なのか）、疾患名、文献報告の内容（変異が実際にサンガー法などの別法で確認されているかのか）、複数の報告者により報告されているか、などを目安に総合的に判断する必要があります。

中でも登録サンプル数は、解析時点における、バリアントの遺伝学的エビデンスの高さを反映すると考えられます。

COSMICでエビデンスのある変異を取り出す:

例えばCOSMIC報告サンプル数≧3をフィルタリング条件として使うと

COSMIC報告サンプル数≧3のバリアントを取り出す“(フィルタリング条件)“は以下の通りです。

```
"( ( CNT[0] >= 3 ) | (CNT[0] + CNT[1] >= 3 ) | (CNT[0]+CNT[1]+CNT[2]>=3 ) )"
```

*実習では、README.txtに記載されたコマンドに従ってください。

4.3 生物学的フィルタリング②（10）

技術フィルタリング＋生物学的フィルタリング①の後に、COSMICで３例以上のエビデンスのある変異を取り出す:

コマンド表記すると以下のようになります*。

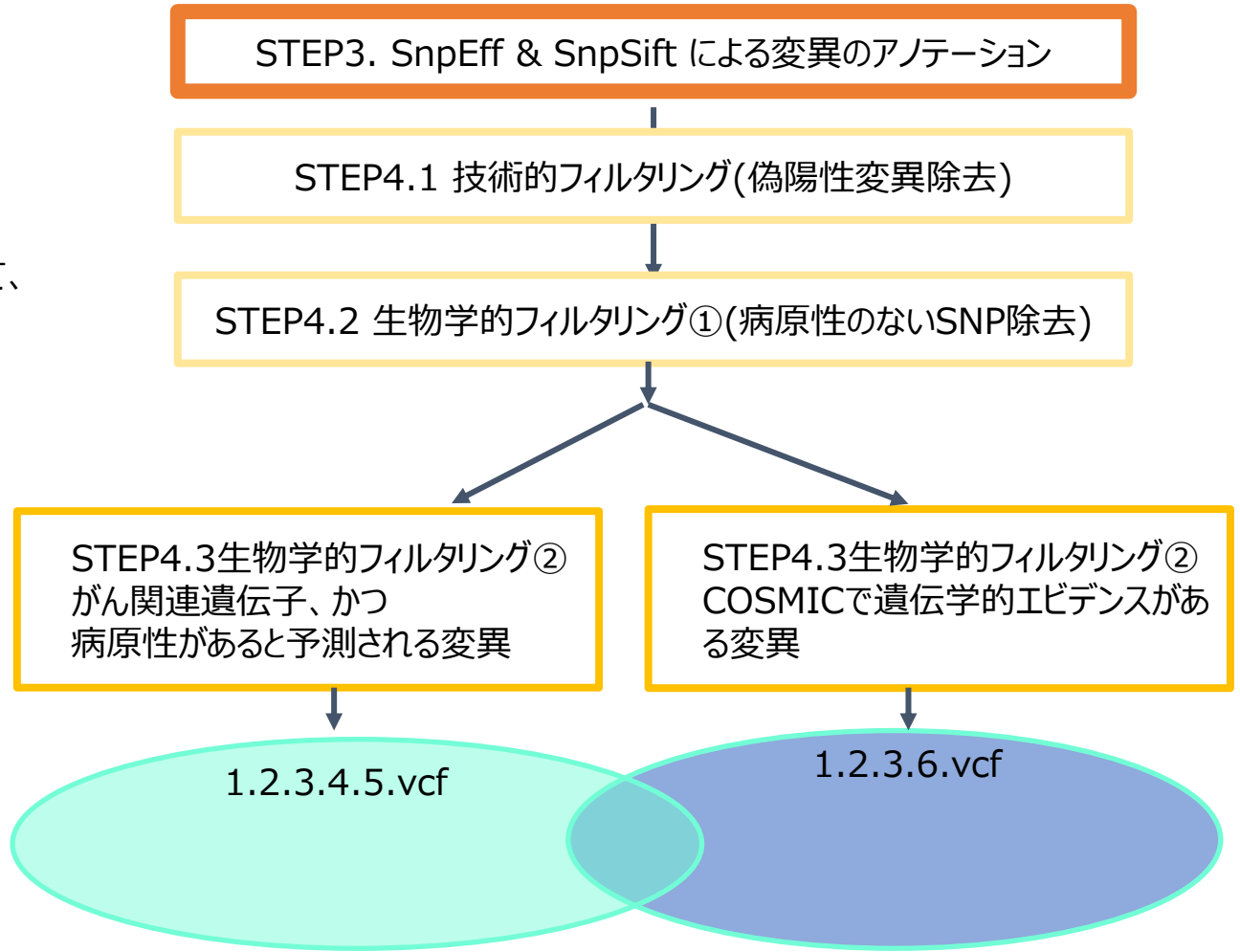
*実習では、README.txtに記載されたコマンドに従ってください。

```
#技術+生物学的フィルタリング①  後にCOSMICで３例以上のエビデンスのある変異を取り出す
cat ~/jinzai4/data1/test.anno.1.2.3.vcf | java -jar ~/jinzai4/bin/SnpSift.jar filter " ((
 CNT[0] >= 3 ) | (CNT[0] + CNT[1] >= 3 )| (CNT[0]+CNT[1]+CNT[2]>=3 ) )" >
 ~/jinzai4/data1/test.anno.1.2.3.6.vcf
```

5.フィルタリング操作のまとめ

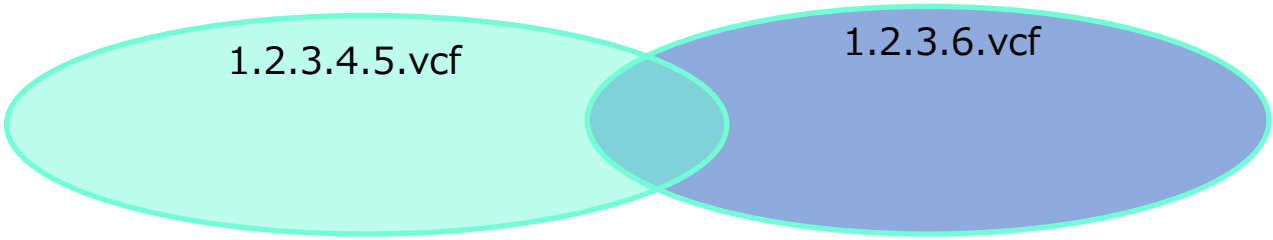
- ここまでの操作を復習しましょう(右図フロー参照)。
- まずSTEP3では変異のアノテーションを行いました。
- STEP4.1では、エラーなど偽陽性の可能性があるバリアントをfilter outして、エラーの可能性が低いバリアントを選択しました。
- STEP4.2では病原性のないSNPを除去しました(生物学的フィルタリング①)。
- STP4.3では病原性がある変異の絞り込みを行いました。(生物学的フィルタリング②)

  -がん関連遺伝子で病原性が予測される変異(1.2.3.4.5.vcf)
  - COSMICで遺伝学的エビデンスがある変異(1.2.3.6.vcf)



5. FilterされたVCFのマージ

STEP4.3生物学的フィルタリング②
がん関連遺伝子、かつ
病原性があると予測される変異



STEP4.3生物学的フィルタリング②
COSMICで遺伝学的エビデンスがある変異

STEP.5では出力された2つのvcfファイルを結合します(上記ベン図の和集合の変異が得られます)。

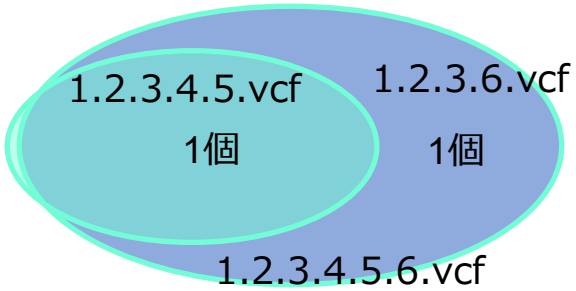
- 2つのvcfファイルを結合してみましょう。
  前のstepで出力された2つのvcfファイル（1.2.3.4.5.vcfと1.2.3.6.vcf）を結合します*。

```
# 1.2.3.4.5.vcf と 1.2.3.6.vcf を結合し、重複を取り除く
java -jar ~/jinzai4/bin/SnpSift.jar sort ~/jinzai4/data1/test.anno.1.2.3.4.5.vcf
 ~/jinzai4/data1/test.anno.1.2.3.6.vcf | sort -u > ~/jinzai4/data1/test.anno.1.2.3.4.5.6.vcf
```

*実習では、README.txtに記載されたコマンドに従ってください。

今回の実習では、元の２つのvcfファイルと、２つを結合して得られた1.2.3.4.5.6.vcf fileの関係は以下の様になっています。

がん関連遺伝子で、
病原性があると予測される変異



COSMICで、
遺伝学的エビデンスがある変異

5. 個数のカウント

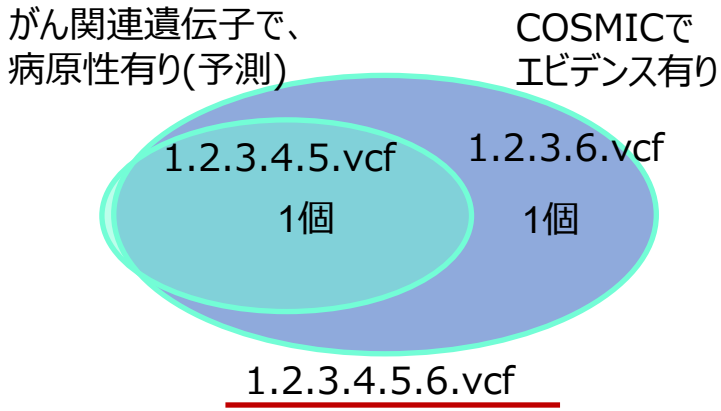
変異の個数を確認してみます*。

```
# 変異の個数をcount
files=`ls -v ~/jinzai4/data1/test.anno.vcf ~/jinzai4/data1/test.anno.*.vcf`
for file in $files
do
    count=`grep -v ^# $file | wc -l`
    echo $file $count
done
```

*実習では、README.txtに記載されたコマンドに従ってください。

カウント実行結果

```
#COLO
/home/eigosi/jinzai4/data1/test.anno.vcf 2402
"( GEN[1].DP >= 25 ) & (exists MBQ[*] ) & ( MBQ[*] >= 20 ) &( GEN[0].AF <= 0.04 ) & ( GEN[1].AF >= 0.08 )"
/home/eigosi/jinzai4/data1/test.anno.1.vcf 92
" (( exists AF) & (AF < 0.01) | ! (exists AF) )"
/home/eigosi/jinzai4/data1/test.anno.1.2.vcf 89
-n "(( CLNSIG has 'Likely_benign' ) | ( CLNSIG has 'benign')) "
/home/eigosi/jinzai4/data1/test.anno.1.2.3.vcf 89
"(ANN[*].IMPACT == 'HIGH' | ANN[*].IMPACT == 'MODERATE')"
/home/eigosi/jinzai4/data1/test.anno.1.2.3.4.vcf 1
"ANN[*].GENE in SET[0]"
/home/eigosi/jinzai4/data1/test.anno.1.2.3.4.5.vcf 1
 " (( CNT[0] >= 3 ) | (CNT[0] + CNT[1] >= 3 )| (CNT[0]+CNT[1]+CNT[2]>=3 ) )"
/home/eigosi/jinzai4/data1/test.anno.1.2.3.6.vcf 2
```



最初の変異call後2402個あった変異が最終的に生成されたvcfでは2個まで絞り込まれている事が分かります(赤線部)。

6. VCFからTSVへの変換

STEP.6ではVCF形式からTSV形式へ変換します。

- VCF形式からTSV形式へ変換してみましょう。
  SnpEff、SnpSift でアノテーションされた情報は、vcfファイルの INFO のカラムに追加されます。

  “chr1　183198　rs1379219406　C　T” の INFO カラムのデータ:

```
TYPE=SNV;called_by=strelka2;num_callers=1;CSQ=ENSG00000279928|unprocessed_pseudogene||non_coding_transcr
 ipt_exon_variant||||rs1379219406|||ENST00000624431.2:n.118C>T||MODIFIER|OR4F5|||||FO538757.2|Clone_bas
 ed_ensembl_gene;CancerGeneCensus=||||||||||;ANN=T|upstream_gene_variant|MODIFIER|LOC100996442|LOC100996
 442|transcript|XR_002958515.1|pseudogene||n.-
 2068G>A||||2068|,T|downstream_gene_variant|MODIFIER|WASH9P|...
```

このデータを TSV（Tab Separated Value）形式のデータに変換することで、Excel のカラムに情報を分割して、フィルタリングを行うことができるようになります。
事前に準備した awk のスクリプトを使用し、下記のコマンドで変換することができます。*

```
#awkのスクリプトを使用し、VCFファイルの INFO, FORMAT のデータをtab区切りのファイルに変換する。
awk -f ~/jinzai4/bin/info_format.awk ~/jinzai4/data1/test.anno.1.2.3.4.5.6.vcf > ~/jinzai4/data1/output.tsv
```

*実習では、README.txtに記載されたコマンドに従ってください。

6. 不要なアノテーション列の削除

STEP.6では不要なアノテーション列の削除も行います。

- 不要なアノテーション列を削除してみましょう。
  vcfファイルからフィルタリングに不要な列を除き、自分にとって必要な列*1のみ抜き出したvcfファイルを作成します。

```
#CHROM    POS       ID        REF       ALT       cosmic:GENE         cosmic:LEGACY_ID   cosmic:CDS
          cosmic:HGVSC        cosmic:HGVSP        cosmic:HGVSG        cosmic:CNT_total   tommo:AF
          cancercensus:CancerCensus    GRCh38:ANN          mutect:AS_SB_TABLE mutect:DP mutect:ECNT
          mutect:MBQ          mutect:MFRL         mutect:MMQ
```

コマンドは下記になります*2。

```
#cutで不要なアノテーション列の削除

cut -f 1,2,3,4,5,15,18,19,20,21,22,25,59,148,149,150,151,152,153,154,155 ~/jinzai4/data1/output.tsv >
~/jinzai4/data1/output2.tsv
```

*1 列名は仮のものであり、実習で用いる実際の列名とは異なります。

*2 実習では、README.txtに記載されたコマンドに従ってください。

7. ドライバー変異の最終絞り込み（1）

STEP.7ではドライバー変異の最終絞り込みを行います。

- ドライバー変異の最終絞り込みを実施してみましょう。

先ほどの出力されたtsvファイルから、ドライバー変異候補の遺伝子名*を、（遺伝子名の重複が起こらないように）取り出してみましょう。

```
 Driver genesドライバー変異候補の遺伝子名を、（遺伝子名の重複が起こらないように）取り出す
cut -f 149 ~/jinzai4/data1/output.tsv |  tail -n +2 |cut -d '|' -f 4 | sed 's/¥([^:]*¥):.¥+/¥1/g' | sort
-u |  perl -p -e 's/¥n/,/g' | sed 's/,$/¥n/g'
```

*BRAF*と*IQCJ*という遺伝子の変異が候補として残っている事がわかります。

```
$ >cut -f 149 ~/jinzai4/data1/output.tsv |  tail -n +2 |cut -d '|' -f 4 | sed 's/\([^:]*\):.\+/\1/g'
BRAF,IQCJ
```

*遺伝子名は仮のものです。実習で用いる実際のものとは異なる事があります。

7. ドライバー変異の最終絞り込み（2）

1. 最終的に得られたoutput2.tsv fileをEXCELで開いてみましょう

- 最終的に得られたoutput2.tsv fileをEXCELで開いてみましょう

#CHROM	POS	ID	REF	ALT	cosmic:GENE	cosmic:LEGA	cosmic:CDS	cosmic:HGVS	cosmic:HGVS	cosmic:HGVS	cosmic:CNT_	tommo:AF	cancercensu	GRCh38:ANN	mutect:AS_S	mutect:DP	mutect:ECNT	mutect:MBQ	mutect:MFRL	mutect:MMQ
chr3	159069997	COSV67333	A	G	IQCJ-SCHIP1	COSN33659,	c.9+556A>G	ENST00000476809.7:c.9+	3:g.1590699		5			G|intron_varian	47,56|33,20	165	1	37,37	623,589	60,60
chr7	140753336	13961;COSV	A	T	BRAF_ENST(	COSM476,C(	c.1799T>A,c	ENST000004	ENSP000004	7:g.1407533	118508			T|missense_va	44,52|72,81	257	1	37,37	582,578	60,60

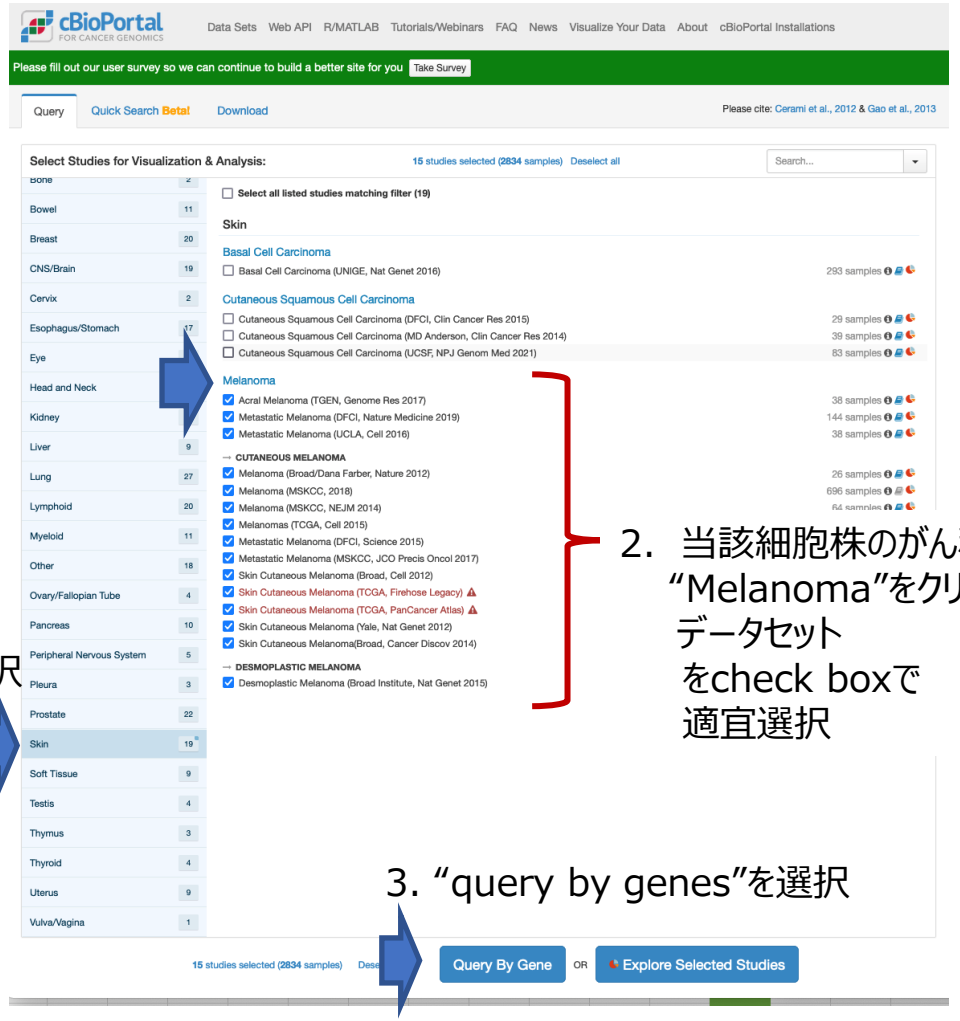
2個の変異が残っています。アノテーション”GRCh38:ANN”欄を参照してみます。

2. *BRAF*と*IQCJ*(赤線部)という遺伝子の変異が*driver*の候補です。

0
GRCh38:ANN
G|intron_variant|MODIFIER|IQCJ|IQCJ|transcript|NM_001042705.3|protein_coding|1/4|c.9+556A>G||||||,G|intron_variant|MODIFIER|IQCJ-SCHIP1|IQCJ-SCHIP1|transcript|NM_001197
T|missense_variant|MODERATE|BRAF|BRAF|transcript|NM_001374258.1|protein_coding|16/20|c.1919T>A|p.Val640Glu|2145/9807|1919/2424|640/807||,T|missense_variant|MODER

7. ドライバー変異の最終絞り込み（3）

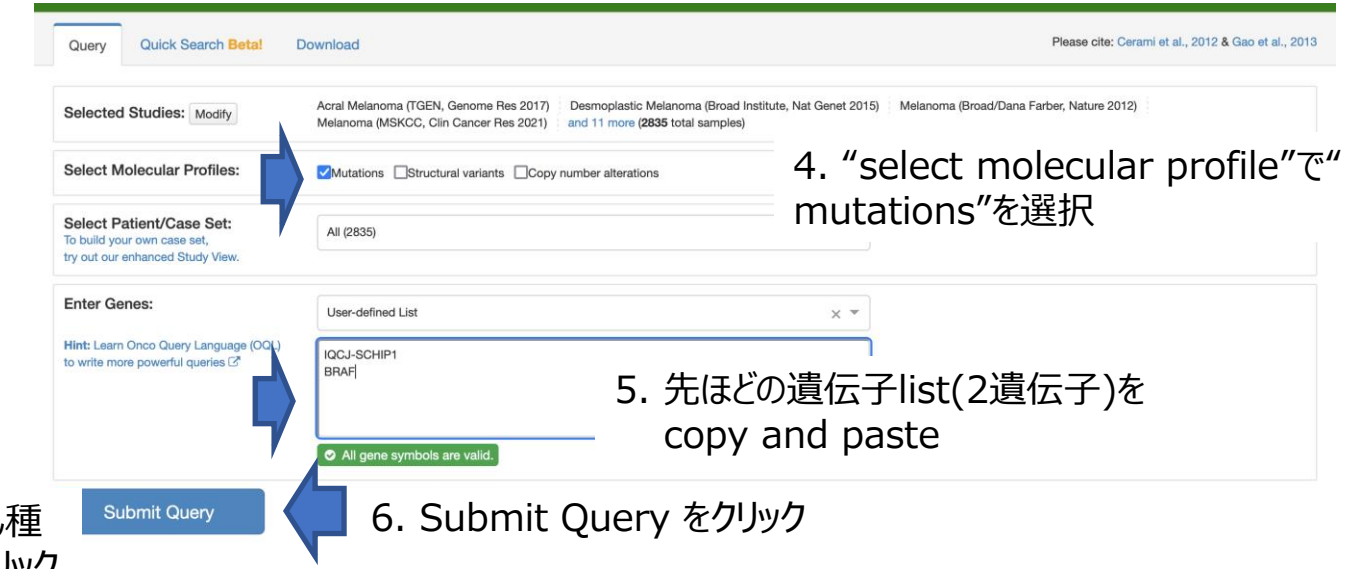
cBioPortal*に移動します（第 Ⅴ 章でも学習します）。



1. Skin を選択

2. 当該細胞株のがん種 “Melanoma”をクリック データセット をcheck boxで 適宜選択

3. “query by genes”を選択



4. “select molecular profile”で“ mutations”を選択

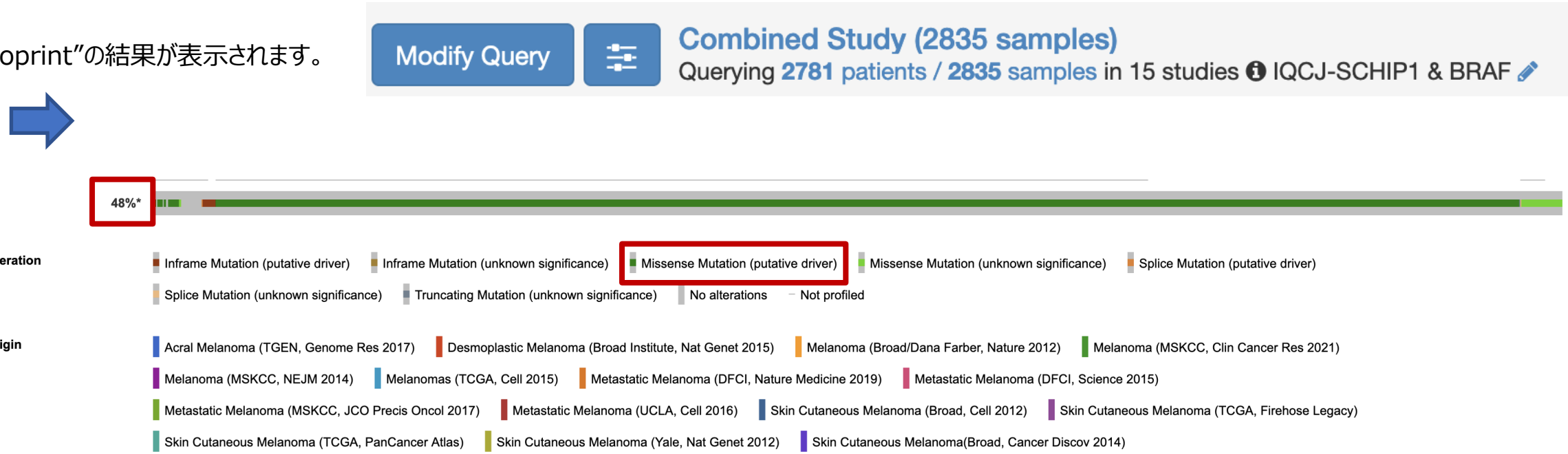
5. 先ほどの遺伝子list(2遺伝子)を copy and paste

6. Submit Query をクリック

* https://www.cbioportal.org/

7. ドライバー変異の最終絞り込み（4）

”oncoprint”の結果が表示されます。

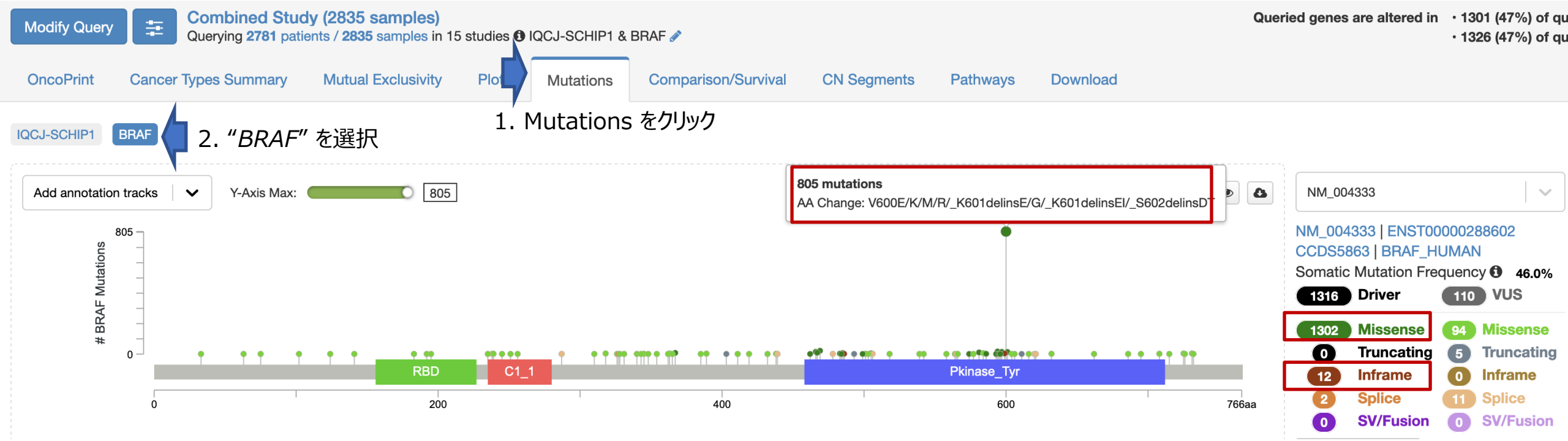


Melanoma 2835 検体で、高頻度に検出される”putative driver”として”*BRAF*”のmissense変異 がある事が分かります。

7. ドライバー変異の最終絞り込み（5）

次にmelanomaでの*BRAF*遺伝子変異のパターンを見てみましょう。

本例の*BRAF*遺伝子変異と同様に殆どがhotspotのミスセンスかin-frame変異であり、oncogene に矛盾しません。



8. 変異のIGVでの確認（1）

STEP.8ではIGVを使って直感的な操作で変異を確認します。

- IGV（Integrative Genomics Viewer）で変異を確認してみましょう。
  - ✓ Broad Instituteが開発した ゲノムブラウザ でGUIで直感的な操作が行えます。
  - ✓ bam、bed、vcfなどのファイル形式に対応

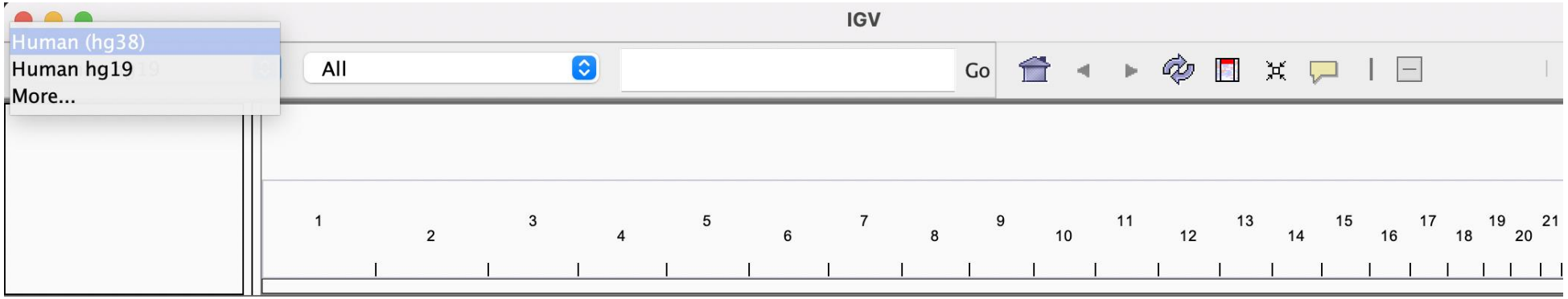
IGVを起動します。

```
igv.sh
```

3章で出力したマッピング結果のソート済みbamファイル（及び、インデックスを付与したbaiファイル）がある事を確認します。

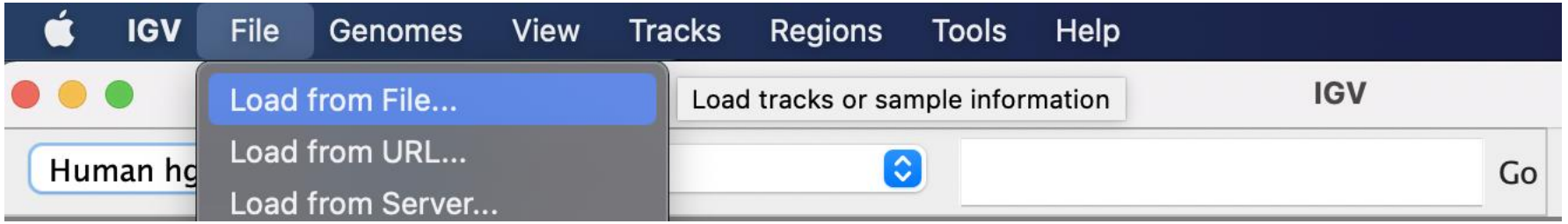
```
ls
```

左一番上のタブでhuman (hg38) リファレンス配列を指定します 。



8. 変異のIGVでの確認（2）

Fileタブから
「Load from File…」を選択します。**マッピングデータとしてbamファイルとフィルタリングしたvcfファイルを読み込みます。**
具体的には、Colo_tumor.bam（仮名） と Colo_Bl.bam と test_filtered1_2_vcf の3ファイルを順次読み込みます。



表示記号:

- ✓ IGVの表示は拡大・縮小ができます。
  - ➢ ＋で十分に拡大すれば、変異や挿入・欠失などの情報を確認することができます。
- ✓ リファレンスに対して異なる塩基配列をもつリードの該当塩基は、色分けされてハイライト表示されます。
- ✓ ズームアウトした場合は、縞模様のように表示されます
- ✓ ズームインすると、
  - ➢ **Insertions（挿入）**は「I」マークで表示されます。近くにカーソルを持っていけば、挿入された塩基情報が表示されます。
  - ➢ **Deletions（欠失）**は、「－」で表示されます。リファレンス配列を参照することで何が欠失しているかが分かります。

表示箇所の移動: 検索ボックスの利用

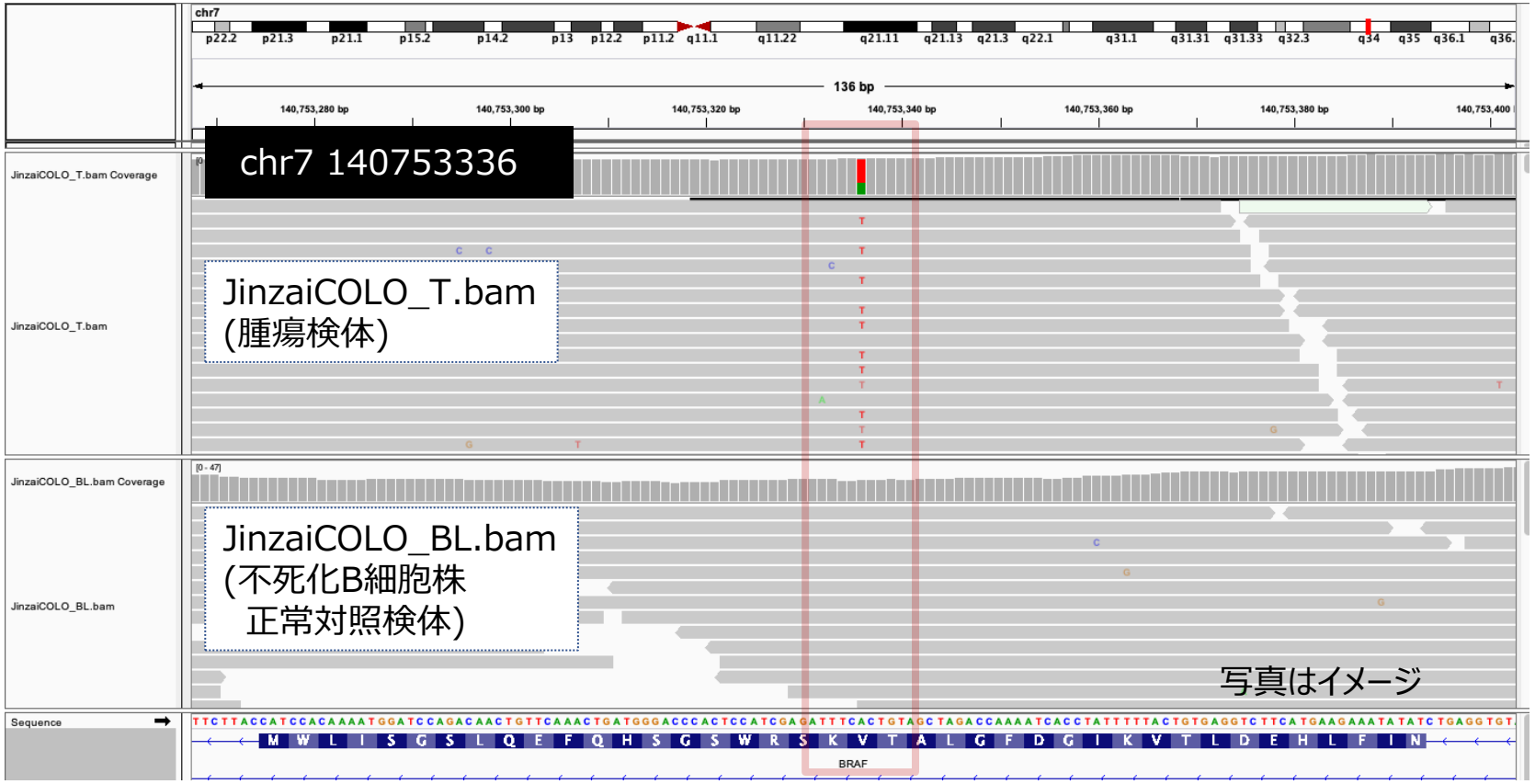
- ✓ メニューバーの下の1) 染色体の選択・2)検索ボックスでの移動が主な移動手段です。
- ✓ 特定の場所が表示されたら、マウスでドラッグして近くを自由に移動することが出来ます。
- ✓ 遺伝子名やゲノム座標を直接入力することで特定の場所へ移動することが出来ます。

8. 変異のIGVでの確認（3）

1. output2.tsv fileの*BRAF*遺伝子変異の染色体番号:染色体座標(赤枠)をコピーします

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
#CHROM	POS	ID	REF	ALT	cosmic:GENE	cosmic:LEGA	cosmic:CDS	cosmic:HGVS	cosmic:HGVS	cosmic:HGVS	cosmic:CNT_	tommo:AF	cancercensu	GRCh38:ANN	mutect:AS_S	mutect:DP	mutect:ECNT	mutect:MBQ	mutect:MFRL	mutect:MMQ
chr3	159069997	COSV673334	A	G	IQCJ-SCHIP1	COSN33659,	c.9+556A>G	ENST00000476809.7:c.9+	3:g.1590699		5			G|intron_vari	47,56|33,20	165	1	37,37	623,589	60,60
chr7	140753336	13961;COSV	A	T	BRAF_ENST(	COSM476,C(	c.1799T>A,c	ENST000004	ENSP000004	7:g.14075333	118508			T|missense_	44,52|72,81	257	1	37,37	582,578	60,60

2. IGVで腫瘍と正常のbam fileをloadした後、染色体番号:染色体座標をコピー＆ペーストで検索ボックスに入力して「Go」を押してみましょう



確かにドライバー変異として
BRAF p.V600E変異がtumorにあり、
controlにない事を確認できました(赤枠)。

事例2: AML(急性骨髄性白血病)

次に模擬AML症例(腫瘍量8割以上)のbam fileの変異コールからスタートして、
driver変異候補を検出する操作を行なってみましょう。

Annotation & Filtering workflow

模擬例2実習の概要:

1.環境設定*

2. Mutect2によるbam fileからの変異call(実習)

3. SnpEff & SnpSiftによるアノテーション (実習)

4. SnpSift を用いたVCFのフィルタリング(実習)

5-7.fileの整形と変異の確認(実習)

8. cBioPortalの検索(実習)

*今回の実習では既に実施済みのため
ユーザーが行う必要はない

4. SnpSift を用いたVCFのフィルタリング(実習)

ドライバー変異（候補）の絞り込み

4.1 技術的フィルタリング
目的:エラーの可能性が低い変異を選択
フィルタリング条件: 1+2
1.DP: Tumorでdepth ≧ 25
2.AF: TumorのVAF ≧ 0.05
NormalのVAF ≦ 0.02

4.2 生物学的フィルタリング①:
目的:病原性ではない変異の除外
filter条件:1または2の変異を除外
1.ToMMoSNPデータベースでMAFが0.01以上のSNP
2.ClinVarでbenign, likely benignと判断される変異

4.3 生物学的フィルタリング②:
目的: AMLにおいて病原性を有すると考えられる変異の選択
フィルタリング条件: 1+2
1.AML関連遺伝子:
- 造血器腫瘍ゲノム検査ガイドラインで ”Tier1(臨床的有用性が１)”
2.病原性有りと予測される変異:
-SnpEffのAnnotation Impactが “High”か”Moderate”

5. VCFからTSVへの整形と
不要なアノテーション列の削除（実習）

6. ドライバー変異の最終絞り込み（実習）

7. 変異のIGVでの確認（実習）

Mutect2による変異コール

今回は、bam ファイルからスタートして、下記のコマンドで Mutect2 を動かします。

```
#JAVA環境の設定　export JAVA_TOOL_OPTIONS='-XX:+UseSerialGC -Xmx10g -Xms2g’

~/jinzai3/bin/gatk Mutect2 ¥
        --callable-depth 10 ¥                                  #オプション(depth10未満の変異はcallしない)
        --min-base-quality-score 10 ¥                          #オプション(MBQ10以下の変異はcallしない)
        -R ~/jinzai3/data/ref/Homo_sapiens_assembly38.fasta ¥  #リファレンスファイルを指定
        -I ~/jinzai4/data/JinzaiAML_SW.bam ¥                   #normalのサンプル名を指定
        -I ~/jinzai4/data/JinzaiAML_BM.bam ¥                   #tumorのbamを指定
        --normal-sample JinzaiAML_SW ¥                         #でcontrolのbamを指定
        -O ~/jinzai4/data2/test.vcf ¥                          #output fileとしてtest.vcfを出力
        --java-options '-DGATK_STACKTRACE_ON_USER_EXCEPTION=true'   #JAVAのエラー出力(スタックトレース)表示を許可する
```

出力ファイルは、test.vcf になります。

変異数をカウントしてみます

```
$ >grep –v ^# ~/jinzai4/data2/test.anno.vcf | wc -l      # test.anno.vcf の行数を計算
269 ~/jinzai4/data2/test.anno.vcf                        #結果の行数
```

200個以上の変異がコールされています。

#コマンドの説明です

SnpSiftによるアノテーション

出力ファイルは、test.vcf になります。次にSnpSiftでannotationをtest.vcf fileに下記のコマンドでつけます

順番に、GRCh38、Tommo, clinvar, Cosmic, dbsnp, dbnsfpとアノテーションをつけます。

```
#JAVA環境の設定
export JAVA_TOOL_OPTIONS='-XX:+UseSerialGC -Xmx6g -Xms2g'

#Annotation
export vcf=~/jinzai4/data2/test.vcf
java -jar ~/jinzai4/bin/snpEff.jar GRCh38.p13.RefSeq $vcf > ${vcf%.*}.RefSeq.vcf #GRCh38のアノテーションをつける
java -jar ~/jinzai4/bin/SnpSift.jar annotate ~/jinzai4/bin/db/tommo-8.3kjpn-20200831-af_snvall-autosome.vcf.gz ${vcf%.*}.RefSeq.vcf >
${vcf%.*}.RefSeq.tommo.vcf #ToMMoのアノテーションをつける
java -jar ~/jinzai4/bin/SnpSift.jar annotate ~/jinzai4/bin/db/clinvar_20210814.vcf.gz  ${vcf%.*}.RefSeq.tommo.vcf >
${vcf%.*}.RefSeq.tommo.clinvar.vcf #ClinVarのアノテーションをつける
java -jar ~/jinzai4/bin/SnpSift.jar annotate ~/jinzai4/bin/db/CosmicCodingMuts.vcf.gz   ${vcf%.*}.RefSeq.tommo.clinvar.vcf >
${vcf%.*}.RefSeq.tommo.clinvar.cosmic.vcf   #COSMICのアノテーションをつける
java -jar ~/jinzai4/bin/SnpSift.jar annotate ~/jinzai4/bin/db/DBexome20170802.vcf.gz  ${vcf%.*}.RefSeq.tommo.clinvar.cosmic.vcf >
${vcf%.*}.RefSeq.tommo.clinvar.cosmic.dbsnp.vcf   #dbSNPのアノテーションをつける
java -jar ~/jinzai4/bin/SnpSift.jar annotate ~/jinzai4/bin/db/00-All.vcf.gz  ${vcf%.*}.RefSeq.tommo.clinvar.cosmic.dbsnp.vcf >
${vcf%.*}.RefSeq.tommo.clinvar.cosmic.dbsnp.dbnsfp.vcf #dbNSFPのアノテーションをつける
mv ${vcf%.*}.RefSeq.tommo.clinvar.cosmic.dbsnp.dbnsfp.vcf ${vcf%.*}.anno.vcf # vcf fileの名称をtest.anno.vcfに変更
```

今回使用するFiltering条件

今回使用するフィルタリング条件の例は以下の通りです

技術的フィルタリング（Technical Filtering）

フィルタリング条件：

- tumorでのカバレッジ（DP） 25以上
- Control “GEN[0]”のVAF が0.02以下で、Tumor “GEN[1]”のVAFが0.05以上

生物学的フィルタリング（Biological Filtering）

①病原性ではないSNPを除外するフィルタリング条件：

- 1.ToMMoでMAF1%以上の変異をFilter out(=MAF1%未満かMAFの記載がない変異をpick up)
- 2.ClinVarで clinical significance が”benign”または”likely benign”とアノテーションされた病原性ではないSNPをFilter out

②ゲノムガイドラインにある病原性有りと予測される変異を取り出す生物学的フィルタリング条件：

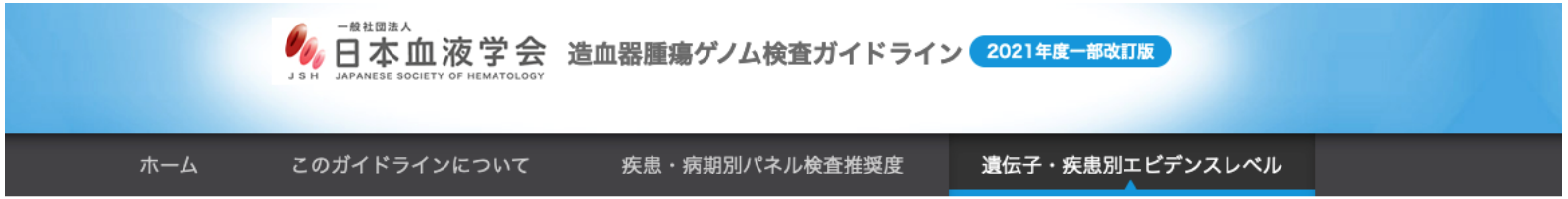
- 3.AML関連遺伝子*である
- 4. SnpEffのAnnotation Impact がHIGHかMODERATEである

* 次頁のスライドで解説します

生物学的フィルタリング②準備(1)

- AML関連遺伝子のフィルタリング準備を行います

造血器腫瘍ゲノム検査ガイドライン[1]の下記ウェブサイトにアクセスしてください。
http://www.jshem.or.jp/modules/genomgl/



AML関連遺伝子として、
AMLで「診断」、「治療法選択」、「予後予測」のいずれかで、エビデンスレベルが高く重要な遺伝子を検索します。

1. “項目別検索”のタブをクリックします
2. 臨床的有用性[2]を1(グレード1: 最も高い)を選択します
3. 該当する造血器疾患に”AML”を入力します
4. ”検索する“をクリックして検索実行します

1: 日本血液学会が公開している、遺伝子異常の臨床的有用性を、学会指針、文献等をもとに専門のワーキンググループで評価し、造血器腫瘍、及びその類縁疾患に対する「がん遺伝子パネル検査」の基盤となる遺伝子群を提示しているガイドライン

2:「診断」、「治療法選択」、「予後予測」のいずれかのカテゴリーで、米国「がんゲノム診断における3学会合同指針(J Mol Diagn. 2017;19(1):4-23.)に基づき、エビデンスレベルがA若しくはBと評価された異常の事

生物学的フィルタリング②準備(2)

検索結果:
AML関連遺伝子(AMLにおいて、臨床的有用性が グレード1の遺伝子とその異常)の検索結果が表示されます。

76 件のデータが検索されました。
2 / 2ページ中　※項目名をクリックすると降順または昇順に並び替わります。

遺伝子	別名	該当する造血器疾患	臨床的有用性	遺伝子変異の機能的意義とその種類	関連する染色体構造変化	Fast-track対象遺伝子異常
NUP214		AML with t(6;9)(p23;q34.1); DEK-NUP214	1	機能獲得（融合：DEK/NUP214）	t(6;9)(p22;q34)	
NUP98		AML	1	機能獲得（融合：NUP98/NSD1, NUP98/KDM5Aなど）	t(5;11)(q35;p15.5), t(11;12)(p15;p13) など	
PDGFRB		AML with MRC	1	機能獲得（融合：NDE1/PDGFRB）	t(5;17)(q32;p13.2)	
PHF6		AML				
PICALM	CALM	AML		合：PICALM/MLLT10)		
PPM1D		AML	1	のnonsenseもしくはframeshift変異)		
PRDM16		AML with MRC	1	機能獲得（再構成：RPN1/PRDM16）	t(1;3)(p36.3;q21.2)	
RAD21		AML	1	機能喪失		
RBM15		AML (megakaryoblastic) with t(1;22)(p13.3;q13.3); RBM15-MKL1	1	機能獲得（融合：RBM15/MKL1）	t(1;22)(p13;q13)	
RPN1		AML with MRC	1	機能獲得（再構成：RPN1/PRDM16）	t(1;3)(p36.3;q21.2)	
RUNX1	AML1	AML with t(8;21)(q22;q22.1); RUNX1-RUNX1T1	1	機能獲得（融合：RUNX1/RUNX1T1）	t(8;21)(q22;q22)	

1. 赤枠部の遺伝子をcopyします

2. JSH_AML.txtとして遺伝子名をAML関連遺伝子としてtxt形式で保存します
(実習では作成済み)

JSH_AML.txt

```
1  ABL1
2  ASXL1
3  BCOR
4  BCR
5  CBFA2T3
6
```

SnpSiftによるフィルタリング

Annotationの出力ファイル、test.anno.vcf に対して、技術フィルタリングと生物学的フィルタリングを行います

```
#JAVA環境の設定
#
export JAVA_TOOL_OPTIONS='-XX:+UseSerialGC -Xmx13g -Xms2g'
set -xv
#4.1 技術的フィルタリング
#
cat ~/jinzai4/data2/test.anno.vcf | java -jar ~/jinzai4/bin/SnpSift.jar filter "( GEN[0].DP >= 25 ) & ( GEN[1].AF <= 0.02 ) & ( GEN[0].AF >= 0.05 )"  >
~/jinzai4/data2/test.anno.1.vcf        #depthとアレル頻度で技術的filterをかける

##########################################################################
#4.2 生物学的フィルタリング
#
cat ~/jinzai4/data2/test.anno.1.vcf | java -jar ~/jinzai4/bin/SnpSift.jar filter " (( exists AF) & (AF < 0.01) | ! (exists AF) )"  >
~/jinzai4/data2/test.anno.1.2.vcf     # ToMMoでMAF1%以上の変異を除外

cat ~/jinzai4/data2/test.anno.1.2.vcf | java -jar ~/jinzai4/bin/SnpSift.jar filter  -n " (( CLNSIG has 'Likely_benign' ) | ( CLNSIG has 'benign')) " >
~/jinzai4/data2/test.anno.1.2.3.vcf  # ClinVarで病原性ではない変異を除外

cat ~/jinzai4/data2/test.anno.1.2.3.vcf | java -jar ~/jinzai4/bin/SnpSift.jar filter " (ANN[*].IMPACT == 'HIGH' | ANN[*].IMPACT == 'MODERATE') " >
~/jinzai4/data2/test.anno.1.2.3.4.vcf                #病原性有りと予測される変異を取り出す

cat ~/jinzai4/data2/test.anno.1.2.3.4.vcf | java -jar ~/jinzai4/bin/SnpSift.jar filter -s ~/jinzai4/bin/db/JSH_AML.txt "ANN[*].GENE in SET[0] ">
~/jinzai4/data2/test.anno.1.2.3.4.5.vcf #Set option でAML関連遺伝子において、病原性有りと予測される変異を取り出す
```

変異個数のカウント

変異の個数を確認してみます*。

```
# 変異の個数をcount
#
files=`ls -v ~/jinzai4/data2/test.anno.vcf ~/jinzai4/data2/test.anno.*.vcf`
for file in $files
do
    count=`grep -v ^# $file | wc -l`
    echo $file $count
done
```

実行結果

```
/home/eigosi/jinzai4/data2/test.anno.vcf 269
"( GEN[0].DP >= 25 ) & ( GEN[1].AF <= 0.02 ) & ( GEN[0].AF >= 0.05 )"
/home/eigosi/jinzai4/data2/test.anno.1.vcf 35
" (( exists AF) & (AF < 0.01) | ! (exists AF) )"
/home/eigosi/jinzai4/data2/test.anno.1.2.vcf 26
-n "(( CLNSIG has 'Likely_benign' ) | ( CLNSIG has 'benign')) "
/home/eigosi/jinzai4/data2/test.anno.1.2.3.vcf 14
"(ANN[*].IMPACT == 'HIGH' | ANN[*].IMPACT == 'MODERATE')"
/home/eigosi/jinzai4/data2/test.anno.1.2.3.4.vcf 14
"ANN[*].GENE in SET[0]"
/home/eigosi/jinzai4/data2/test.anno.1.2.3.4.5.vcf 4
```

269個あった変異が最終的に4個まで絞り込まれている事が分かります。*実習では、README.txtに記載されたコマンドに従ってください。

VCFからTSVへの変換と不要なアノテーション列の削除

VCF形式からTSV形式へ変換します。

```
#awkのスクリプトを使用し、VCFファイルの INFO, FORMAT のデータをtab区切りのファイルに変換する。
set -xv
awk -f ~/jinzai4/bin/info_format.awk ~/jinzai4/data2/test.anno.1.2.3.4.5.vcf > ~/jinzai4/data2/output.tsv
```

不要なアノテーション列の削除を行います

```
# Extract columns
cut -f 1,2,3,4,5,15,18,19,20,21,22,25,59,148,149,150,151,152,153,154,155 ~/jinzai4/data2/output.tsv > ~/jinzai4/data2/output2.tsv
```

出力されたtsvファイルから、ドライバー変異候補の遺伝子名*を、（遺伝子名の重複が起こらないように）取り出します

```
# ドライバー変異候補の遺伝子名を、（遺伝子名の重複が起こらないように）取り出す
set +xv
cut -f 149 ~/jinzai4/data2/output.tsv | tail -n +2 | cut -d '|' -f 4 | sed 's/¥([^:]*¥):.¥+/¥1/g' | sort -u | perl -p -e 's/¥n/,/g' | sed 's/,$/¥n/g'
```

*実習では、README.txtに記載されたコマンドに従ってください。

```
$ >cut -f 149 ~/jinzai4/data2/output.tsv | tail -n +2 | cut -d '|' -f 4 | sed 's/\([^:]*\):.\+/\1/g' | sort -u | perl -p -e 's/\n/,/g' | sed 's/,$/\n/g'
FLT3,NPM1
```

ドライバー変異候補の遺伝子名として*NPM1*と*FLT3*の２遺伝子が絞り込まれています。

ドライバー変異の最終絞り込み

1. 最終的に得られたoutput2.tsv fileをEXCELで開いてみましょう。

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
#CHROM	POS	ID	REF	ALT	cosmic:GENE	cosmic:LEGA	cosmic:CDS	cosmic:HGVS	cosmic:HGVS	cosmic:HGVS	cosmic:CNT_	tommo:AF	cancercens	GRCh38:ANN	utect:AS_S	mutect:DP	mutect:ECNT	mutect:MBQ	mutect:MFRL
chr13	28028203	376108;COS	G	C	FLT3	COSM58793	c.2028C>G	ENST000002	ENSP000002	13:g.280282(	7			C|missense_	07,77|25,11	228	1	32,36	170,201
chr5	171410548	.	T	TCCCTA										TCCCTA|frar	91|0,22	121	4	36,36	212,219
chr5	171410549	.	G	GC										GC|frameshif	91|0,22	117	4	36,36	212,219
chr5	171410550	.	G	C										C|missense_	91|0,22	115	4	36,36	212,219

4 個の変異が残っています。アノテーション”GRCh38:ANN”欄(赤枠)を参照してみます。

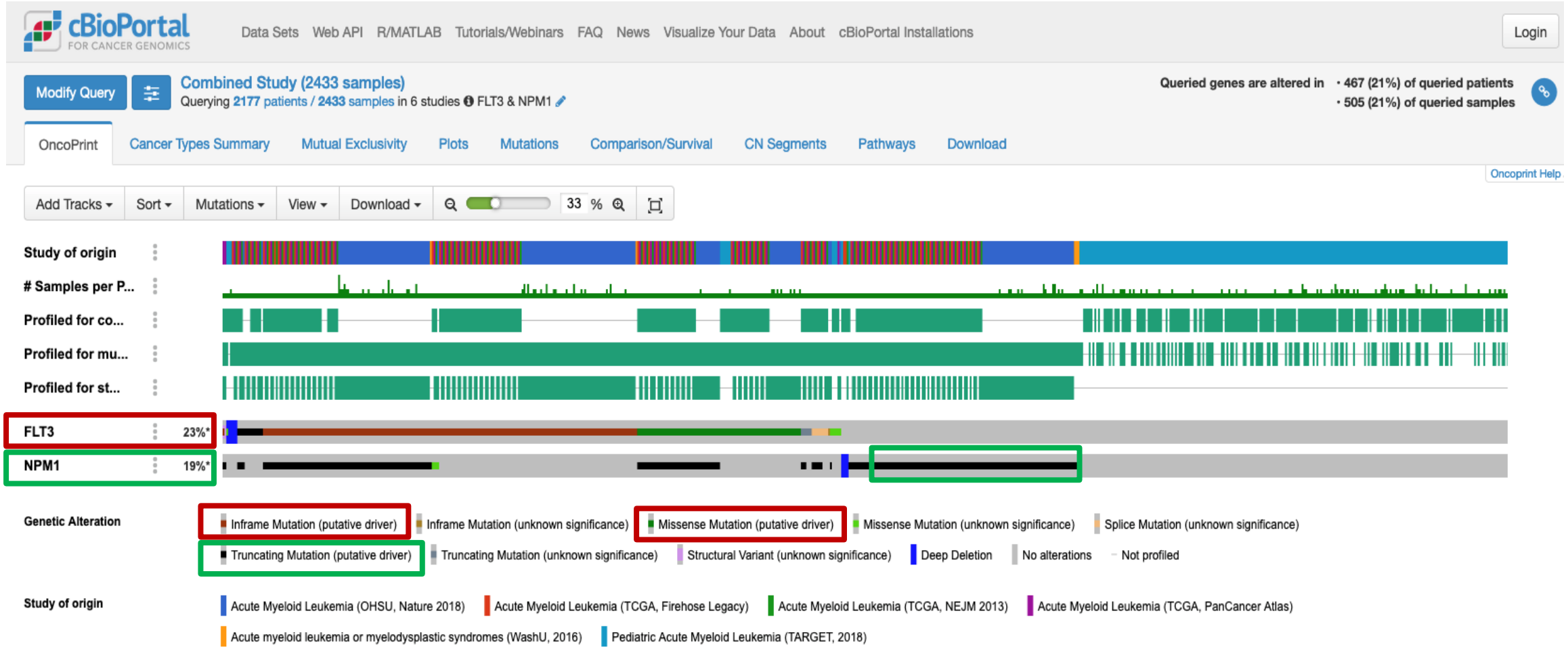
2. *FLT3*と*NPM1*(赤線部)という遺伝子の変異がdriverの候補である事がわかります。

O
GRCh38:ANN
C|missense_variant|MODERATE|FLT3|FLT3|transcript|NM_004119.3|protein_coding|16/24|c.2028C>G|p.Asn676Lys|2094/3826|2028/2982|676/993||,C|missense_
TCCCTA|frameshift_variant|HIGH|NPM1|NPM1|transcript|NM_001355006.2|protein_coding|12/12|c.868_869insCCCTA|p.Trp290fs|1004/1355|869/885|290/294||,T
GC|frameshift_variant|HIGH|NPM1|NPM1|transcript|NM_001355006.2|protein_coding|12/12|c.869_870insC|p.Trp290fs|1005/1355|870/885|290/294||,GC|frameshi
C|missense_variant|MODERATE|NPM1|NPM1|transcript|NM_001355006.2|protein_coding|12/12|c.870G>C|p.Trp290Cys|1005/1355|870/885|290/294||,C|missense

3. cBIoPortalで前回と同様にして、この二つの遺伝子をAML例で検索してみます。

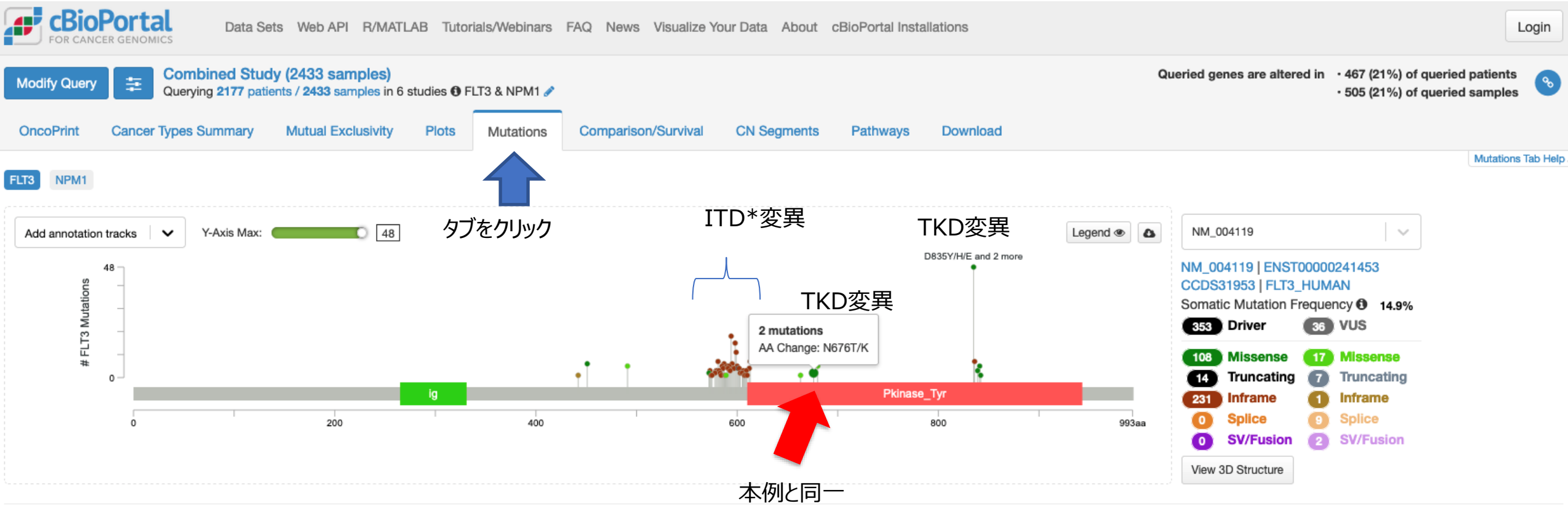
cBioPortalのAML例で*FLT3*と*NPM1*を検索した結果

cBIoPortalで前回と同様にして、先ほど絞り込みで得られたの二つの遺伝子をAML例で検索してみます。



AML 2433 検体で、高頻度に検出される”putative driver”として”*FLT3*”のinframe変異とmissense変異(赤枠)
“*NPM1*”のtruncating変異(緑枠)がある事が分かります。

cBioPortalでの*FLT3*変異（AML症例）



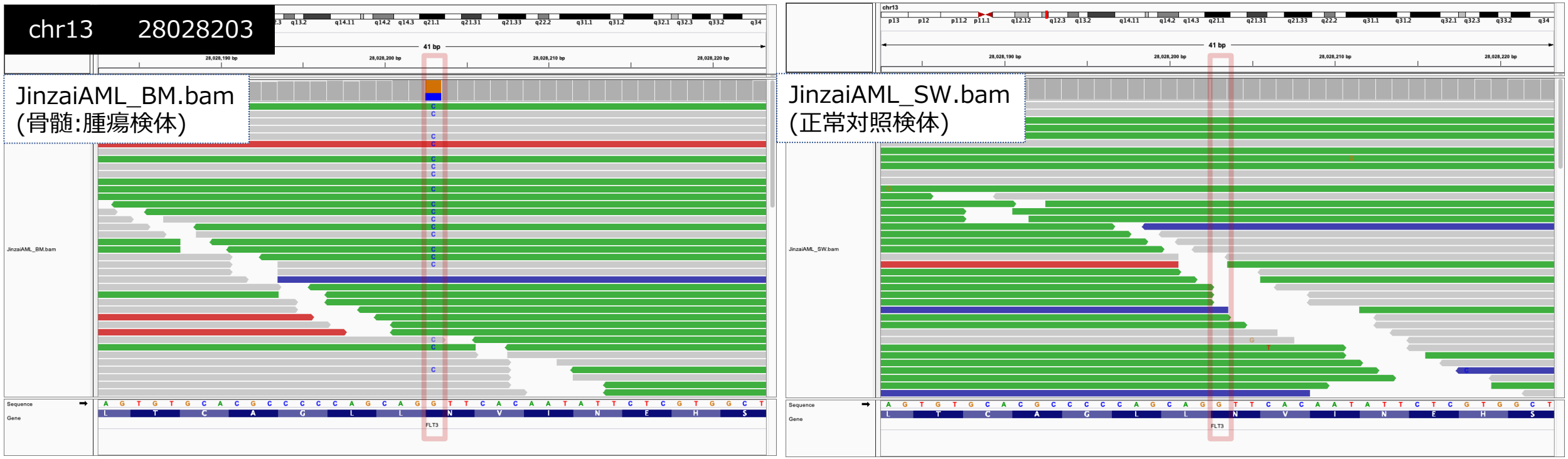
*遺伝子内縦列重複（*internal tandem duplication*, *ITD*）

変異のIGVでの確認（1）

1. output2.tsv file の*FLT3*遺伝子変異の染色体番号:染色体座標(赤枠)をコピーします

#CHROM	POS	ID	REF	ALT	cosmic:GENE	cosmic:LEGA	cosmic:CDS	cosmic:HGVS	cosmic:HGVS	cosmic:HGVS	cosmic:CNT_	tommo:AF	cancercensu	GRCh38:ANN	mutect:AS_S	mutect:DP	mutect:ECNT	mutect:MBQ	mutect:MFRL
chr13	28028203	376108;COS	G	C	FLT3	COSM58793	c.2028C>G	ENST000002	ENSP000002	13:g.280282(	7			C|missense_	107,77|25,11	228	1	32,36	170,201
chr5	171410548	.	T	TCCCTA										TCCCTA|frar	0,91|0,22	121	4	36,36	212,219
chr5	171410549	.	G	GC										GC|frameshi	0,91|0,22	117	4	36,36	212,219
chr5	171410550	.	G	C										C|missense_	0,91|0,22	115	4	36,36	212,219

2. IGVで腫瘍と正常のbam fileをloadした後、染色体番号:染色体座標をコピー＆ペーストで検索ボックスに入力し、「Go」を押してみましょう



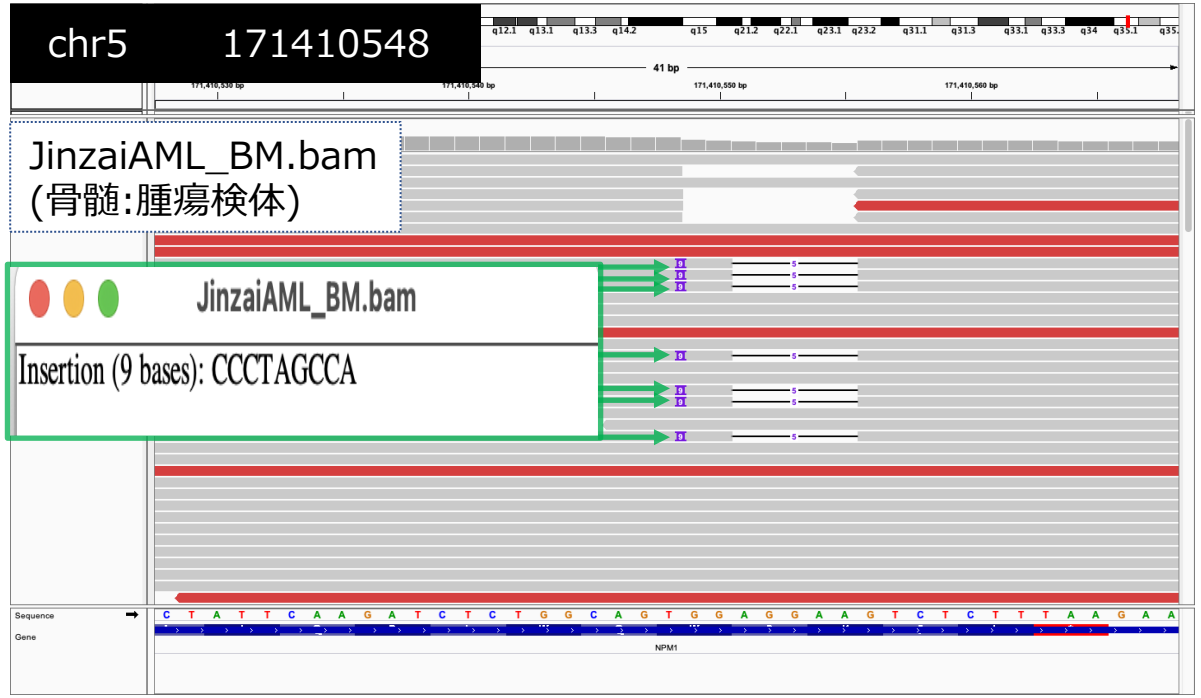
確かにドライバー変異として*FLT3* 遺伝子のTKD変異がtumorにあり、controlにはほぼない事を確認できました(赤枠)。

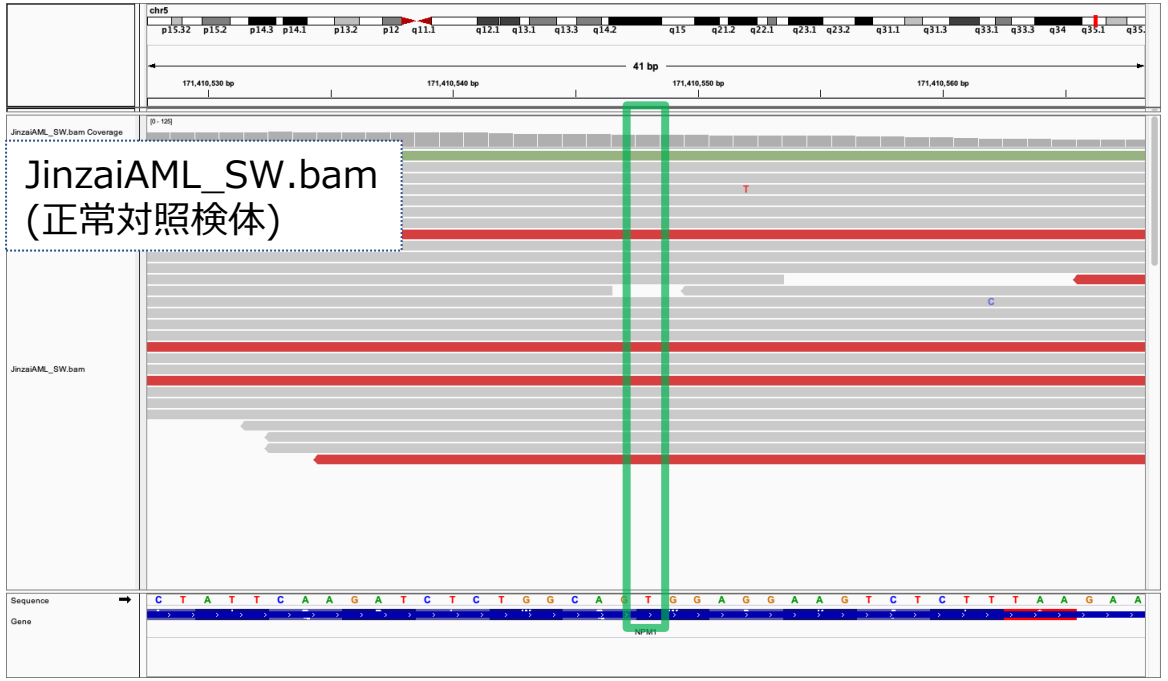
変異のIGVでの確認（2）

1. output2.tsv file の*NPM1*遺伝子変異の染色体番号:染色体座標(赤枠)をコピーします

#CHROM	POS	ID	REF	ALT	cosmic:GENE	cosmic:LEGA	cosmic:CDS	cosmic:HGVS	cosmic:HGVS	cosmic:HGVS	cosmic:CNT_	tommo:AF	cancercensu	GRCh38:ANN	mutect:AS_S	mutect:DP	mutect:ECNT	mutect:MBQ	mutect:MFRL
chr13	28028203	376108;COS	G	C	FLT3	COSM58793	c.2028C>G	ENST000002	ENSP000002	13:g.280282(	7			C|missense_	107,77|25,11	228	1	32,36	170,201
chr5	171410548	.	T	TCCCTA										TCCCTA|frar	0,91|0,22	121	4	36,36	212,219
chr5	171410549	.	G	GC										GC|frameshi	0,91|0,22	117	4	36,36	212,219
chr5	171410550	.	G	C										C|missense_	0,91|0,22	115	4	36,36	212,219

2. IGVで腫瘍と正常のbam fileをloadした後、染色体番号:染色体座標をコピー＆ペーストで検索ボックスに入力し、「Go」を押してみましょう





確かにドライバー変異としてinsertion(緑矢印)の結果生じる、*NPM1*のtruncating変異がtumorにあり、controlにはない事を確認できました(緑枠)。

造血器腫瘍ゲノム検査ガイドライン検索結果

http://www.jshem.or.jp/modules/genomgl/

検索

1. *FLT3*を検索

フリーワード検索（全項目） FLT3 検索する

▼ 項目別検索（複数の項目で絞り込みが可能です）

遺伝子 A-Z Index

A	B	C	D	E
F	G	H	I	J
K	L	M	N	O
P	R	S	T	U
V	W	X	Z	

11 件のデータが検索されました。

1 / 1ページ中　※項目名をクリックすると降順または昇順に並び替わります。

遺伝子	別名	該当する造血器疾患	臨床的有用性	遺伝子変異の機能的意義とその種類	関連する染色体構造変化	Fast-track 対象遺伝子異常
FLT3		PMF	1	機能獲得（変異：ITD）		D835、N676K
FLT3		MPAL, T/myeloid, NOS	2	機能獲得（変異）		D835、N676K
FLT3		MDS	1	機能獲得（変異：ITD）		D835、N676K
FLT3		JMML	2	機能獲得（変異）		D835、N676K
FLT3		ETP-ALL	1	機能獲得（変異）		D835、N676K
FLT3		B-ALL/LBLL BCR-ABL1-like	1	機能獲得（変異：ITD, TKD）		D835、N676K
FLT3		B-ALL/LBLL	2	機能獲得（変異）		D835、N676K
FLT3		B-ALL/LBLL	2	機能獲得（変異）		D835、N676K
FLT3		AUL	1	機能獲得（変異：ITD, TKD）		D835、N676K
FLT3		AML	1	機能獲得（変異：F691Lなど）		
FLT3		AML	1	機能獲得（変異：ITD, TKD）		D835、N676K

1 / 1ページ中

2. AMLを選択してクリック

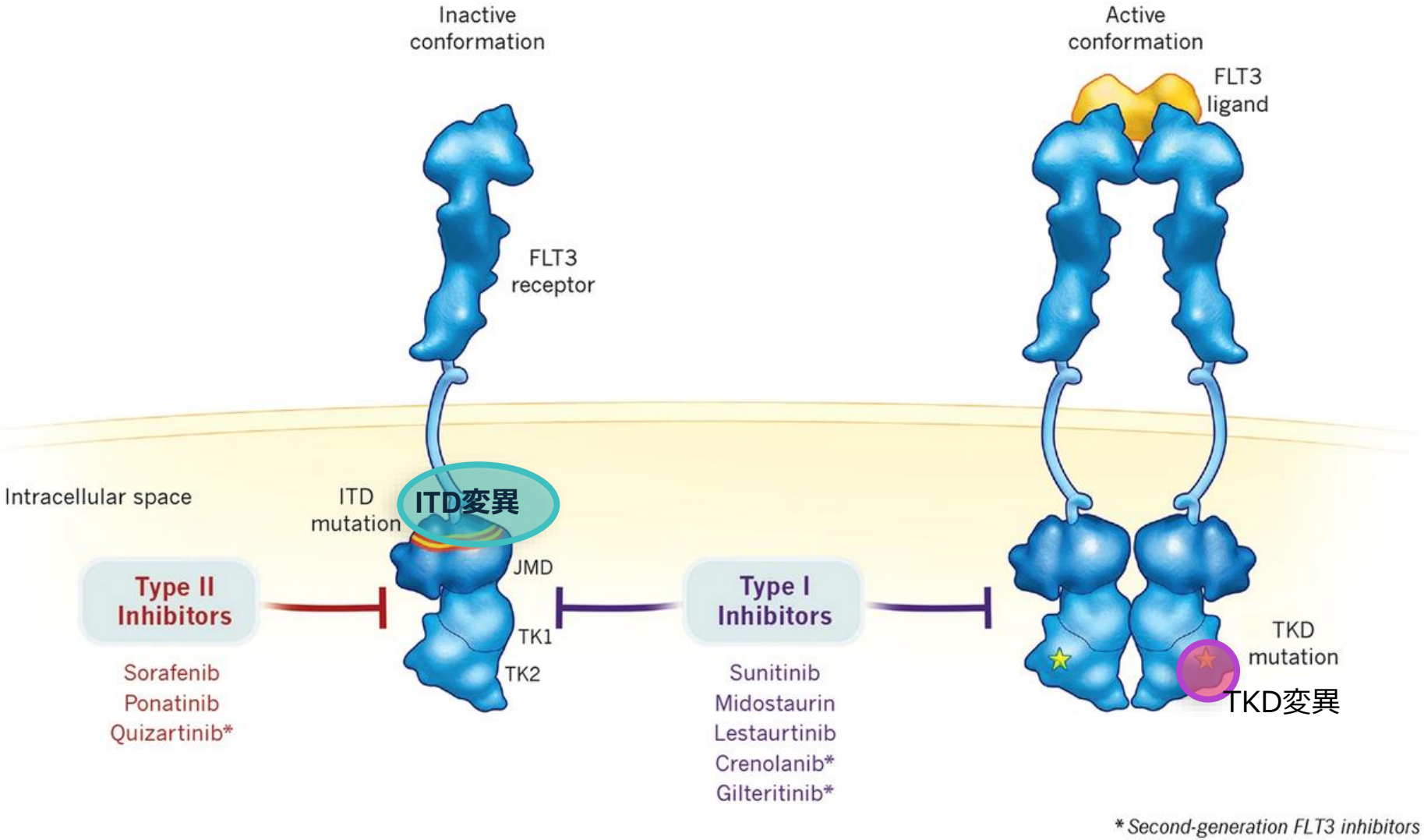
詳細情報

遺伝子名 FLT3

該当する造血器疾患	AML
臨床的有用性	1
遺伝子変異の機能的意義とその種類	機能獲得（変異：ITD, TKD）
関連する染色体構造変化	
Fast-track 対象遺伝子異常	D835、N676K
エビデンスレベル 診断	A　根拠となる論文、学会指針、臨床試験：WHO2017, ELN2017, NCCN2020
エビデンスレベル 治療法選択	A　根拠となる論文、学会指針、臨床試験：PMID: 28644114, 28645776, 29859851, NCCN2020
エビデンスレベル 予後予測	A　根拠となる論文、学会指針、臨床試験：NCCN2020, ELN2017
薬剤	遺伝子異常と関連して、当該疾患に対して薬事承認された薬剤：Gilteritinib, Quizartinib (ITDのみ) 遺伝子異常と関連して、当該疾患に対してFDA承認された薬剤：Midostaurin, Gilteritinib 遺伝子異常と関連して、他の腫瘍に対してFDA承認された薬剤：Midostaurin, Sorafenib, Gilteritinib
その他コメント	FLT3-ITDとFLT3-TKDでは予後因子としての意味合いが異なる可能性がある

FLT3阻害剤

Type I FLT3阻害剤は*ITD*と*TKD*変異の両方に有効
一方、Type II FLT3阻害剤は*ITD*変異には有効だが、*TKD*変異には効果がない



Leukemia 2019;33(2):299-312

FLT3 p.N676K変異に対する Type Ⅰ FLT3阻害剤の効果についての報告

TypeIのFLT3阻害剤のギルテリチニブが奏功する可能性

REGULAR ARTICLE

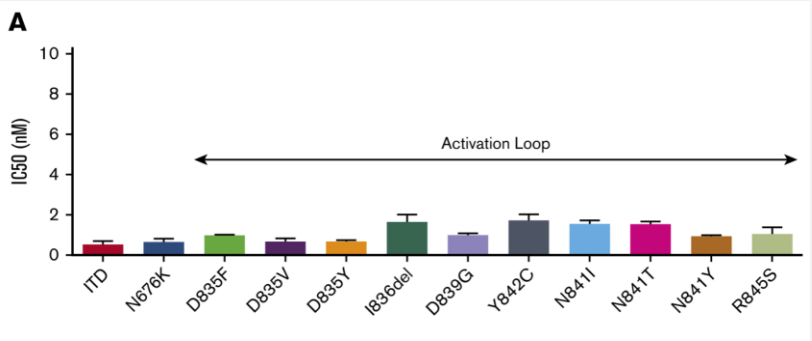
blood advances

Check for updates

Gilteritinib is a clinically active FLT3 inhibitor with broad activity against FLT3 kinase domain mutations

Theodore C. Tarver,[1] Jason E. Hill,[2] Leena Rahmat,[3] Alexander E. Perl,[4] Erkut Bahceci,[2] Kenichi Mori,[5] and Catherine C. Smith[1]

[1]Division of Hematology-Oncology, Department of Medicine, University of California, San Francisco, San Francisco, CA; [2]Astellas Pharma US, Inc., Northbrook, IL; [3]Division of Oncology–Hematologic Malignancies, Department of Medicine, Johns Hopkins University, Baltimore, MD; [4]Division of Hematology-Oncology, Department of Medicine, University of Pennsylvania, Philadelphia, PA; and [5]Astellas Pharma, Inc., Tokyo, Japan



In vitroの検証で*FLT3* p.N676K変異陽性細胞株の増殖をギルテリチニブが抑制することを示した

Blood Adv (2020) 4 (3): 514–524.

症例報告

Gilteritinib が奏効した *FLT3*-N676K 変異を有する再発急性骨髄性白血病

宮島 徹[1], 原田晋平[2], 小笠原励起[1], 横山絵美[1], 泉山 康[1], 盛 暁生[1], 齋藤 誠[1], 森岡正信[1], 池 成基[3], 南 陽介[3], 近藤 健[1]

症例は 68 歳女性。2019 年 5 月に急性骨髄性白血病（AML-M5b，G-band 正常核型，*FLT3*-ITD 変異陰性）と診断された。寛解導入療法，地固め療法を施行し寛解を維持していたが，2020 年 1 月に口唇，歯肉の腫脹および末梢血中の芽球の増加を認めた。CAG 療法（cytarabine, aclarubicin, G-CSF）による救援療法を施行したが非寛解であった。包括的遺伝子解析を行い，現行のコンパニオン診断薬では検出できない *FLT3*-N676K 変異が判明した。Gilteritinib 単剤療法を開始し，口唇，歯肉の腫脹は著明に改善し，28 日目に CRi が得られた。その後 106 日目に再発を認め gilteritinib を中止し，病勢コントロール不能となり死亡した。再発時の遺伝子解析で *NRAS* 遺伝子変異が新たに検出された。包括的遺伝子解析が治療方針の決定に有用であると考えられた 1 例であった。（臨床血液 **63**（1）：51～54，2022）

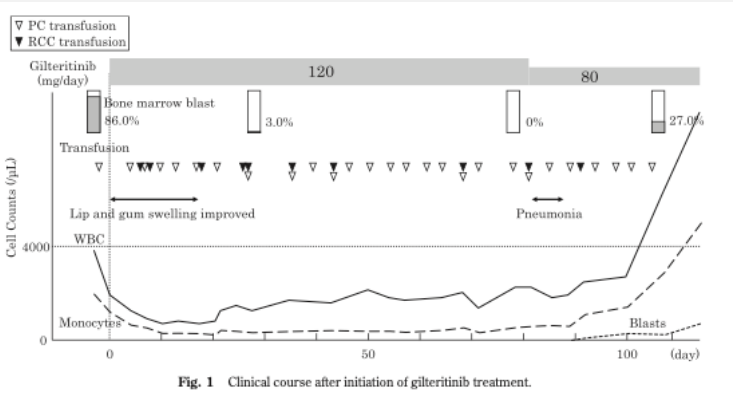


Fig. 1 Clinical course after initiation of gilteritinib treatment.

FLT3p.N676K変異陽性AML症例にギルテリチニブが奏功した報告

臨床血液 2022 年 63 巻 1 号 p. 51-54

Appendix

2. 本実習で使用したソフトとデータベースのversion


```
# SHIROKANE5
# SnpEffとSnpSift: version 5.0e (2021-03-09)
# CrossMap (v0.5.4)
# tabix (TAB-delimited file InderXer) Version: 0.2.5 (r1005)
#GRCh38.p13.RefSeq  (March 2019)

#  COSMIC
#source=COSMICv94
##reference=GRCh38
##fileDate=20220628

#  ClinVar 2021/8/14
##reference=GRCh38
#  dbSNPv151 20180418
##dbSNP_BUILD_ID=151
##reference=GRCh38.p7
#  Tohoku Medical Megabank
#8.3kjpn-2020083
##reference=GRCh37

#  Cancer Gene Census（2021/8/25時点のもの）
```

2.1 ソフトのインストール

- SnpEff は下記のウェブサイトからダウンロードしてください。
  SnpEff ダウンロードサイト：https://pcingola.github.io/SnpEff/

ダウンロード後、下記のコマンドで解凍してください。

unzip snpEff_latest_core.zip

解凍後の SnpEff フォルダー内には、下記のようなファイルがあります。

- snpEff
- LICENSE.md
- SnpSift.jar
- examples
- exec
- galaxy
- scripts
- snpEff.config
- snpEff.jar

この中で使用するファイルは、
snpEff.jar
SnpSift.jar

です。

2.3 SnpSiftアノテーション用カスタムデータベースのダウンロード

- 東北メディカルメガバンク機構 ToMMo 8.3KJPN Allele Frequency Panel (v20200831)

  東北メディカルメガバンクのデータベースは、日本人8,300名のSNPのデータベースで、これを使用することにより検出された変異の中から日本人特有のSNPを取り除くこ とができます。

  下記のウェブサイトから2つのファイルをダウンロードしてください。

  ToMMoダウンロードサイト：https://jmorp.megabank.tohoku.ac.jp/202102/

  - tommo-8.3kjpn-20200831-af_snvall-autosome.vcf.gz
  - tommo-8.3kjpn-20200831-af_snvall-autosome.vcf.gz.tbi

- ClinVar

  ClinVarはNCBIが提供するデータベースで、変異に関連する疾病のデータベースです 。
  ClinVarのデータベースを使用することにより、検出された変異が何らかの疾病に関連するかどうかを確認することができます。
  ClinVar のアノテーションデータベースを取得するには下記のコマンドを実行してください。
  データベースをダウンロードする場合は、使用するリファレンスを確認してください。

```
curl -O https://ftp.ncbi.nlm.nih.gov/pub/clinvar/vcf_GRCh38/clinvar_20210814.vcf.gz
curl -O https://ftp.ncbi.nlm.nih.gov/pub/clinvar/vcf_GRCh38/clinvar_20210814.vcf.gz.tbi
```

2.3 SnpSiftアノテーション用カスタムデータベース作成

- 公開データの取得と利用

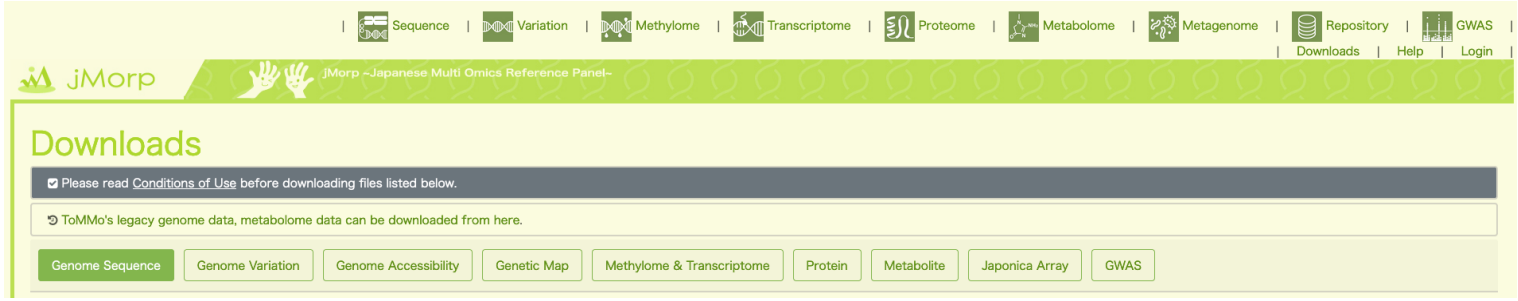
東北メディカルメガバンク機構　ToMMo 8.3KJPN Allele Frequency Panel (v20200831)

今回使用する東北メディカルメガバンクのデータベースは、日本人8,300名のSNPデータベースで、
これを使用することにより検出された変異の中から日本人特有のSNPを取り除くことができます。

下記をクリックしてください。
https://jmorp.megabank.tohoku.ac.jp/202102/

右上のDownloads をクリック

Genomic Variants をクリック



2.3 SnpSiftアノテーション用カスタムデータベース作成

tommo-8.3kjpn-20200831-af_snvall-autosome.vcf.gz をクリックしてダウンロード

同様に、tommo-8.3kjpn-20200831-af_snvall-autosome.vcf.gz.tbiをクリックしてダウンロード

（実習ではダウンロード済み）

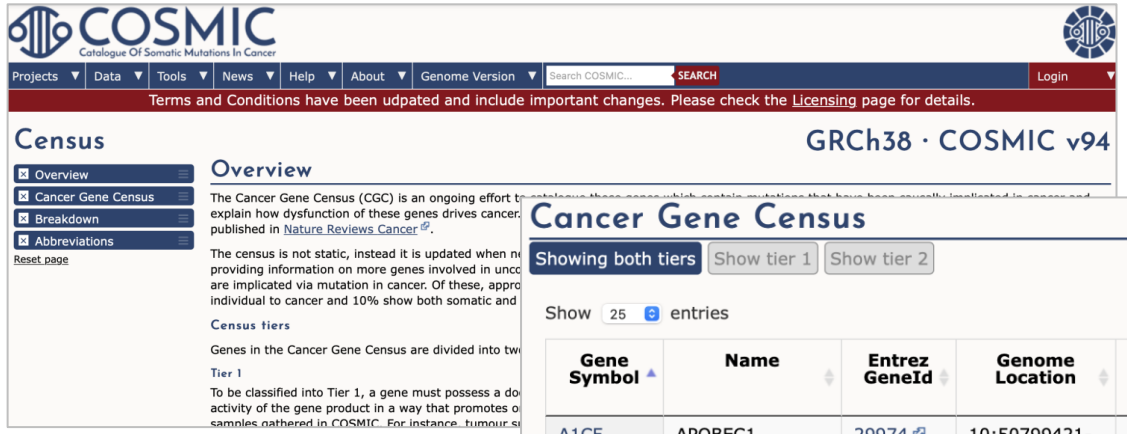
2.3 SnpSiftアノテーション用カスタムデータベース作成

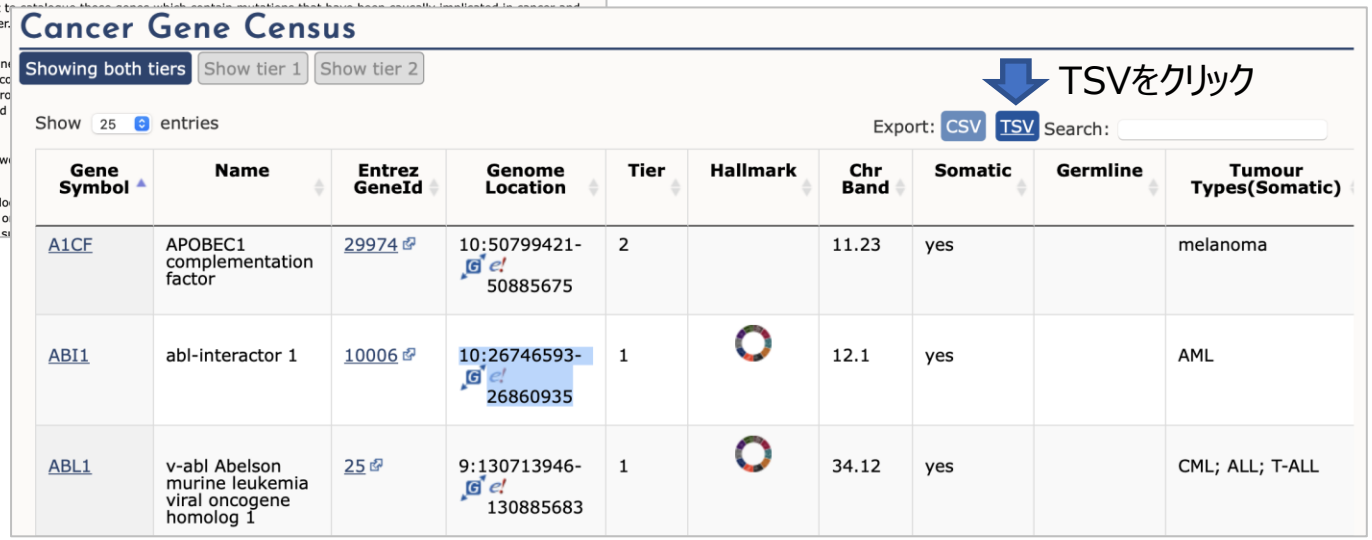
- CancerCensus

CancernCensusのデータベースは、がんに関連すると考えられる遺伝子、がんを促進させる可能性がある遺伝子のデータベースです。

下記のウェブサイトからファイルをダウンロードしてください。
https://cancer.sanger.ac.uk/census

Cancer Gene Census をクリック





Gene Symbol	Name	Entrez GeneId	Genome Location	Tier	Hallmark	Chr Band	Somatic	Germline	Tumour Types(Somatic)
A1CF	APOBEC1 complementation factor	29974	10:50799421-50885675	2		11.23	yes		melanoma
ABI1	abl-interactor 1	10006	10:26746593-26860935	1		12.1	yes		AML
ABL1	v-abl Abelson murine leukemia viral oncogene homolog 1	25	9:130713946-130885683	1		34.12	yes		CML; ALL; T-ALL

Census_allWed_Aug_25_01_47_38_2021.tsvをダウンロード（実習ではダウンロード済み）

2.3 SnpSiftアノテーション用カスタムデータベース作成

- ClinVar

ClinVarはNCBIが提供するデータベースで、変異に関連する疾病のデータベースです。
ClinVarのデータベースを使用することにより、検出された変異が何らかの疾病に関連するかどうかを確認することができます。

ClinVar のアノテーションデータベースを取得する為には下記のコマンドを実行してください。
データベースをダウンロードする場合は、使用するリファレンスを確認してください。

```
curl –O https://ftp.ncbi.nlm.nih.gov/pub/clinvar/vcf_GRCh38/clinvar_20210814.vcf.gz.tbi
curl -O https://ftp.ncbi.nlm.nih.gov/pub/clinvar/vcf_GRCh38/clinvar_20210814.vcf.gz
```

2.3 SnpSiftアノテーション用カスタムデータベース作成

- **COSMIC**

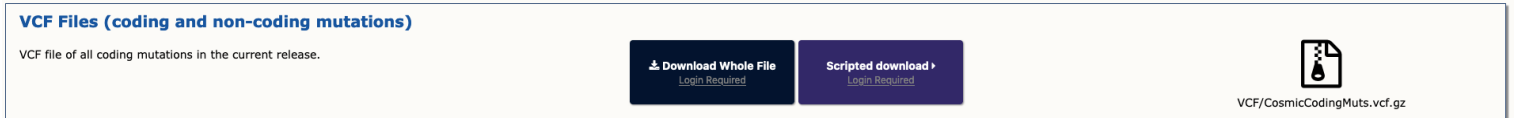
  COSMIC（Catalogue of Somatic Mutation in Cancer）はSanger Institute が提供する、がんと関連する体細胞変異の情報を集積したデータベースです。

  COSMIC のデータをダンロードするには、ユーザー登録をする必要があります。

  ログインして、vcf Files (coding and non-coding mutations) をダウンロードします

  https://cancer.sanger.ac.uk/cosmic/download

  下記をクリックしてください。



（実習ではダウンロード済み）

2.3 SnpSiftアノテーション用カスタムデータベース作成

CancerCensusデータの SnpSift で使用できるフォーマットに変換

ダウンロードしたファイル Census_allWed_Aug_25_01_47_38_2021.tsv を SnpSift で使用できる gmt ファイルフォーマットに変換します。
準備してある awk のスクリプトを使用して変換します。
下記のコマンドを実行してください。

```
awk -f census.awk Census_allWed_Aug_25_01_47_38_2021.tsv > Census_allWed_Aug_25_01_47_38_2021.gmt
```

2.3 SnpSiftアノテーション用カスタムデータベース作成

CancerCensus データ変換の awk script
（census.awk）

```
/^Gene Symbol/ {
    hn = split( $0, header, /¥t/ );
}

! /^Gene Symbol/ {
    dn = split( $0, data, /¥t/ );
    for ( i = 2; i <= dn; i ++ ) {
        printf header[ i ]”_"data[ i ]"#";
    }
    printf "¥t"data[ 2 ]"¥t"data[ 1 ]"¥n";
}
```

2.4 カスタムデータベースのGRCh38への座標変換

ToMMoなどのダウンロードした一部のカスタムデータベースは、GRCh37をreferenceとして作成されている為、 GRCh38 への変換が必要です

GRCh37 から GRCh38 の座標の変換は、CrossMap.py を使用します。
https://crossmap.readthedocs.io/en/latest/

例 CrossMap.py vcf <chain_file> <input.vcf> <refGenome.fa> <output_file> [options]

他にも liftOver なども同様に GRCh37 から GRCh38 への変換ツールです（一部マッピング出来ない場合があるのでその点に注意が必要です）。

https://genome-store.ucsc.edu/

tabix/bgzip は、genome 関連のファイルにインデックスをつけ、アクセスを高速化するツールです。
https://www.htslib.org/doc/tabix.html

圧縮された vcf ファイルを展開します。
gzip database.GRCh37.vcf.gz

CrossMap を使って vcf ファイルのゲノム座標を GRCh37 から GRCh38 に変換します。
CrossMapでの座標変換には、UCSCで提供されているchainファイルという GRCh37 と GRCh38をペアワイズにアラインメントしたファイル、
座標変換したいvcfファイルを用意します。
hg19ToHg38.over.chain.nochr database.GRCh37.vcf は以下でダウンロードできます。

https://hgdownload.cse.ucsc.edu/goldenpath/hg19/liftOver/

2.4 カスタムデータベースのGRCh38への座標変換

```
wget --timestamping   'ftp://hgdownload.cse.ucsc.edu/goldenPath/hg19/liftOver/hg19ToHg18.over.chain.gz'
        -O hg19ToHg18.over.chain.gz
```

まず初めにGRCh37のデータベースを、CrossMap.py を使用して、GRCh38のデータベースに変換します。
その後、tabix で提供されているツール bgzip を使用して圧縮し、tabix でインデックスを作成します。

```
CrossMap.py vcf hg19ToHg38.over.chain.nochr database.GRCh37.vcf
Homo_sapiens.GRCh38.dna.primary_assembly.fa database.GRCh38.vcf

bgzip database.GRCh38.vcf

tabix -p vcf database.GRCh38.vcf.gz
```

CrossMap.py は、python のライブラリです。
インストールするには、python3 をインストールして、下記のコマンドでインストールします。

```
pip3 install --user CrossMap
```

4.3 生物学的フィルタリング②

CGC Genes のレビュー課程で考慮されるhallmarks of cancerについて:

- 10種類のがんの特性 “hallmarks of cancer”*:

1. ゲノムの不安定化と変異 (Genome instability and mutation)
2. 無制限な複製による不死化 (Enabling replicative immortality)
3. 増殖抑制の回避 (Evading growth suppressors)
4. 細胞死 (Cell death)
5. エネルギー代謝のリプログラミング (Reprogramming energy metabolism)
6. 血管新生 (Angiogenesis)
7. 免疫による攻撃からの逃避 (Avoiding immune destruction)
8. 炎症の促進 (Tumor-promoting inflammation)
9. 増殖シグナルの維持 (Sustaining proliferative signaling)
10. 浸潤能および転移能の活性化 (Activating of invasion and metastasis)

* Hanahan D, Weinberg Robert A. Hallmarks of Cancer: The Next Generation. *Cell.* 2011; 144:646– 674.

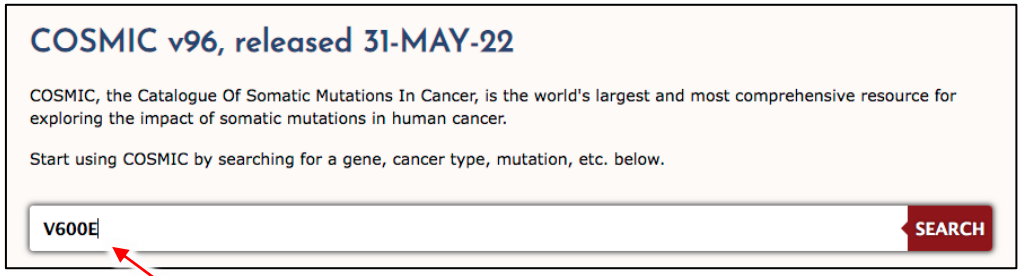
第Ⅴ章
データ解析発展
（解釈や論文参照など）

国立がん研究センター
先端医療開発センタートランスレーショナルインフォマティクス分野　ユニット長
山下理宇

COSMICでがんに関する情報の取得

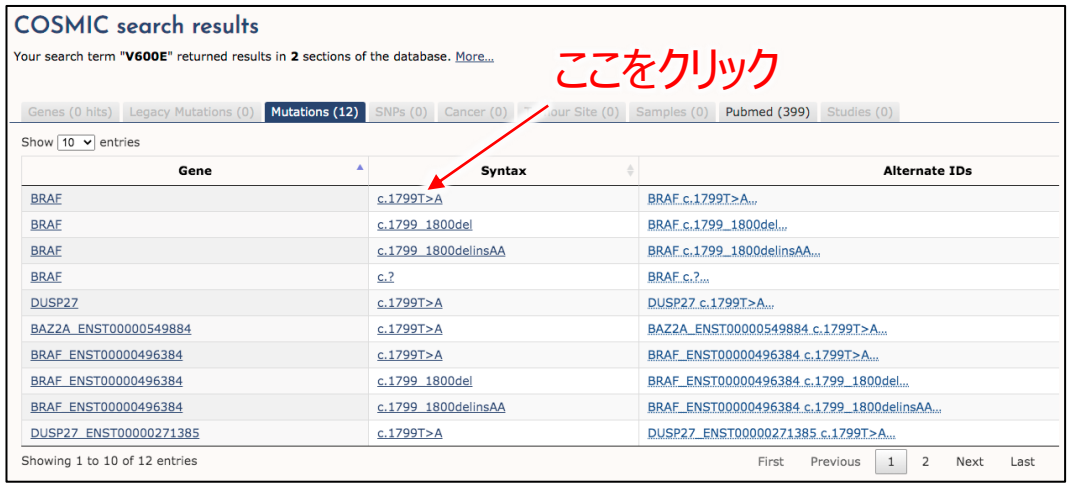
がん変異統合データベースであるCOSMICでデータを検索します。

1. COSMIC（https://cancer.sanger.ac.uk/cosmic)にアクセス

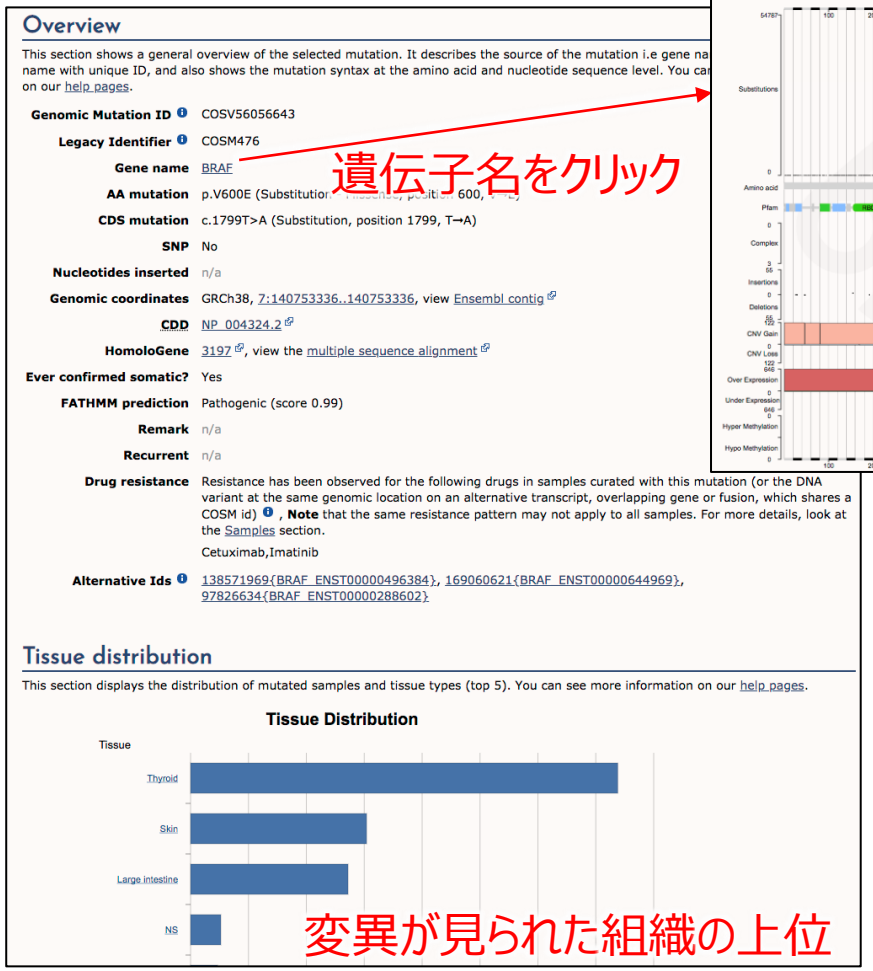


2. BRAFのV600Eを選択

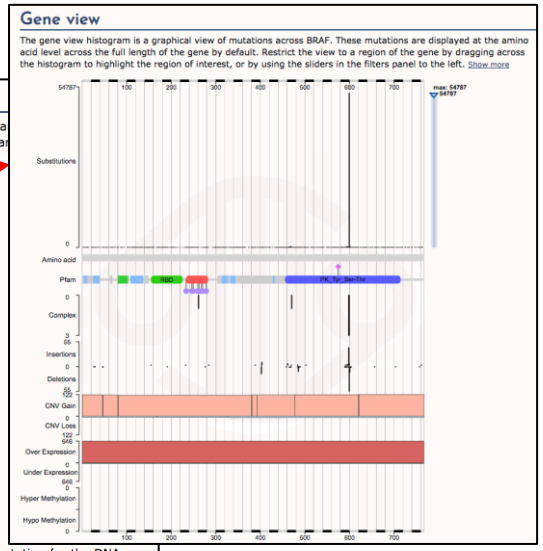
✓ 候補の中で調べたい変異をクリックする。
今回の場合はBRAFなので一番上が該当



3. 検索結果（Mutation）



Geneの画面に切り替わる。

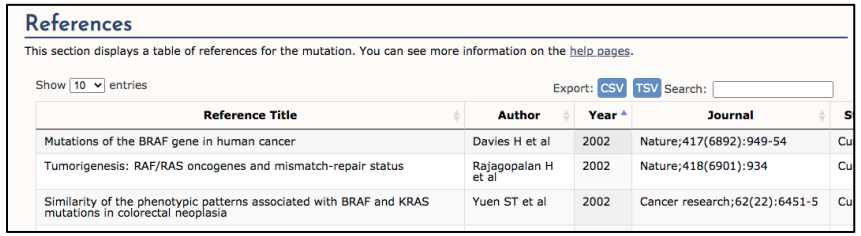


COSMICでがんに関する情報の取得

検索した変異に対して、論文やPathway(代謝・生化学経路) の情報を取得します。

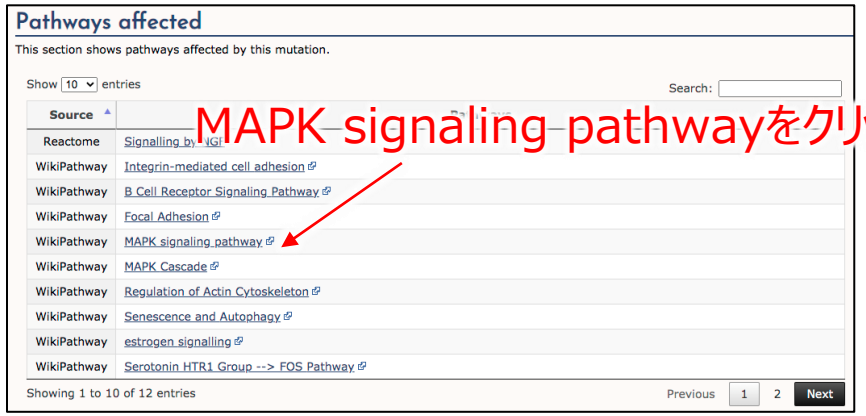
1. References: 変異や遺伝子に関わる論文へのリンク

- ✓ 前ページのV600Eを検索した直後の画面（Mutation, Gene）で表示されます。



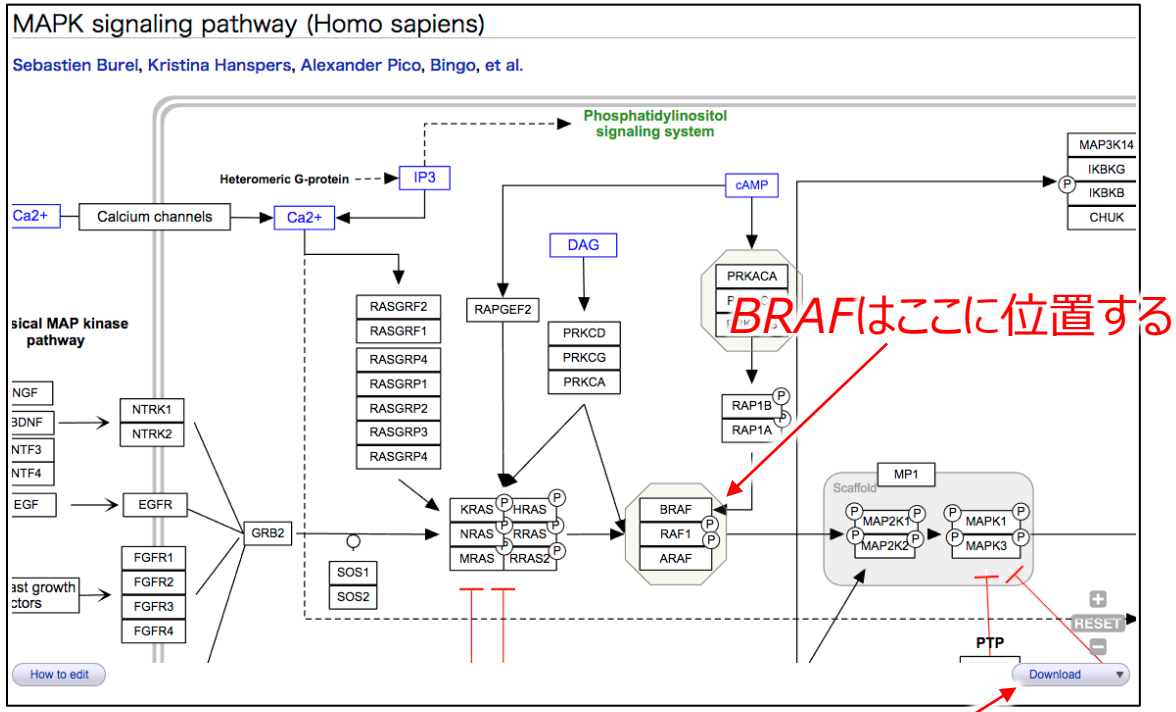
2. Pathways affected: 関連するpathwayの調査

- ✓ 遺伝子がどのPathwayにあるかWikiPathwaysのリンクがたどれます。
- ✓ 前ページのV600Eを検索した直後の画面（Mutation）で下の方に表示されます。



MAPK signaling pathwayをクリック

3. WikiPathwaysで興味のあるパスウェイを表示し、どこに位置するかを確認



BRAFはここに位置する。

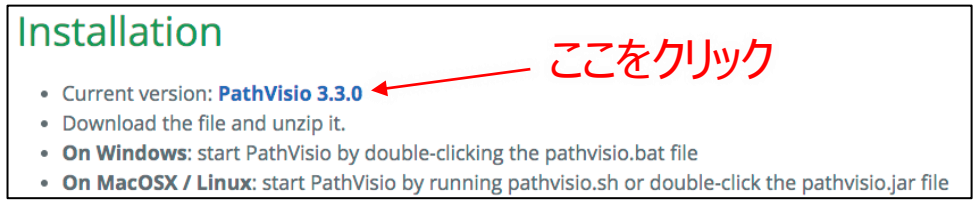
Pathwayの図を加工可能な形（GPMLフォーマット）でダウンロード

Pathway図の加工の仕方（PathVisio）

Pathwayの図をダウンロードしPathVisioで自分用に加工します。

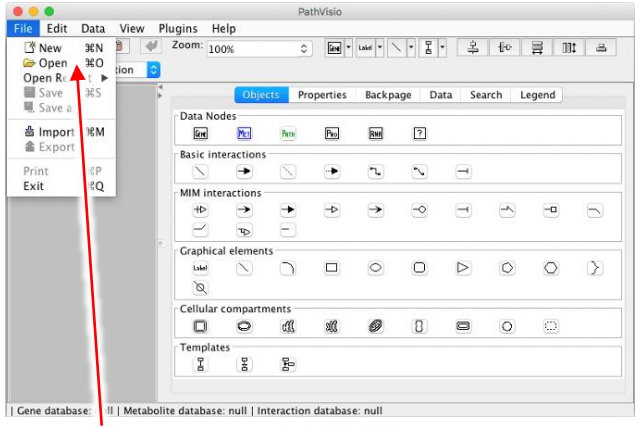
1. PathVisioをダウンロード

- ✓ https://pathvisio.github.io/downloads.htmlにアクセスする。

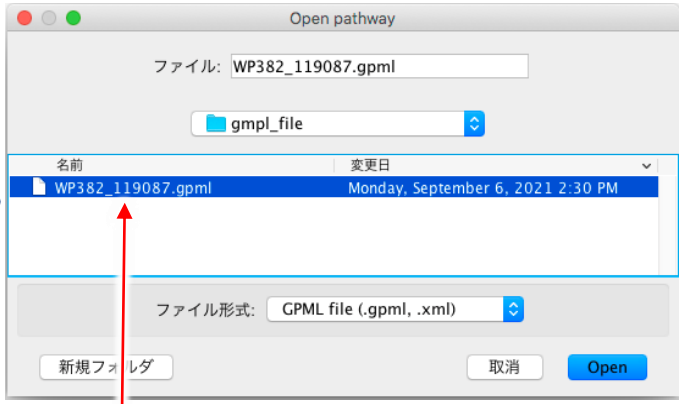


2. PathVisioの実行

- ✓ Windowsの場合には、 pathvisio.bat をダブルクリックする。
- ✓ Linux/MacOSの場合には、 pathvisio.jar をダブルクリックする。
- ✓ Fileからダウンロードしたgpmlファイルを選択する。

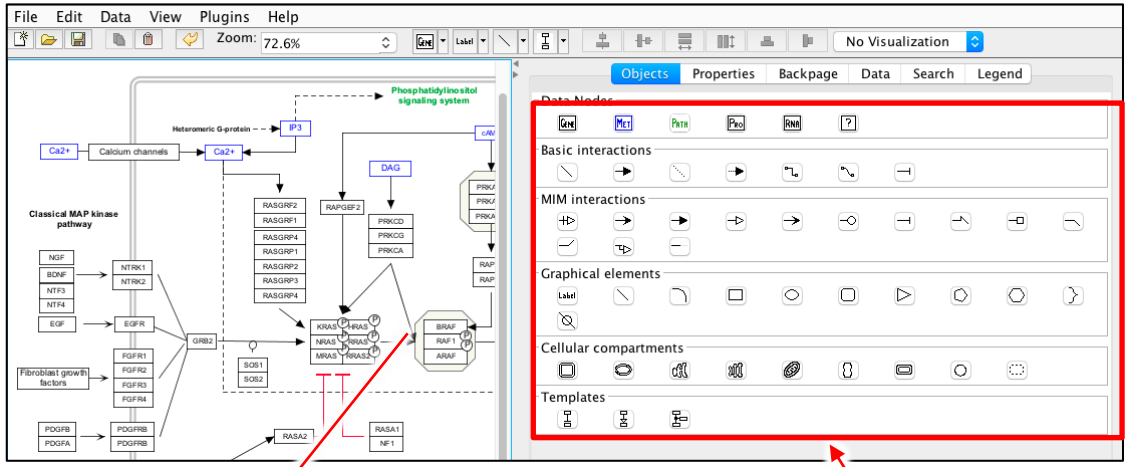


File-Openを選択



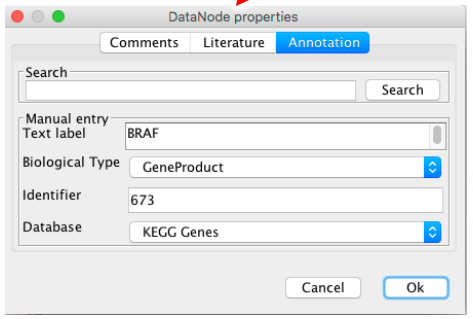
ファイルを選択するとPathwayが表示されます。

3. PathVisioの画面が開けます。



BRAFを選択

データを追加できます。



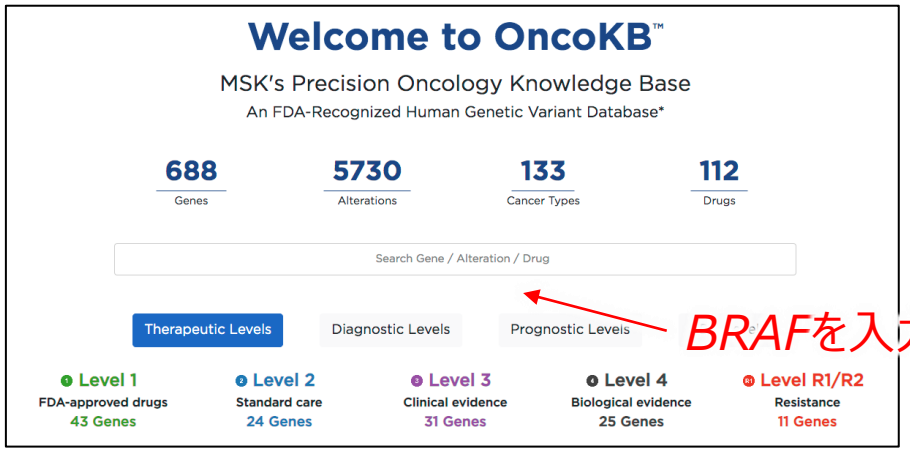
名前を変更したりコメントや論文情報を入れることができます。

OncoKBで遺伝子変異効果と薬剤の検索

OncoKBを使って特定のがんの遺伝子変異の効果や治療への影響に関す情報を入手します。

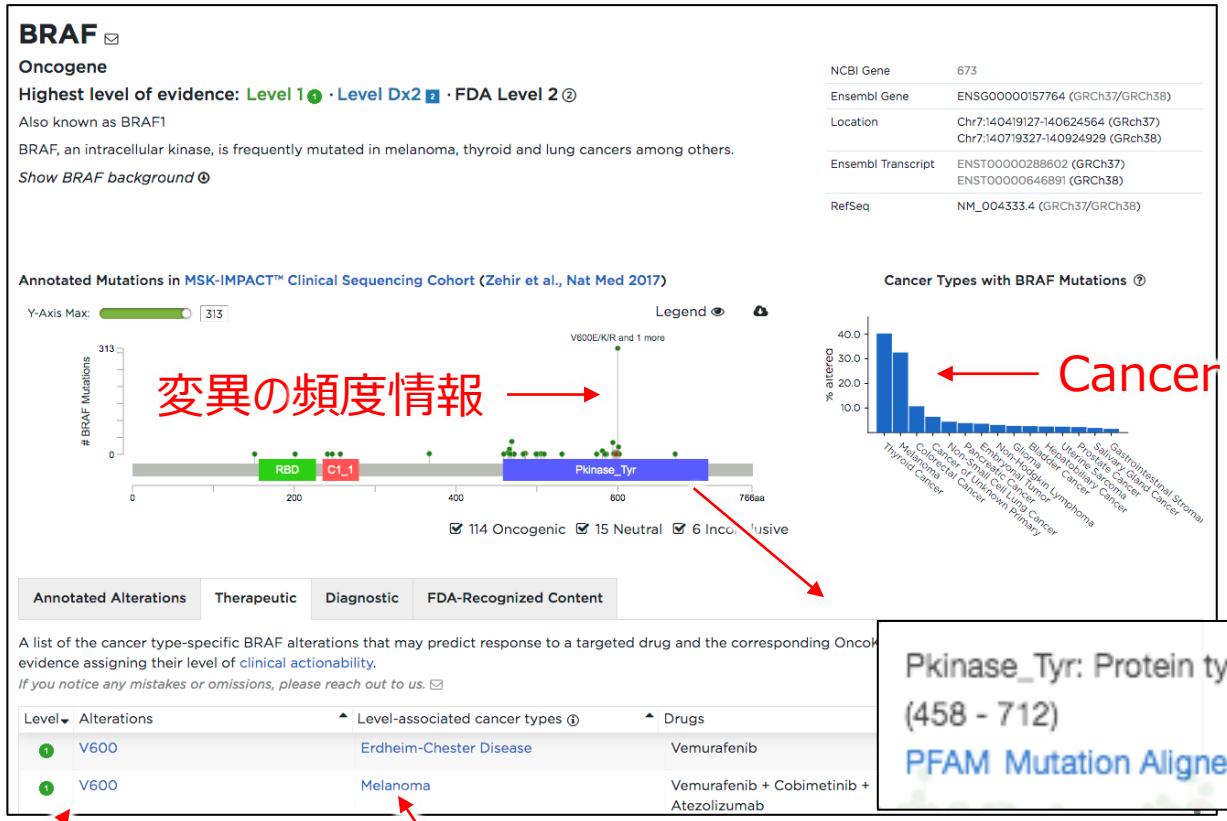
1. OncoKBにアクセスします。

- ✓ https://www.oncokb.org/にアクセスします。
- ✓ 薬剤の情報が下記のレベル別に登録されています。
  - ➢ Level 1 FDA-approved drugs
  - ➢ Level 2 Standard Care
  - ➢ Level 3 Clinical evidence
  - ➢ Level 4 Biological evidence
  - ➢ Level R1/R2 Resistance



BRAFを入力

2. OncoKBの画面の例



レベル別に表示

がん種ごとの情報詳細

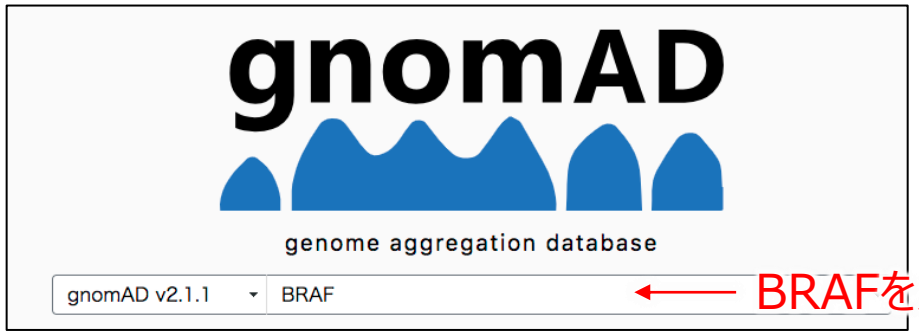
ドメインにカーソルを当てると機能が表示される。

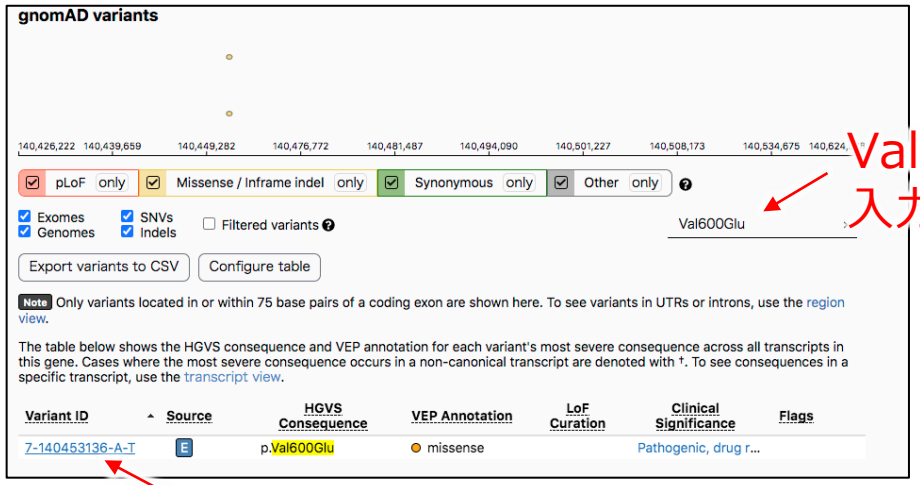
gnomADで変異の検索

gnomADは、変異情報がまとめられたデータベースです。Germlineの変異を調べるには非常に便利です。

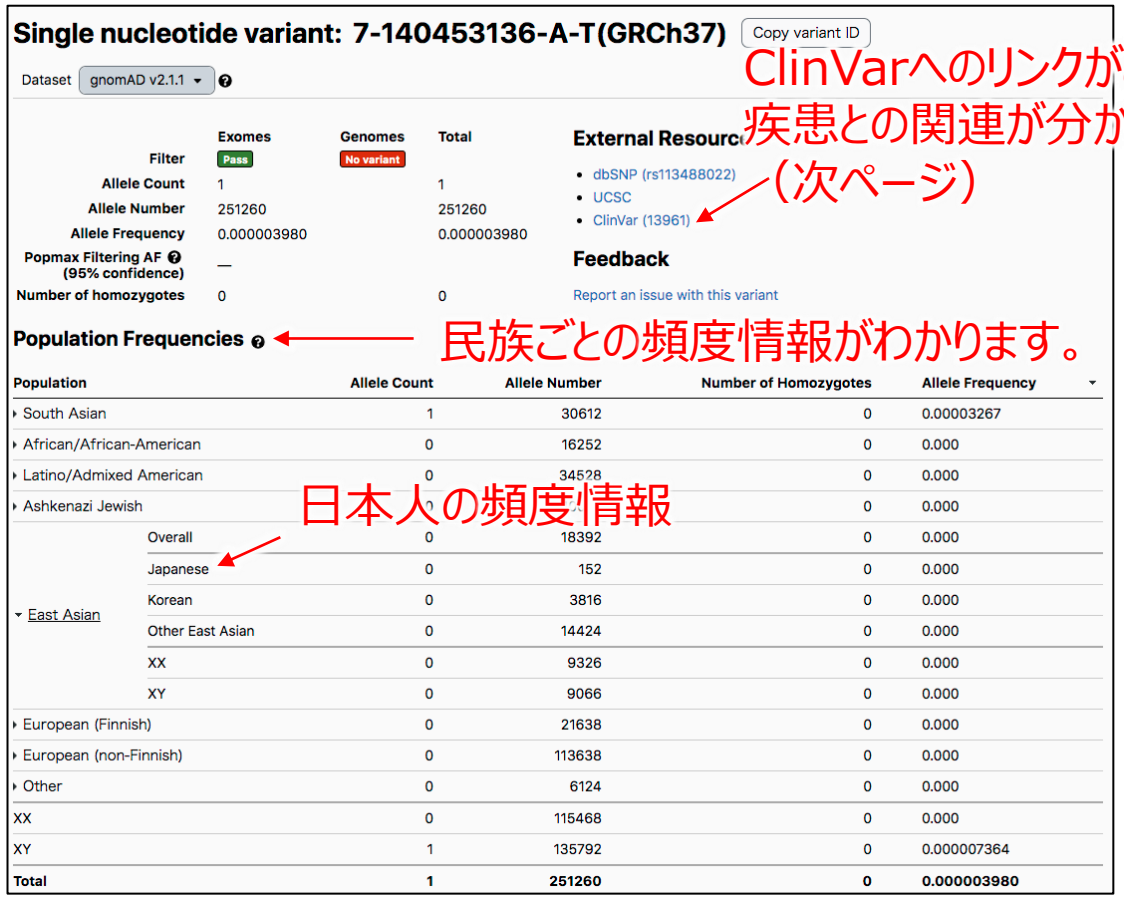
1. gnomADにアクセスします。

✓ https://gnomad.broadinstitute.org/にアクセスします。





2. 検索結果（人類集団ごとの頻度情報、他データベースへのリンク）



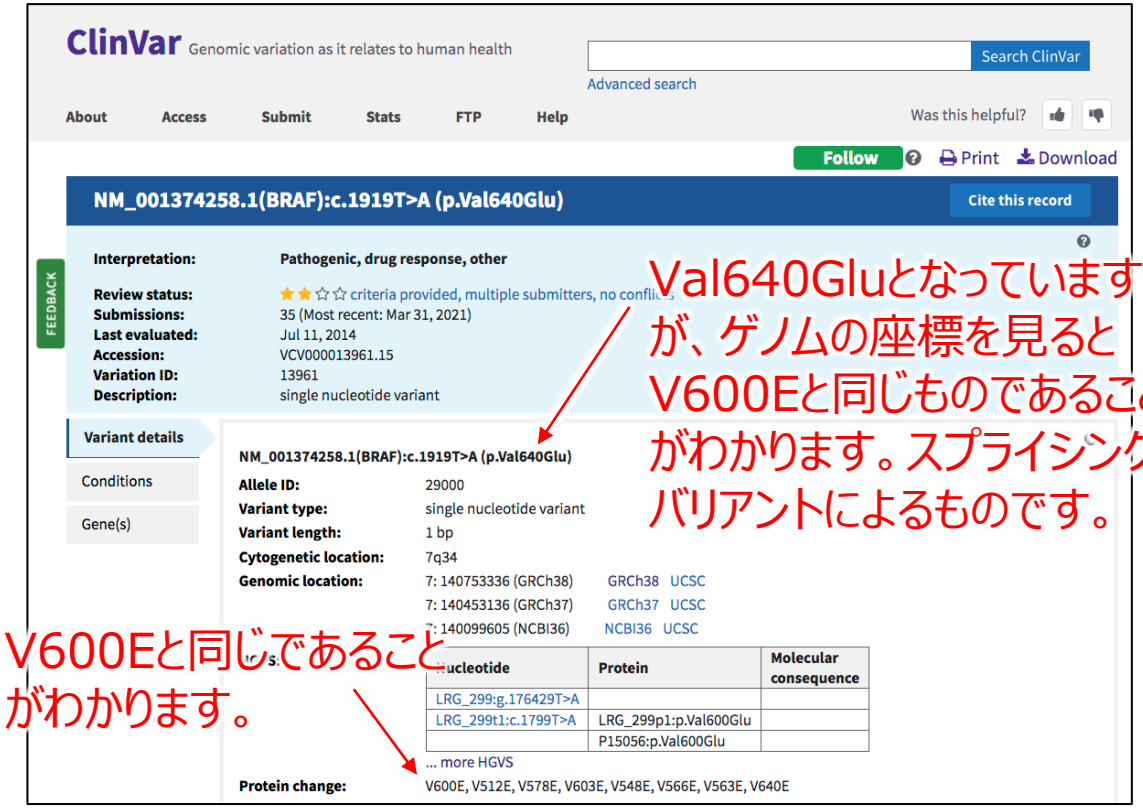
V600Eは、Germlineではほとんど観察されない変異であることがわかります。

ClinVarで変異と疾患との関連検索

ClinVarは、変異情報と疾患との関連をまとめたデータベースです。

1. ClinVarにアクセスします。

- ✓ https://www.ncbi.nlm.nih.gov/clinvar/にアクセスします。
- ✓ gnomADからのリンクからもたどれます。
  以下はgnomADのBRAF V600Eのリンクです。



2. 疾患との関連情報の一覧

V600Eに関連する疾患情報の一覧にアクセスすることができます。

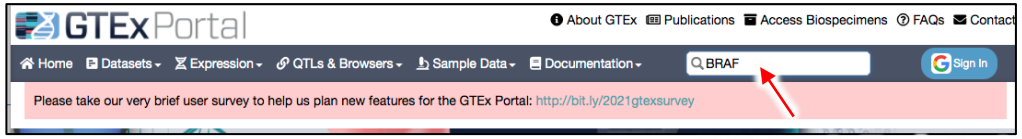
GTExで転写産物とeQTL情報の検索

GTExでは、RNA-seqに基づく遺伝子の転写産物や発現量、eQTL*についての情報が調べられます。

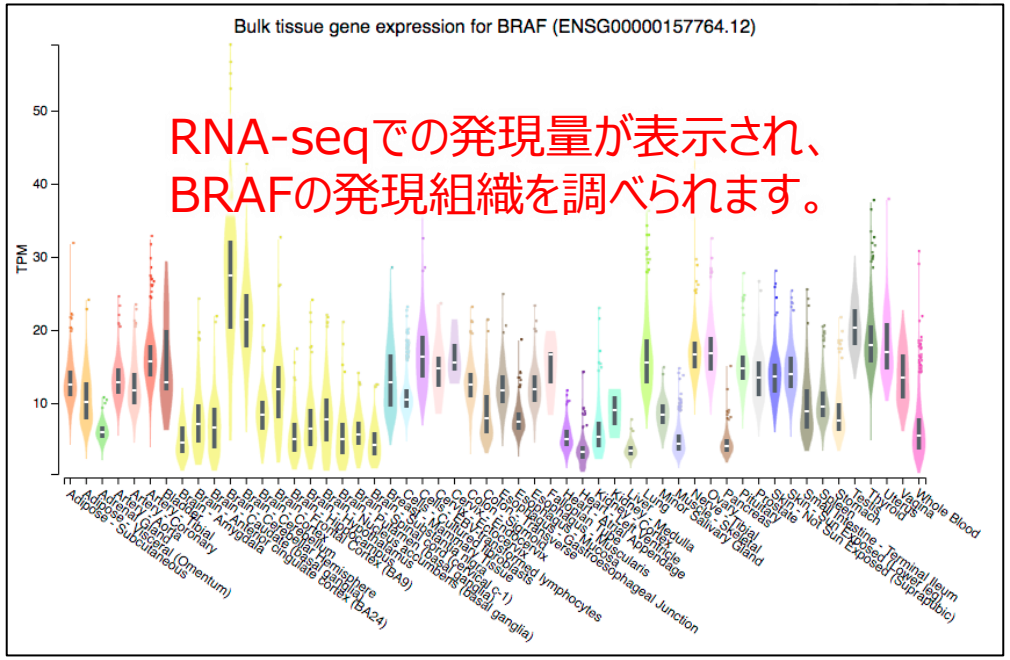
* eQTL (expression Quantitative Trait Locus)：近傍遺伝子等の発現量に影響を及ぼす座位

1. GTExにアクセス

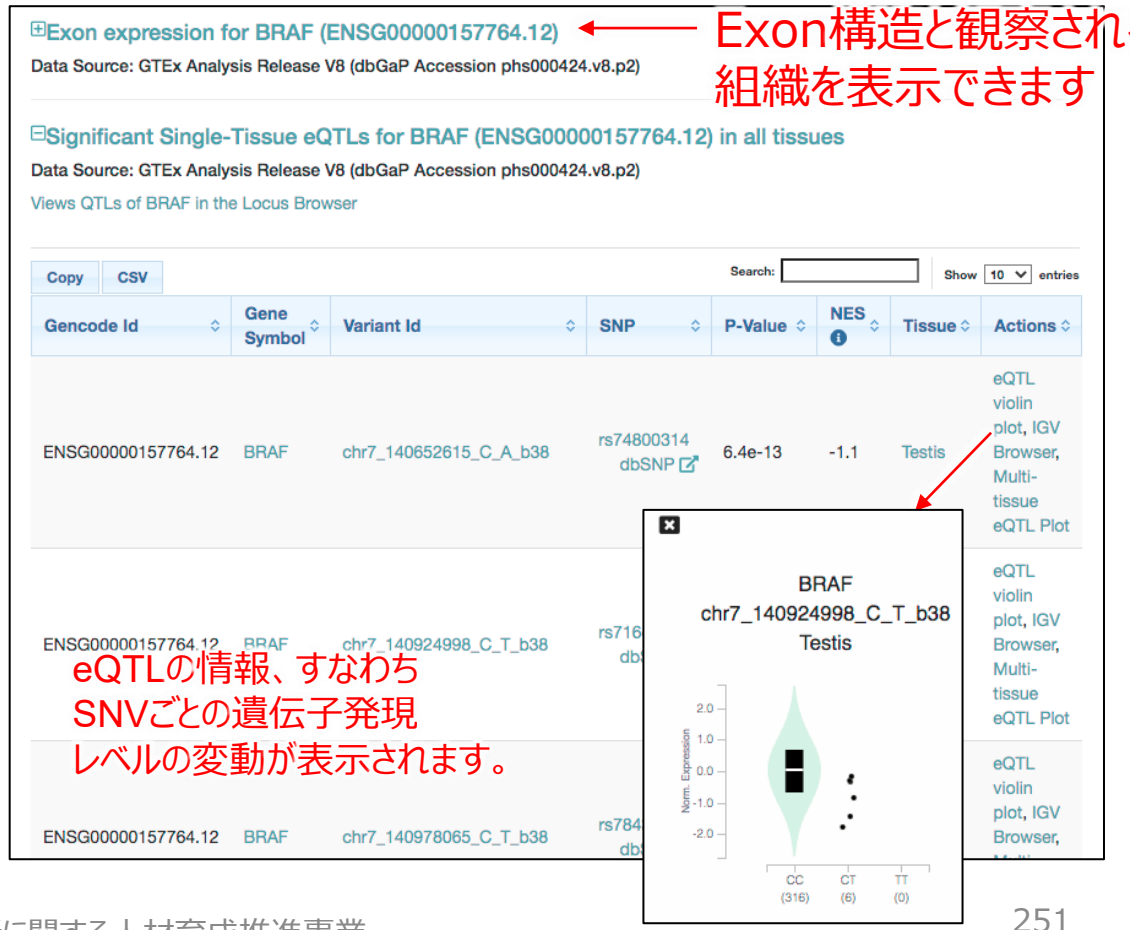
✓ https://gtexportal.org/home/にアクセスします。



BRAFを入力します



2. Exon構造、eQTL情報

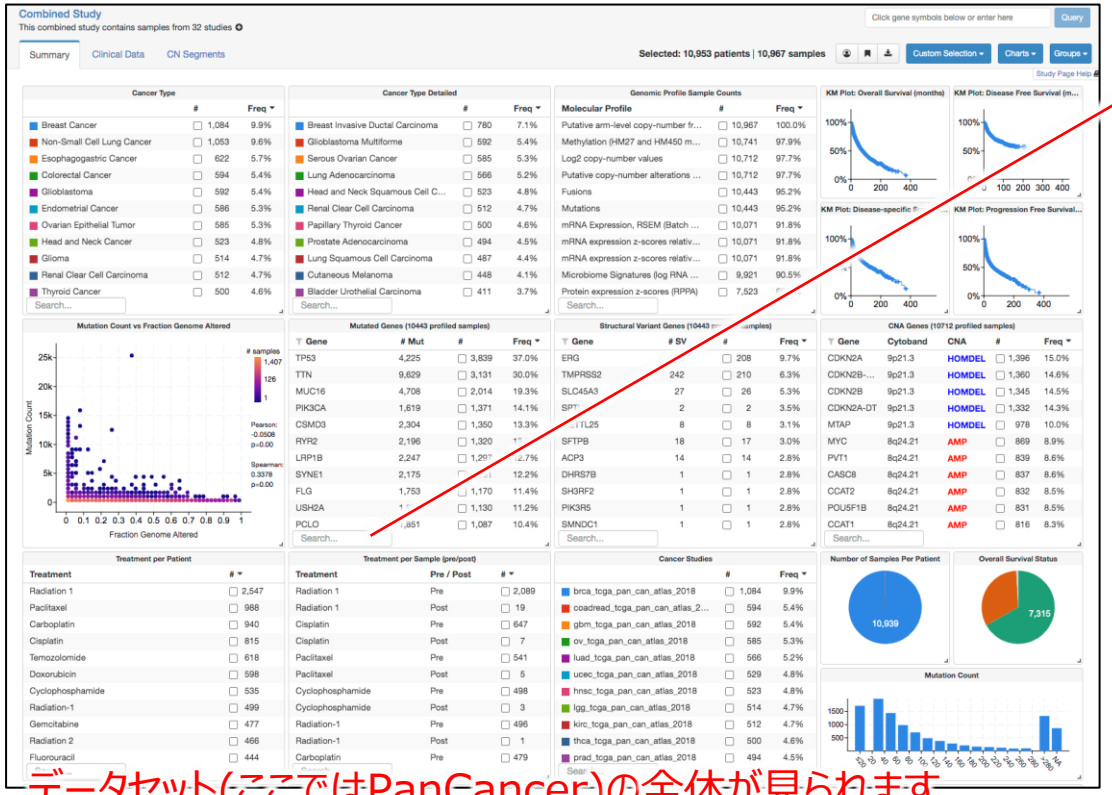


cBioPortalで生存曲線を求める

cBioPortalに格納されている情報を検索します。

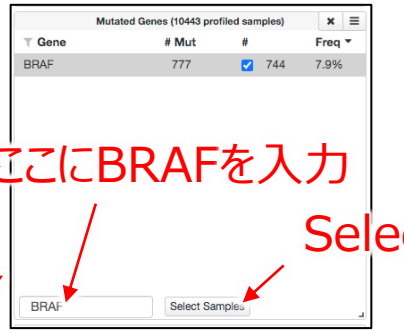
1. cBioPortal

- ✓ https://www.cbioportal.org/にアクセスします。
- ✓ TCGA PanCancer Atlas Studiesを選びます。
- ✓ 下の方にある「Explore Selected Studies」を選びます。

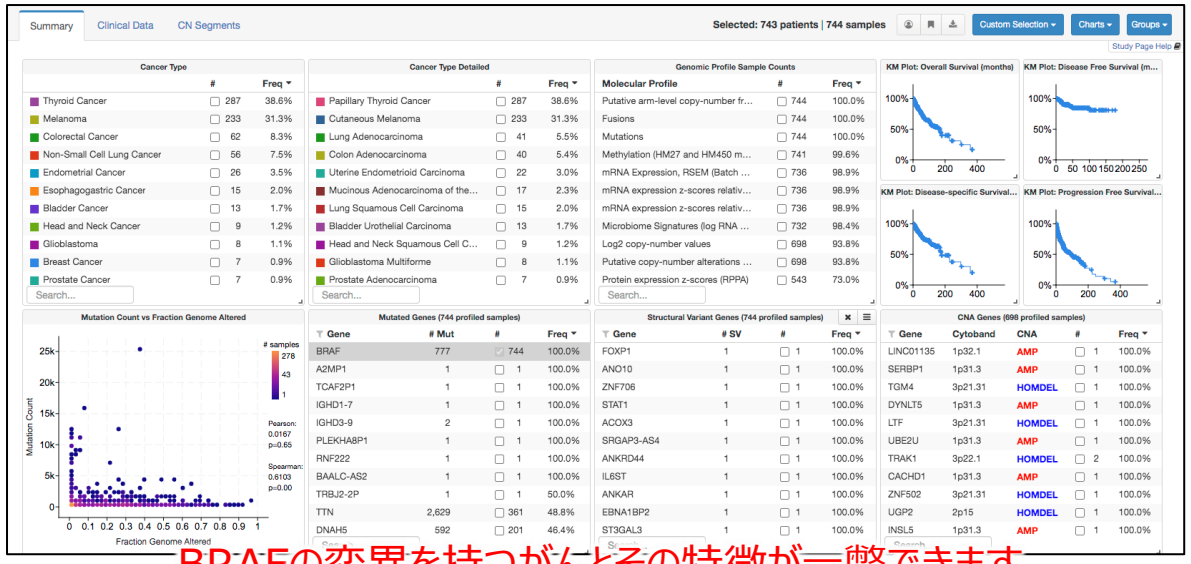


データセット(ここではPanCancer)の全体が見られます

2. Mutated GenesにBRAFを入れると以下の画面に遷移します。



3. BRAF変異を持つ患者のみの情報が抽出されます。



BRAFの変異を持つがんとその特徴が一瞥できます

cBioPortalで生存曲線を求める

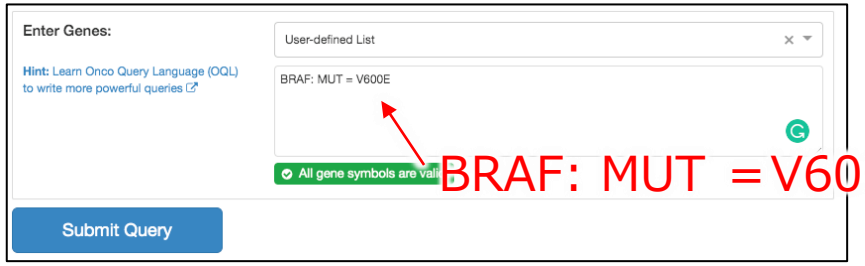
cBioPortalに格納されている膨大な変異と臨床情報を元に生存曲線を作成します。

1. cBioPortal

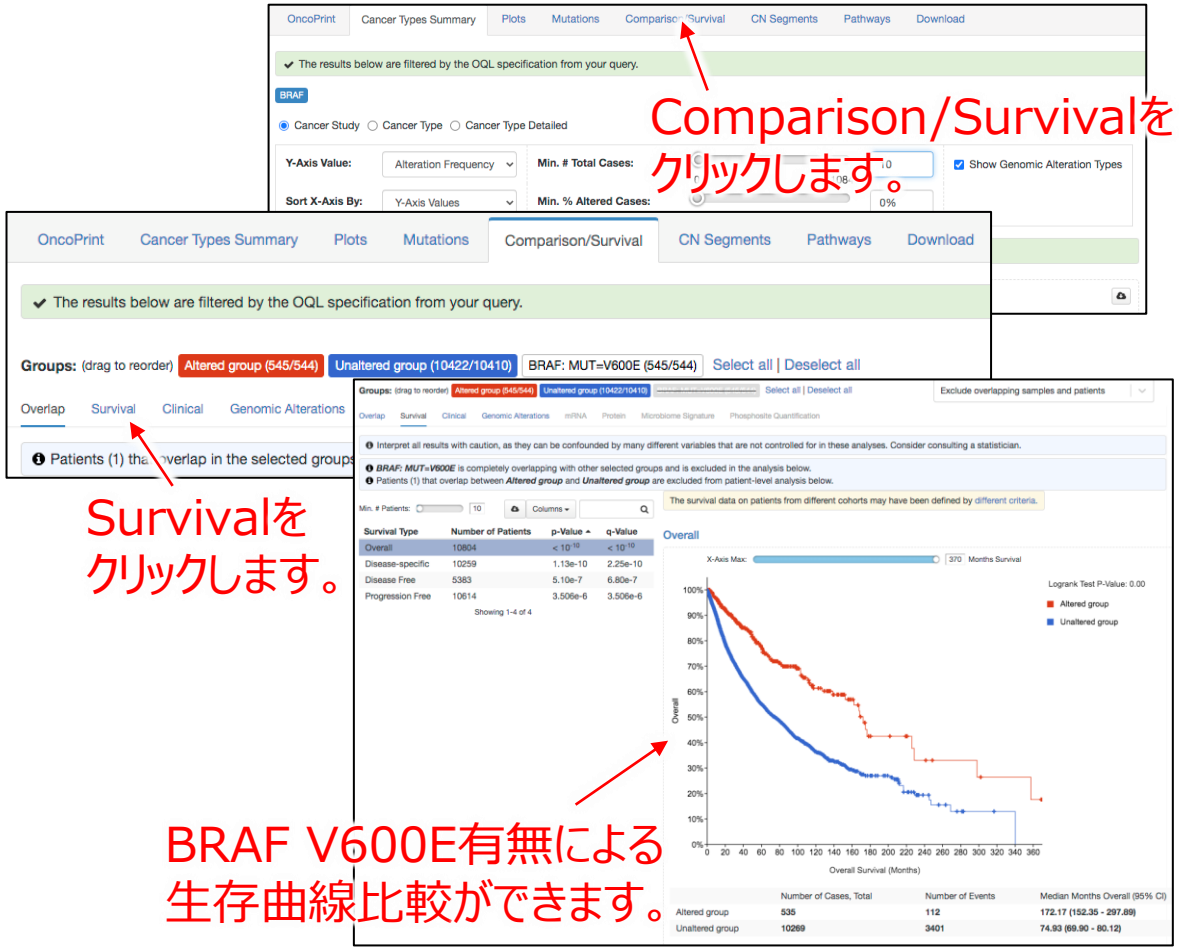
- ✓ https://www.cbioportal.org/にアクセスします。
- ✓ TCGA PanCancer Atlas Studiesを選びます。
- ✓ 下の方にある「Query By Gene」を選びます。

2. 検索したい変異を選びます。

- ✓ V600Eを検索したい場合には、以下のように入力します。
- ✓ 単にBRAFだけを検索することもできます。



3. Studyごとの変異頻度数が表示されます。



Tumor Mutation Burdenを求める（1）

準備とcoding region(翻訳領域)の非同義置換数の算出

1. Tumor Mutation Burden（TMB）について

- ✓ TMBは、腫瘍ゲノムのcoding regionに存在するsomaticな非同義置換の総数として定義され、1Mb中にある非同義置換数と挿入・欠失変異数の合計で表されます。
- ✓ パネルや報告ごとに、若干算出方法が異なります。特に、somatic変異のコール方法が異なると値が変わってくるので、過去の報告と相関していることを確認する必要があります。
- ✓ 参考: *Transl Lung Cancer Res.* 2018 Dec; 7(6): 661–667.

2. 今回使うファイルの準備

- ✓ 以下にアクセスしてHiseqXのHighCoverageのデータCOLO-829, HCC-1143, HCC-1187のデータを入手します。

https://www.nygenome.org/bioinformatics/3-cancer-cell-lines-on-2-sequencers/

Data Availability

The raw data is available on dbGAP.

The somatic variant files, obtained from the high-coverage data are accessible below (or directly in Variants.HighCoverage.tar.gz 140MB):

Cell line	SNV/indel	CNV	SV	SV high confidence
COLO-829 (HiSeqX)	VCF	BED	bedpe	bedpe
COLO-829 (NovaSeq)	VCF	BED	bedpe	bedpe
HCC-1143 (HiSeqX)	VCF	BED	bedpe	bedpe
HCC-1143 (NovaSeq)	VCF	BED	bedpe	bedpe
HCC-1187 (HiSeqX)	VCF	BED	bedpe	bedpe
HCC-1187 (NovaSeq)	VCF	BED	bedpe	bedpe

3. ファイルの中身の確認

✓以下の3つのファイルがあることを確認。全てVCFフォーマット。

名前	変更日	サイズ
COLO-829--COLO-829BL.snv.indel.final.v6.annotated.vcf	20:07	36.2 MB
HCC-1143--HCC-1143BL.snv.indel.final.v6.annotated.vcf	20:07	7.1 MB
HCC1187--HCC-1187BL.snv.indel.final.v6.annotated.vcf	20:07	4.3 MB

4. TMBという名前のディレクトリを作成し、そこに全てのファイルを格納する。

✓例えば、ターミナルが使える環境で以下の操作を行う。
✓必要に応じてスパコン等にscpを行う。

```
$ mkdir TMB
$ cd TMB
$ ls
COLO-829--COLO-829BL.snv.indel.final.v6.annotated.vcf
HCC-1143--HCC-1143BL.snv.indel.final.v6.annotated.vcf
HCC1187--HCC-1187BL.snv.indel.final.v6.annotated.vcf
```

5. ターミナルから以下のコマンドを打ち込む（COLO-829用）

```
$ grep -a 'protein_coding' COLO-829--COLO-829BL.snv.indel.final.v6.annotated.vcf |grep -v 'intron_variant' |cut -f 8 |perl -pe 's/¥|/¥t/g' |cut -f 4 |sort|  uniq -c|awk '{print $2"¥t"$1}
```

注: ¥ はバックスラッシュと同じです。

Tumor Mutation Burdenを求める（2）

Tumor Mutation Burden（TMB）の計算

6. 5のコマンドの説明

- ✓ `grep -a 'protein_coding'`
  ファイルの中から「protein_coding」すなわち転写領域のみを抽出
  -a でファイルをテキストとみなす。
- ✓ `grep -v 'intron_variant'`
  「intron_variant」を除く。これで転写領域からイントロン部分を除く。
- ✓ `cut -f 8`
  8列目のアノテーションが付いている列だけを抜き出す。
- ✓ `perl -pe 's/¥|/¥t/g'`
  「|」をタブに置換して列に分解する。
- ✓ `cut -f 4`
  分解したうち、4列目（結果の変異の種類に相当）のみを抽出する。
- ✓ `sort|uniq -c`
  並べ替えて重複をカウントする。
- ✓ `awk '{print $2"¥t"$1}`
  1列目と2列目を入れ替える。

7. シークエンスされた遺伝子領域のサイズを決める。

- ✓ 元のダウンロードファイルは全ゲノムの結果なので、coding regionは、ゲノム上のcoding regionと一致します。今回の場合、ダウンロードファイルでリファレンスとした正確なcoding regionが分かりません。
- ✓ そこで、UCSC genome browser にある refGene.txt から coding region を重複部分を取り除いて計算した結果の 33.3Mb を利用します。
- ✓ refGene.txtには、refSeqのNM（キュレーションされたmRNAのセット）の座標情報があります。

8. 非同義置換・挿入・欠失を定義し、TMBを計算する

- ✓ 5のコマンドの結果を表にまとめる
  - ➢ 非同義置換・挿入・欠失のみを選び出してカウントし、33.3で割る。
  - ➢ V600Eを持つCOLO-829のTMBが高いことがわかる。

これらを非同義置換・挿入・欠失とする

変異の種類	COLO-829	HCC-1143	HCC-1187
3_prime_UTR_variant	948	220	121
5_prime_UTR_variant	337	42	22
coding_sequence_variant&5_prime_UTR_variant	3	0	1
downstream_gene_variant	2748	587	316
splice_acceptor_variant	11	3	1
splice_donor_variant	10	1	1
splice_region_variant&5_prime_UTR_variant	3	1	0
splice_region_variant&synonymous_variant	3	1	0
synonymous_variant	322	65	29
upstream_gene_variant	3670	714	467
frameshift_variant	42	5	14
frameshift_variant&splice_region_variant	2	1	0
inframe_deletion	17	5	2
inframe_insertion	1	0	1
missense_variant	857	170	96
missense_variant&splice_region_variant	23	1	6
protein_altering_variant	1	0	0
start_lost	24	0	0
stop_gained	65	20	7
stop_gained&splice_region_variant	3	0	1
stop_lost	1	0	0
stop_lost&3_prime_UTR_variant	1	0	0
非同義置換数	**1037**	**202**	**127**
TMB(/1Mb)	31	6.1	3.8

注：今回ご紹介した方法は、ここで使用したVCFファイルに特化していますので、他のVCFファイルで同様にはできないかもしれません。ただし、アイデアは同じです。

Rの基本的な操作と変数

Rでディレクトリの移動、データ構造等の基本的な知識です。

- **ワーキングディレクトリの確認**
  - ✓ 自分が今いるディレクトリの場所を確認します。

```
> getwd()
[1] "/Users/user name"
```

- **ワーキングディレクトリの作成**
  - ✓ 作業用ディレクトリworkを作成します。
  - ✓ .は今自分のいるディレクトリ。今いるディレクトリ直下にworkができる。

```
> dir.create("./work")
```

- **ワーキングディレクトリの移動**
  - ✓ 作業用ディレクトリに移動します。ここでは、自分のホームディレクトリにworkというディレクトリを作って作業します。

```
> setwd("/Users/user name/work")
> getwd()
[1] "/Users/user name/work"
```

- **Rの変数型：数値型**
  - ✓ 数値型は、数を取り扱い四則演算ができます。

```
> num = c(1)
> num
[1] 1
```

- **Rの変数型：文字列型**
  - ✓ “”で囲むと文字列型として認識されます。このままでは四則演算はできません。

```
> num = c("1")
> num
[1] "1"
```

- **Rの変数型：論理値型**
  - ✓ TRUEかFALSEのいずれかを取ります。文字列と違って””でかこまれていません。

```
> x = TRUE
> x
[1] TRUE
```

Rのデータ構造

Rには、データをまとめて取り扱えるデータ構造としてベクトル、行列、配列、リスト、データフレーム、因子、順序付き因子があります

- **ベクトル（Vector）**
  - ✓ 複数の数値を格納して取り扱えます。

```
> x = c(1, 2, 3, 4, 5, 6)
> x
[1] 1 2 3 4 5 6
```

- **因子（Factor）**
  - ✓ 文字列に対して整数を割り振ったもので質的係数を表す
  - ✓ 例えば、男性1, 女性2として、as.factor()で定義する

```
> x = c( 1, 1, 1, 2, 2, 2 )
> x
[1] 1 1 1 2 2 2
> x = as.factor( x )
> x
[1] 1 1 1 2 2 2
Levels: 1 2
```

  - ✓ 因子に順序が付いた順序付き因子という型もあります。

- **データフレーム型**
  - ✓ Excelの表のイメージで、ファイルを読み込んだときの基本形です。
  - ✓ 中身は以下のようなテーブル形式です。

name	sex	class	score	passed
Giyu	male	A	68	TRUE
Tengen	male	B	98	TRUE
Kyojyuro	male	A	32	FALSE
Sanemi	male	A	64	TRUE
Mitsuri	female	B	72	TRUE
Shinobu	female	B	100	TRUE

  - ✓ 一番上はheader行です。列名として利用できます。
  - ✓ headerを定義しておくと、その名前で参照できます。
  - ✓ 行数・列数は全てで同じである必要があります。

＊ その他のデータ構造には、行列（一般的な行列）、リスト（異なるデータ構造をまとめて1つのオブジェクトとしたもの）がありますので、必要に応じて調べてみて下さい。

ファイルからのデータの読み込みと内容確認

データを読み込み中身を確認します。

- **データの読み込み**
  - ✓ read.tableで読み込んだデータをdf（データフレーム）に代入します。
  - ✓ ~/workにあるiris.txtを読み込みます。
  - ✓ sepで列の区切り文字を指定します。
  - ✓ header=TRUEを指定すると、headerでアクセスできます。

```
> df = read.table("~/work/iris.txt", sep="¥t", header=TRUE)
```

- **データの内容を確認**
  - ✓ headでdfの最初の部分を表示できます。
  - ✓ 単にdfとすると全てのデータを表示しようとします。

```
> head(df)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
…
```

- **データの行数・列数を確認**
  - ✓ headerは含まれない。

```
> dim(df)
[1] 150   5
```

- **summaryによるデータの要約**
  - ✓ Summaryでデータの要約ができる。
  - ✓ 因子型になっている列に対しては、数がカウントされる。

```
> summary(df)
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width          Species
 Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100   setosa    :50
 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300   versicolor:50
 Median :5.800   Median :3.000   Median :4.350   Median :1.300   virginica :50
 Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
 3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
 Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
```

- **データの内部へのアクセス**
  - ✓ 2行目のデータにアクセス

```
> df[2,]
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
2          4.9           3          1.4         0.2  setosa
```

  - ✓ 3列目のデータにアクセス
  - ✓ df[,3]でもアクセスできるが、列名でアクセスした方がデバッグが容易

```
> df$Petal.Length
[1] 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 1.5 1.6 …
```

Signature解析 (1)

deconstructSigsを使ってsignature解析を行います。

Signature 解析とは

変異の塩基置換パターンを分類することで、変異の発生過程や関連する背景因子などを推定する解析です。
例えばC>Aの変異が多いSignature.4に分類されるとその患者さんの変異は、喫煙が要因であると推定されます。

- **変異情報のファイルを作成（mutations.txt）**
  - ✓ VCFから下記の様なタブ区切りの5列のテキストファイルを作成する
  - ✓ cutコマンドで当該部分を取ってくる、または、Excelで行っても良い
  - ✓ sample名、染色体番号、変異の位置、ref allele, variant alleleの順番
  - ✓ sampleが複数ある場合には重ねていく
  - ✓ ファイルのmutations.txtとして保存

```
Sample        chromosome    start         ref           variant
COLO-829      chr1          183180        G             A
COLO-829      chr1          761264        G             A
 :
HCC-1143        chr1    66264   A       AT
 :
HCC1187_80      chr1    514015  T       A
 :
```

- **サンプルのファイルを作成（sample_def.txt）**
  - ✓ サンプル名を入力したテキストファイルを作成する
  - ✓ sampleが複数ある場合には重ねていく
  - ✓ mutations.txtファイルから一気に作る場合には、ターミナルでunixシェル上で以下のコマンドでできる

```
cut -f 1 mutations.txt | uniq > sample_def.txt
```

  - ✓ 以下の様なファイルが同じディレクトリにsample_def.txtという生であればOK

```
Sample
COLO-829
HCC-1143
HCC1187_80
```

- **必要なファイルが存在しているか確認(unixシェル上での操作)**
  - ✓ lsコマンドで、mutations.txtとsample_def.txtのファイルがあることを確認する
  - ✓ lessコマンドで、mutation.txtとsample_def.txtのファイルが例のようになっていることを確認する

Signature解析 (2)

deconstructSigsを使ってsignature解析を行います。以下の操作はRで行います。

- **deconstructSigsをインストール**
  - ✓ Rを起動させる。ターミナルでRと打つとRが起動する
  - ✓ 以下のコマンドで必要なパッケージをインストール
  - ✓ “BSgenome.Hsapiens.UCSC.hg19”は、本来ならば不要であるが、Rのversionによっては、 deconstructSigsのインストール時に要求されることがある。
  - ✓ 別ウィンドウでどのサーバーからインストールするかの画面が出る場合には、Japanを選べば良い。

```
#Bioconductorの読み込み
if (!require("BiocManager", quietly = TRUE))
    install.packages("BiocManager")

# 4つのライブラリをインストール
# “BSgenome.Hsapiens.UCSC.hg19”は、本来ならば不要であるが、Rのversionによっては要求されることがある。
BiocManager::install("BSgenome", "BSgenome.Hsapiens.UCSC.hg19", "GenomeInfoDb”, "BSgenome.Hsapiens.UCSC.hg38")

install.packages("deconstructSigs")
```

- **ライブラリのロード**
  - ✓ deconstructSigsとゲノム情報をロードする

```
# インストールしたライブラリを読み出す
library(deconstructSigs)
library(BSgenome.Hsapiens.UCSC.hg38)
```

- **deconstructSigsの実行**

```
# データの読み込み
mut_df <-read.table("mutations.txt", header=T)
sigs.input <- mut.to.sigs.input(
  mut.ref = mut_df,
  sample.id = "Sample",
  chr = "chromosome",
  pos = "start",
  ref = "ref",
  alt = "variant",
  bsg = BSgenome.Hsapiens.UCSC.hg38
)

list_df<-read.table("sample_def.txt", header=T)
ids<-c(as.vector(list_df$Sample))

pdf("out.pdf")
par(mfrow=c(length(ids),1))
result = c()

# signature解析と作図
for (id in ids){
  print(id)
  each_sig <- whichSignatures(
    tumor.ref = sigs.input,
    signatures.ref = signatures.nature2013,
    contexts.needed = TRUE,
    sample.id = id,
    tri.counts.method = genome
  )
  res_line = each_sig$weight

  #図を生成
  plotSignatures(each_sig, sub = "test")
  result <- rbind(result, res_line)
}
dev.off()

#結果をファイルに保存
write.table(result, file="out.txt", sep = "¥t", quote = F, row.names = T, col.names =T)
```

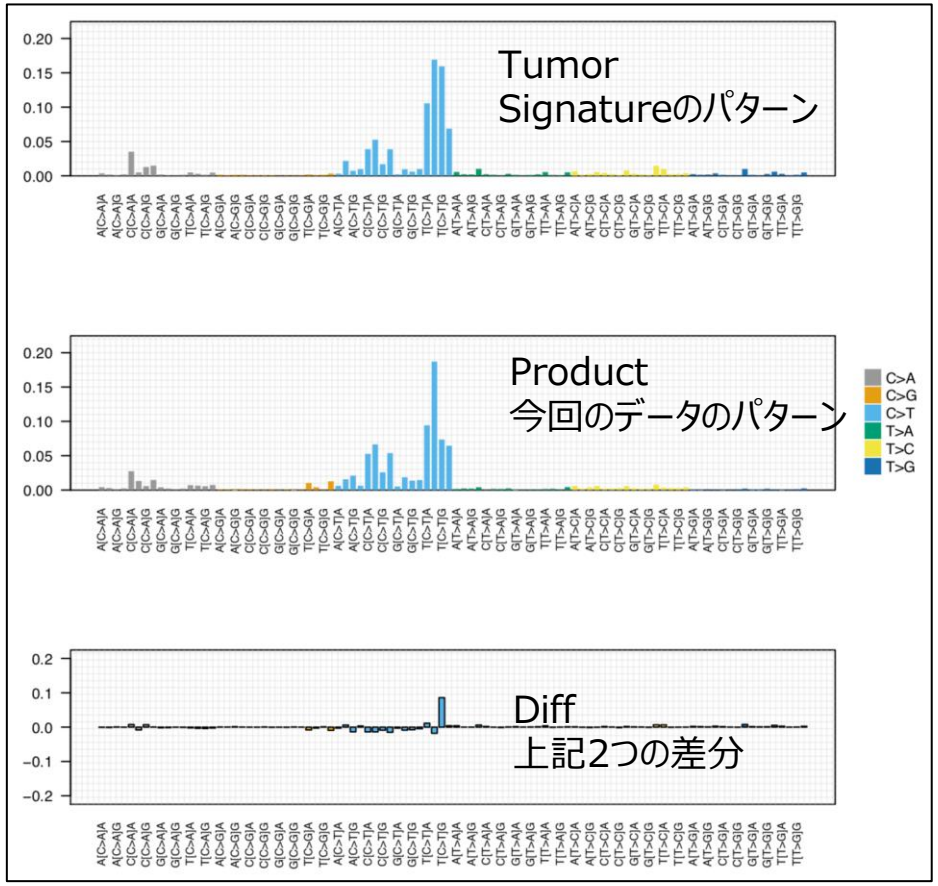
- ✓ このコマンドをそのままRの画面にコピペして実行する
- ✓ Signatureのマトリックスはいくつかあり、今回はnature2013を用いた

Signature解析結果

signature解析の結果を可視化してみます。

- **deconstructSigsの結果（out.pdf)**
  - ✓ 変異とその変異の前後の塩基のパターンが表示されます。

out.pdf



- **Signiture解析の結果を可視化（out.txt)**
  - ✓ データはタブ区切りのテキストファイルです。
  - ✓ 1行目のheaderの部分が一マス分ずれていますので注意！！

out.txt

```
Signature.1A    Signature.1B    Signature.2     Signature.3     Signature.4     Signature.5     Signature.6
                Signature.7     Signature.8     Signature.9     Signature.10    Signature.11    Signature.1Signature.13
                Signature.14    Signature.15    Signature.16    Signature.17    Signature.18    Signature.19
                Signature.20    Signature.21    Signature.R1    Signature.R2    Signature.R3    Signature.USignature.U2
COLO-829        0.0701003147747301              0.12638684342813 0.0748129043387653             0               0
                0               0               0.619335486318396               0               0               0
                0               0               0               0               0               0.0788026746270085
                0               0               0
HCC-1143        0.247635398957036               0               0.101556417412786               0               0
                0               0               0               0               0.0775031321768545              0
                0               0               0.144065333237178               0               0
                0.302870022535305               0               0               0               0               0
                0               0               0               0               0
HCC1187_80      0.249037938500639               0               0               0.139313303442557               0
                0               0               0               0.16206473218697 0.0708909361013918             0
                0               0               0               0               0               0.249575551253486
                0               0               0               0               0               0               0
                0               0               0
```

1. Excelで読み込む
2. A1にセルを挿入し、1行目のみを1列右にずらす
3. Signatureを空文字に置換する
4. 条件付き書式で数値部分にグラデーションを付ける

	1A	1B	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	R1	R2	R3	U1	U2
COLO-829	0.1	0.1	0.1	0	0	0	0	0.6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.1	0	0	0
HCC-1143	0.2	0	0.1	0	0	0	0	0	0	0.1	0	0	0	0.1	0	0	0.3	0	0	0	0	0	0	0	0	0	0
HCC1187_80	0.2	0	0	0.1	0	0	0	0	0.2	0.1	0	0	0	0	0	0	0.2	0	0	0	0	0	0	0	0	0	0

Signature.7は紫外線が原因の可能性が考えられます。
参考：https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3776390/

INDEX

*1章についてはp.102～105の索引を参照ください。
*単語が記載されている全頁を記しているわけではありません。

INDEX

*1章についてはp.102～105の索引を参照ください。

*単語が記載されている全頁を記しているわけではありません。

監修者

本テキストの監修者は下記の通りとなります。

監修者	所属	監修章
井元清哉	東京大学医科学研究所　ヒトゲノム解析センター　健康医療インテリジェンス分野　教授	第4章
加藤護	国立がん研究センター　基盤的臨床開発研究コアセンター　オミックスコア バイオインフォマティクス部門　部門長	第3章、第4章
山口類	愛知県がんセンター研究所　システム解析学分野　分野長	第1章、第2章
松田浩一	東京大学大学院　新領域創成科学研究科 メディカル情報生命専攻　メディカルサイエンス講座　教授	第5章
尾崎遼	筑波大学　医学医療系生命医科学域　バイオインフォマティクス分野 准教授	第1章、第2章、第5章
鈴木健介	イルミナ株式会社	第3章