Linuxで脳画像を扱おう

筑波大学医学医療系精神医学

根本　清貴

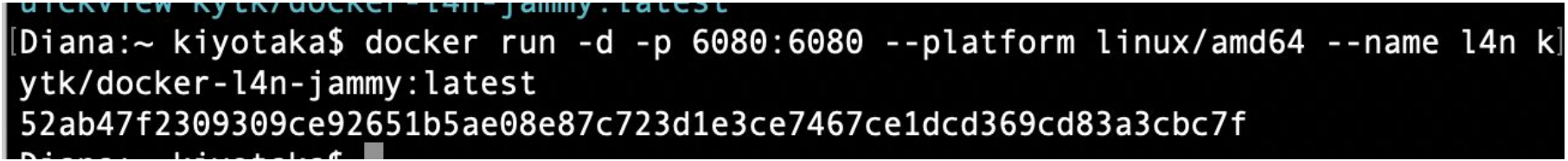
Ver.20241008

コンテナの起動

- Docker Desktop を起動
- ターミナル / Powershell に以下をタイプ
- 改行はせず、全て1行で入力すること

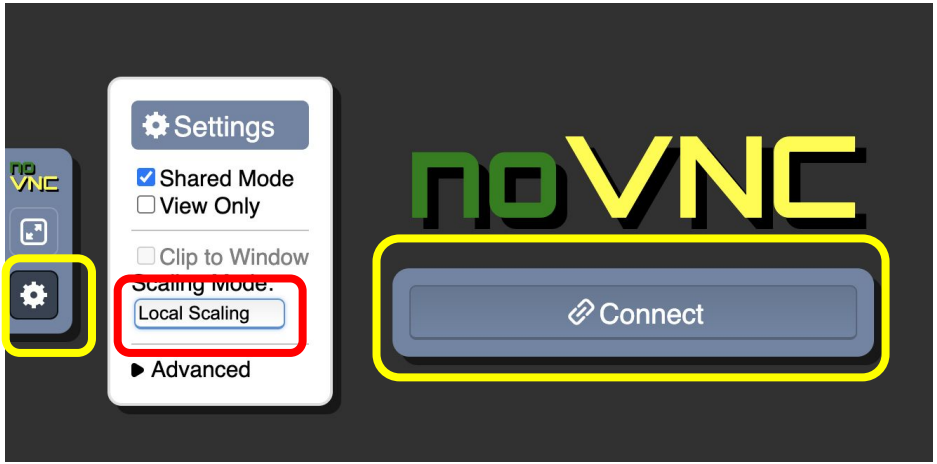
```
docker run -d -p 6080:6080 --privileged --platform
linux/amd64 --name l4n kytk/docker-l4n-jammy:latest
```

- 文字列が表示される（文字列の文字は人によって異なる）

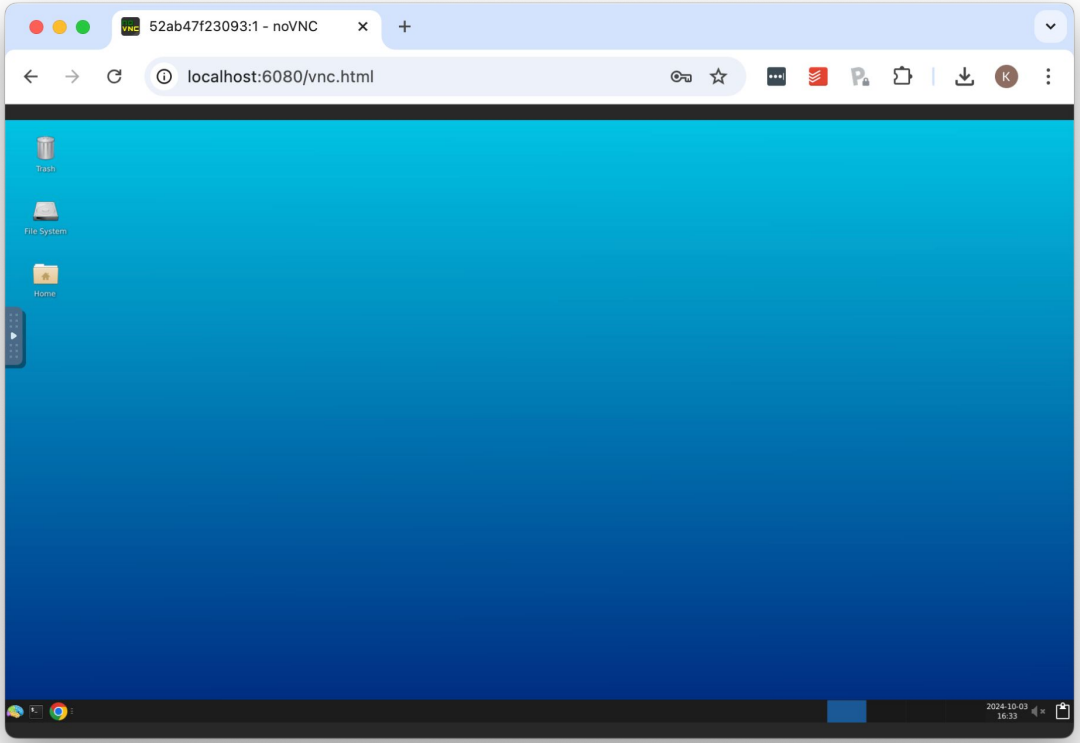


ブラウザからLinuxを起動

- Webブラウザのアドレスに以下を入力
  localhost:6080/vnc.html
- 画面左側の歯車アイコンをクリック、Scaling Modeを**Local Scaling**に変更
- **Connect**をクリック、パスワードは **lin4neuro**

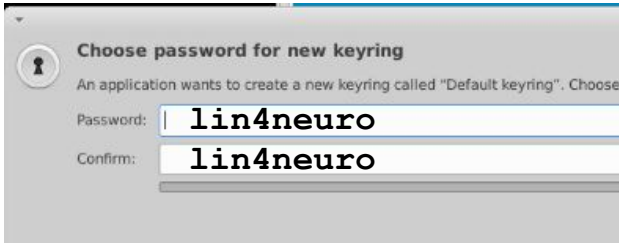


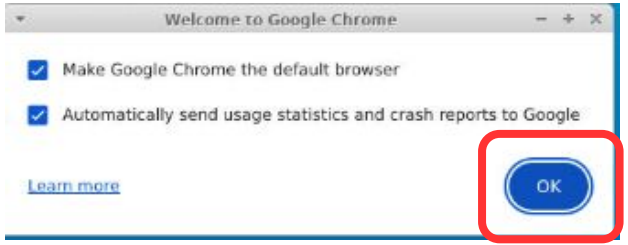
起動したLinuxの画面

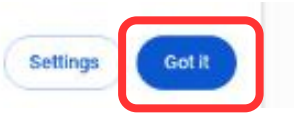


Google Chrome の起動

- Google Chromeを起動（左下のアイコン）
- Choose password for new keyring に "lin4neruo" と2回タイプし、右下の"Continue"をクリック（lin4neuroは表示されない）
- 次の画面で "OK" をクリック
- 次の画面は "Don't Sign in" をクリック
- 最後は "Got it" をクリック







サンプルデータおよびテキストのダウンロード

lin4neuro.net/jtd からダウンロード

- ターミナルから以下をタイプ

```
cd
cd Downloads
unzip shell_practice.zip -d ~/
cd
ls shell_practice
```
（→これで nifti が見えるはず）

本日の内容

- **第1部：UNIX系OSのお作法を知る**
- **第2部：画像解析ソフトのコマンドに触れる**

本講義のルール

- ターミナルでタイプするものは、**青色**（0000cc）で表示
  - 例：`$ fslhd V_ID001.nii.gz`
  - $はタイプする必要はない
- スクリプトに記載する内容は**緑色**（007e00）で表示
- コマンドやスクリプトではあるが、タイプしなくていいものは、**紫色**（9933ff）で表示
- #以降は、解説でありタイプする必要はない
- 「フォルダ」＝「ディレクトリ」
  - Linux/UNIXは、「ディレクトリ」を好む

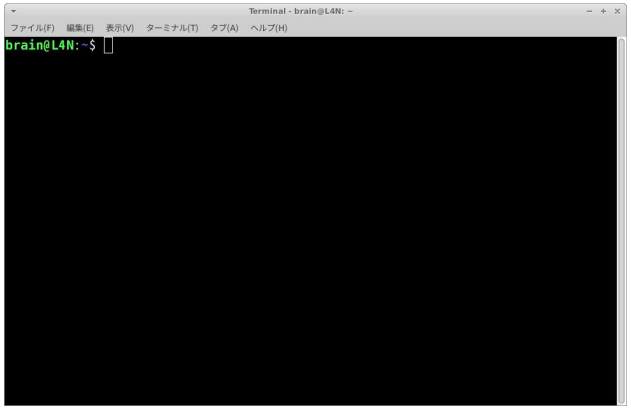
本日の内容

- 第1部：UNIX系OSのお作法を知る

- 第2部：画像解析ソフトのコマンドに触れる

シェルとは？

- アプリケーションのひとつ（ターミナル）
  - コマンドをキーボードから入力し、プログラムを実行する
- プログラミング言語のひとつ
  - コマンドをテキストファイルに羅列
  - 繰り返しや条件分岐
  - いくつもの方言
    - sh, bash, tcsh, zsh…
    - bashがデフォルト



よくあるミスと
便利なショートカット

- よくあるミス
  - スペース忘れ
    - 必要なところでスペースがないとエラーになる
  - タイプミス
    - タイプミスを減らす方法は、Tabキーを上手に使うこと
- 入力する内容の「意味」を考えながらタイプすることが大事
- ターミナルで便利なショートカット
  - Ctrl + A　タイプしたコマンドの最初に移動する
  - Ctrl + E　タイプしたコマンドの最後に移動する

UNIX系OSのお作法

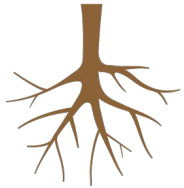
- ヒエラルキーがしっかりしている
- パスの指定には「絶対パス」と「相対パス」がある
- コマンドは英語の「命令文」に通じる
  - 指定するものには「引数」と「オプション」がある
  - 引数にはワイルドカードが使えることが多い
  - オプションはハイフンで指定する
- ファイル名にスペースと日本語は使わない
- 目的を達成するために「パイプ」を使う
  - 複数の工程はパイプでつなげる

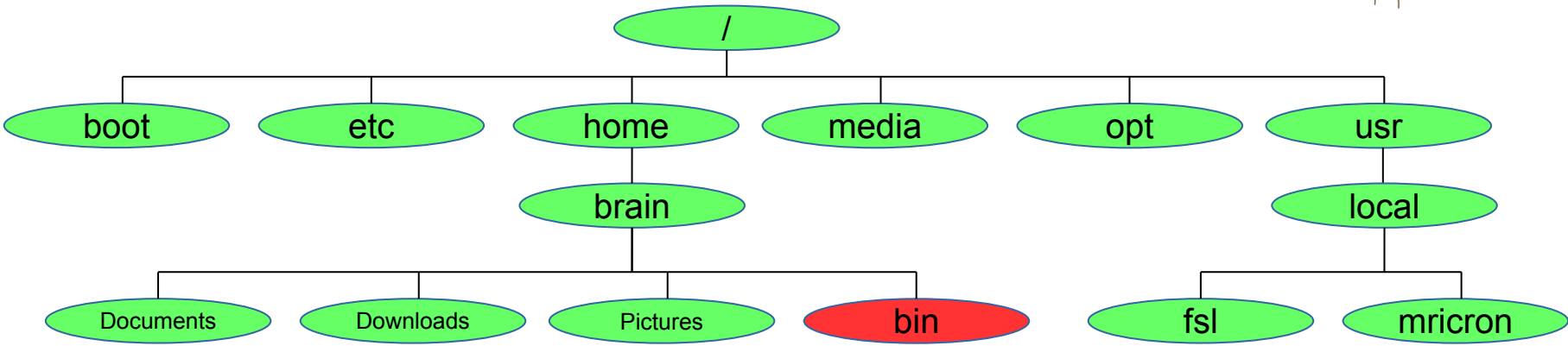
UNIX系OSのお作法

- ヒエラルキーがしっかりしている
- パスの指定には「絶対パス」と「相対パス」がある
- コマンドは英語の「命令文」に通じる
  - 指定するものには「引数」と「オプション」がある
  - 引数にはワイルドカードが使えることが多い
  - オプションはハイフンで指定する
- ファイル名にスペースと日本語は使わない
- 目的を達成するために「パイプ」を使う
  - 複数の工程はパイプでつなげる

システムのヒエラルキーとパス

- UNIX系OSでは、頂点を root (/) といい、その下に様々なディレクトリが位置する（ひっくり返した木）
- ディレクトリまでの経路をパス (path) という

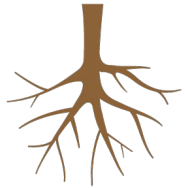


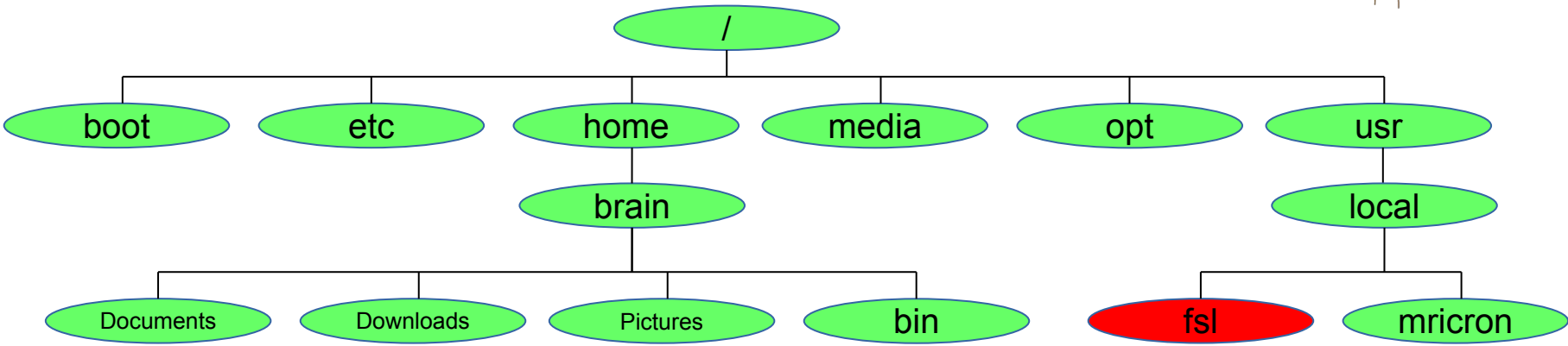


ここのパスは
`/home/brain/bin`

システムのヒエラルキーとパス

- UNIX系OSでは、頂点を root (/) といい、その下に様々なディレクトリが位置する（ひっくり返した木）
- ディレクトリまでの経路をパス (path) という





ここのパスは
`/usr/local/fsl`

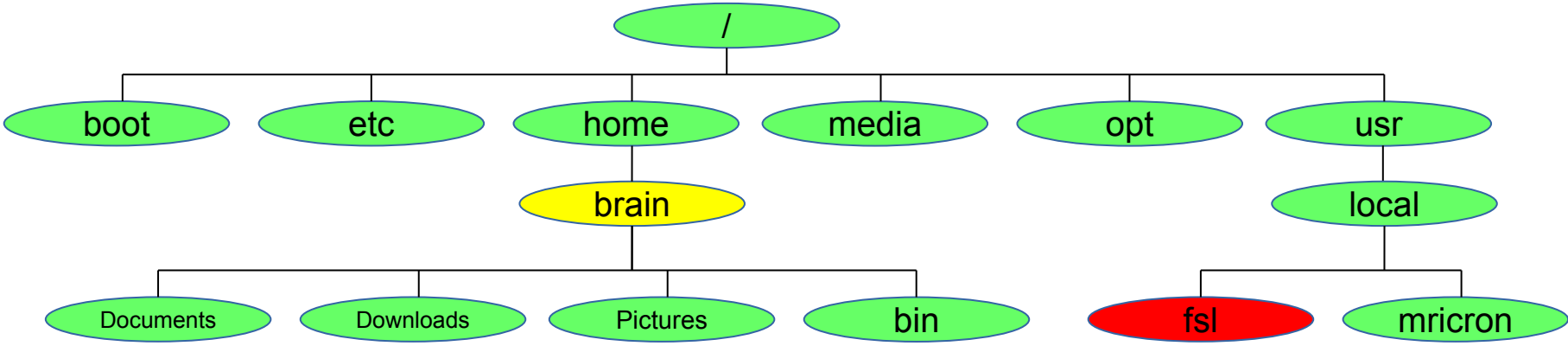
UNIX系OSのお作法

- ヒエラルキーがしっかりしている
- パスの指定には「絶対パス」と「相対パス」がある
- コマンドは英語の「命令文」に通じる
  - 指定するものには「引数」と「オプション」がある
  - 引数にはワイルドカードが使えることが多い
  - オプションはハイフンで指定する
- ファイル名にスペースと日本語は使わない
- 目的を達成するために「パイプ」を使う
  - 複数の工程はパイプでつなげる

絶対パスと相対パス

- 絶対パス：ルート root (/) から出発した経路（パス）
  - フルパスとも言う full path
- 相対パス：あるディレクトリからの経路（パス）
- パスの表示を簡略化するために、3つの記号がある
  - チルダ　~
    - ホームディレクトリを意味
    - Linux: `/home/taro`
    - macOS: `/Users/taro`
  - シングルドット　.
    - 現在のディレクトリ（カレントディレクトリ）
  - ダブルドット　..
    - 親ディレクトリ（自分の上のディレクトリ）

絶対パスと相対パス



- 絶対パス: /usr/local/fsl
- ~/brain からの相対パス： ../../usr/local/fsl
  - 上(home)の上(root)にあがって、usrの下のlocalの下のfsl

UNIX系OSのお作法

- ヒエラルキーがしっかりしている
- パスの指定には「絶対パス」と「相対パス」がある
- コマンドは英語の「命令文」に通じる
  - 指定するものには「引数」と「オプション」がある
  - 引数にはワイルドカードが使えることが多い
  - オプションはハイフンで指定する
- ファイル名にスペースと日本語は使わない
- 目的を達成するために「パイプ」を使う
  - 複数の工程はパイプでつなげる

基本コマンド

- シェルは言語である → 基本的な単語を知らないと命令できない
- コマンドは、基本的に「動詞」←命令だから
- 各コマンドは、ユーザーがキーをたくさんタイプしなくていいようにコンパクトになっている

- 本質的な理解：「コマンドライン」＝「命令」の「羅列」
- 英語で命令文を言うことと同じ
  - 「AをBにコピーしなさい！」「CをDに移動しなさい！」「EにあるFを見つけなさい！」

英語の基本文法でいえば

- 第3文型、第4文型で理解
  - 命令文なので、主語は省略（You = PC）
- 目的語に相当するものを、シェル言語では「引数（ひきすう）」という
- 第3文型：ＳＶＯ → 引数は1つ
  - **cd, mkdir, rmdir** など
- 第4文型：ＳＶＯＯ → 引数は2つ
  - **cp, mv** など

引数の数

- 効率をあげるため、引数は同じ種類のものならば複数指定できることがある
- 例: ディレクトリを作成するコマンド、**mkdir** は基本的に引数は1つだが、複数ディレクトリを作成したいときには複数指定もできる
  - 下の2つは全く同じ意味
  - `mkdir dirA; mkdir dirB; mkdir dirC`
  - `mkdir dirA dirB dirC`

    （コマンドラインにおいて、;は改行と同じ意味であり、複数行の内容を一行で表示できる）

オプション

- オプションは、その名が示すように、コマンドに付随してくる機能のこと
- 例：コピーコマンド cp に、オプションの -r をつけると、コピーしたいディレクトリの中身をすべてコピーしてくれる
- オプションの場所は、原則、コマンドの直後
- オプションは、基本的にハイフン (-) をつけることが多い
  - ハイフン1つの場合は略語、ハイフン2つの場合はオプションのフルネームの場合が多い
- かつ、オプションはつなげることができる
  - `cp -r -v dirA /path/dirB`
  - `cp -rv dirA /path/dirB`

必須コマンド13

コマンド	由来	役割
`ls`	**l**i**s**t	ファイル一覧を表示
`cd`	**c**hange **d**irectory	ディレクトリを移動
`pwd`	**p**rint **w**orking **d**irectory	現在のディレクトリを表示
`cp`	**c**o**p**y	コピー
`mv`	**m**o**v**e	移動/リネーム
`mkdir`	**m**a**k**e **dir**ectory	ディレクトリの作成
`rm`	**r**e**m**ove	削除
`rmdir`	**r**e**m**ove **dir**ectory	ディレクトリの削除
`chmod`	**ch**ange **mod**e	ファイル・ディレクトリの権限を設定
`cat`	con**cat**enate	テキストファイルの結合
`less`		テキストファイルの内容を表示
`history`	`history`	コマンドの履歴を表示
`wc`	**w**ord **c**ount	単語のカウント

ls

- list の略
  - 「ファイル/ディレクトリを表示しなさい」
- 引数は原則ひとつ
  - `ls dirA` で dirA ディレクトリ内のファイル/ディレクトリー覧が表示される
- 引数を指定しないと、カレントディレクトリが表示される
  - `ls .` と同じ意味

よく使われる `ls` のオプション

- **`-a (--all)`**
  - 隠しファイルも表示させる
- **`-l (longのl)`**
  - ファイル名/ディレクトリ名だけでなく、様々な情報を表示させる

- 【練習】以下のコマンドをタイプしてください

```
ls /usr/local/fsl
ls -al /usr/local/fsl
```

ls -al

```
nemoto@ip-172-31-21-143:~$ ls /usr/local/fsl/
LICENCE.FSL       config    lib           python   tcl
LICENSE.txt       data      libexec       qml      translations
bin               doc       man           sbin     var
cmake             envs      mkspecs       share    x86_64-conda-linux-gnu
compiler_compat   etc       phrasebooks   shell    x86_64-conda_cos6-linux-gnu
conda-meta        fonts     pkgs          src
condabin          include   plugins       ssl
nemoto@ip-172-31-21-143:~$
```

```
nemoto@ip-172-31-21-143:~$ ls -al /usr/local/fsl/
合計 484
drwxr-xr-x  33 root root   4096 10月  6 05:35 .
drwxr-xr-x  22 root root   4096 10月  6 05:28 ..
-rw-r--r--   1 root root   1863 10月  6 05:28 .condarc
-rw-rw-r--   2 root root    113  9月 10 07:00 .crates.toml
-rw-rw-r--   2 root root    481  9月 10 07:00 .crates2.json
-rwxrwxr-x   2 root root   3019  9月 25 15:32 LICENCE.FSL
-rw-r--r--   1 root root   2473 10月  6 05:28 LICENSE.txt
drwxr-xr-x   2 root root  49152 10月  6 05:35 bin
drwxr-xr-x   2 root root   4096 10月  6 05:33 cmake
drwxr-xr-x   2 root root   4096 10月  6 05:35 compiler_compat
drwxr-xr-x   2 root root  40960 10月  6 05:35 conda-meta
```

パーミッション　　所有者　　ファイルサイズ　　ファイル/ディレクトリ名

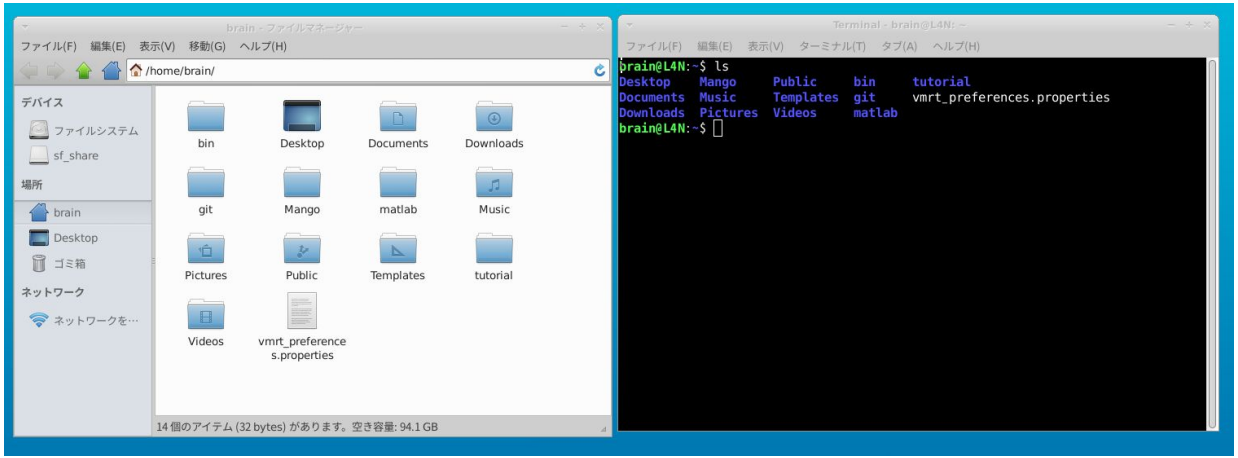
cd

- **c**hange **d**irectory の略
  - 「ディレクトリを移動しなさい」
- 引数はひとつ
  - `cd /path/dirA` は `/path/dirA` に移動する
- 引数を指定しないと、ホームディレクトリに戻る
  - 引数を指定しない `cd` は `cd ~` と同じ意味

- 一番使うコマンドは、`cd` と `ls`
- `cd` したら `ls` する習慣をつける！

cd と ls の組み合わせ

- cd と ls の組み合わせは、ファイルマネージャー（エクスプローラー/Finder）でフォルダを移動した時にファイル一覧が見えることと同じ



練習: cd

- `~/shell_practice` に移動してください

  `__ ~/shell_practice`

- `shell_practice` の中に何があるか確認してください

  `__`

- 上記で見つけたディレクトリの中に何があるか確認してください

  `__ _____`

練習: cd 回答

- `~/shell_practice` に移動してください

  ```
  cd ~/shell_practice
  ```

- `shell_practice` の中に何があるか確認してください

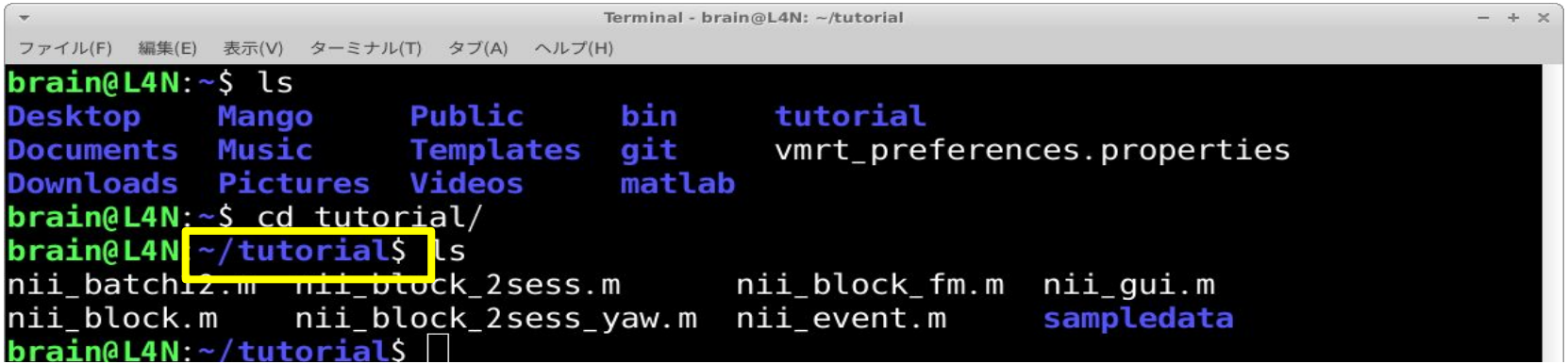
  ```
  ls
  ```

- 上記で見つけたディレクトリの中に何があるか確認してください

  ```
  ls nifti
  ```

pwd

- **p**rint **w**orking **d**irectory の略
- 「ワーキングディレクトリを表示しなさい」
- 自分の今の居場所（＝ワーキングディレクトリ）が表示される
  - Linuxでは、デフォルトで、ターミナルにワーキングディレクトリのパスが表示される



現在のディレクトリは ~/tutorial

cp

- **c**o**p**y の略
- 引数を2つ指定する
  - `cp fileA fileB` で「fileA を fileB として複製しなさい (copy fileA as fileB)」
  - `cp fileA /path/dirC` で「fileA を /path/dirC *にコピーしなさい* (copy fileA to /path/dirC)」

cp でよく使うオプション

- `-r (--recursive)`
  - ディレクトリを再帰的に（下層のディレクトリまで）コピーする
- `-i (--interactive)`
  - コピー先に同じファイル名があったら上書きするか確認する
- `-v (--verbose)`
  - コピー作業中に詳細を表示する

mv

- move の略
- 引数を2つ指定する
- 指定の仕方で結果が若干変わるので注意
  - `mv fileA fileB` で「fileA を fileB として移動（＝名前を変更）しなさい (move fileA as fileB)」
  - `mv fileA /path/dirC` で「fileA を /path/dirC *に移動しなさい* (move fileA to /path/dirC)」

mv でよく使うオプション

- **`-i (--interactive)`**
  - 移動先に同じファイル名があったら上書きするか確認する
- **`-v (--verbose)`**
  - 移動作業中に詳細を表示する

mkdir

- **m**a**k**e **dir**ectory の略
- 作成したいディレクトリ名を引数として指定する
  - `mkdir dirA` で「dirA を作成しなさい」
- よく使うオプション
  - **-p** 親ディレクトリを同時に作成する
  - ホームディレクトリの下に**practice/scripts/fsl** というディレクトリを作成したいが、**practice** も **scripts** もまだ作成されていない場合、
  - `$ mkdir -p ~/practice/scripts/fsl` でディレクトリが一気に作成される

練習: cp mv mkdir

現在、~/shell_practice にいます

- backup というディレクトリを作成してください

```
_____ backup
```

- nifti ディレクトリを backup にコピーしてください

```
__ __ nifti backup
```

- backup に移動してから、その中を確認し、nifti ディレクトリを nifti_original に名前を変えてください

```
__ backup

__

__ nifti nifti_original
```

練習: cp mv mkdir 回答

現在、~/shell_practice にいます

- backup というディレクトリを作成してください

```
mkdir backup
```

- nifti ディレクトリを backup にコピーしてください

```
cp -r nifti backup
```

- backup に移動してから、その中を確認し、nifti ディレクトリを nifti_original に名前を変えてください

```
cd backup
ls
mv nifti nifti_original
```

rm

- **r**e**m**ove の略
- 削除したいファイルやディレクトリを引数として指定する
  - `rm fileA` で「fileA を削除しなさい」
- よく使うオプション
  - `-i` 削除前に確認する
    - rmで削除されたファイルは、ゴミ箱などに行かず、そのまま消されるために要注意！
    - 「rm: ファイル 'fileA' を削除しますか？」に対し、y とタイプすれば削除される
  - `-r` 再帰的に削除する
    - `rm -r dirA` で dirAディレクトリ下のすべてのファイル /ディレクトリが削除される

rm -rf は避ける

- rm のオプションに -f がある
- これは「強制的に削除」を意味する
- rm -rf dirA を指定すると、強制的に削除される
- このため、rm -rf は推奨されていない
  - 自信があるときのみ

rmdir

- **r**e**m**ove **dir**ecotry の略
- 削除したいディレクトリを引数として指定する
  - `rmdir dirA` で「dirA を削除しなさい」
- **rmdir** はディレクトリの中にファイルがあるとそのディレクトリは削除しない
  - ディレクトリが空の時だけ削除されるので安心

chmod

- **ch**ange **mod**e の略
- ファイルやディレクトリの権限（パーミッション）を指定する
- 引数は2つ
- `chmod 権限 fileA` で「fileA の権限を指定のものに設定しなさい」

UNIX系OSにおけるファイルの権限

```
-rwxrwx--- 1 brain brain fileA
drwxr-xr-x 1 brain brain dirA
```

- UNIX系のOSでは、個々のファイルやディレクトリに対して、「読み取り権限」「書き込み権限」「実行権限」を「所有者」「グループ」「他の人々」の3つに対して設定することができる。
- 属性は、`ls -al`で確認できる。
- 最初のひとつがディレクトリか否か、次の3つが「所有者」、次の3つが「グループ」、最後の3つが「その他」
- `r` 読み取り　`w` 書き込み　`x` 実行
  - `-rwxrwx---`　は
    - 所有者とグループ：読み書き実行可、それ以外：すべて不可のファイル
  - `drwxr-xr-x`　は
    - 所有者：すべて可、グループとそれ以外：読み取りと実行可のディレクトリ

chmod で権限を変更する

- chmod での権限の設定方法は2通りある
- `chmod a+x fileA`
  - all (所有者、グループ、その他)に実行権限 **x** を追加
- `chmod 755 fileA`
  - r を4, w を2, x を1として、その合計を記載
  - **rwx**: 4+2+1=7; **r-x**: 4+1=5
  - **chmod 755** は、「所有者はフル権限、グループとその他は読み取りと実行のみ可能」という意味

cat

- con**cat**enate の略
- ファイルを連結して表示する
- 引数は1つ以上
- `cat fileA fileB` で「fileA と fileB を連結して表示しなさい」
  - 引数1つだと、指定したファイルを表示
  - 引数2つ以上だと、連続して表示
    - ファイルを合体したいときに便利

練習: cat

- ~/shell_practice/nifti に移動してください

```
cd ~/shell_practice/nifti
```

- この中に reset.sh があります。これを cat で表示してください

```
cat reset.sh
```

- 同様に、subj1_dwi.bval を cat で表示してください

```
cat subj1_dwi.bval
```

- reset.sh と subj1_dwi.bval を cat で表示してください

```
cat reset.sh subj1_dwi.bval
```

less

- ファイルビューワー
- ファイルの内容を表示する
- 行数が多いファイルの場合、1画面に入る内容だけ表示する
- スペースで画面送り
- 終了したいときは、Qをタイプ
  - “Quit”のQで覚える
- /の後にキーワードを入れると、キーワード検索もできる
  - **n**（小文字のn）で前方検索、N（大文字のN）で後方検索

wc

- word count の略 （トイレではない）
- ファイルの行数、単語数、文字数を表示する
- wc fileA で「fileA の行数、単語数、文字数を表示しなさい」
- よく使うオプション
  - -l (--lines)
    - 行数のみ表示する
  - -w (--words)
    - 単語数のみ表示する
- ls と wc を組み合わせると、ファイル数のカウントに用いることができる

練習: wc

- （移動していない場合は）`~/shell_practice/nifti` に移動してください

  ```
  cd ~/shell_practice/nifti
  ```

- `subj1_dwi.bval` を cat で表示してください

  ```
  cat subj1_dwi.bval
  ```

- `subj1_dwi.bval` の中に何個の単語があるか調べてください

  ```
  __ __ subj1_dwi.bval
  ```

- `subj1_dwi.bval` は何行のファイルか調べてください

  ```
  __ __ subj1_dwi.bval
  ```

練習: wc 回答

- （移動していない場合は）`~/shell_practice/nifti` に移動してください

  ```
  cd ~/shell_practice/nifti
  ```

- `subj1_dwi.bval` を cat で表示してください

  ```
  cat subj1_dwi.bval
  ```

- `subj1_dwi.bval` の中に何個の単語があるか調べてください

  ```
  wc -w subj1_dwi.bval
  ```

- `subj1_dwi.bval` は何行のファイルか調べてください

  ```
  wc -l subj1_dwi.bval
  ```

ワイルドカード

- 引数を指定する際、複数ファイルを表現したいときに便利な記号
- *(アスタリスク) と ? の2つ
  - img.nii, img1.nii, img01.nii があるとする
  - * 長さ0文字以上の任意の文字列にマッチするパターン
    - img*.nii
      - img.nii, img1.nii, img01.nii がすべて該当
  - ? 任意の1文字にマッチするパターン
    - img?.nii
      - img1.nii のみ該当

練習: ワイルドカード

- （移動していない場合は）~/shell_practice/nifti に移動してください

  ```
  cd ~/shell_practice/nifti
  ```

- subj1 からはじまるファイルだけ ls で表示してください

  ```
  ls ______
  ```

- ファイル名に ecc が入っているファイルを ls で表示してください

  ```
  ls _____
  ```

- ファイル名が .nii.gz で終わるファイルを ls で表示してください

  ```
  ls ________
  ```

練習: ワイルドカード 回答

- （移動していない場合は）`~/shell_practice/nifti` に移動してください

  ```
  cd ~/shell_practice/nifti
  ```

- `subj1` からはじまるファイルだけ `ls` で表示してください

  ```
  ls subj1*
  ```

- ファイル名に `ecc` が入っているファイルを `ls` で表示してください

  ```
  ls *ecc*
  ```

- ファイル名が `.nii.gz` で終わるファイルを `ls` で表示してください

  ```
  ls *.nii.gz
  ```

解凍コマンド 3つ

コマンド	意味
`unzip`	zipファイルを解凍
`gunzip`	gzファイルを解凍
`tar`	tarファイル、tar.gzファイル, tar.bz2ファイルなどを解凍 tarのあとに何をつけるかで、tar, tar.gz, tar.bz2ファイルなどを解凍できる x: e**x**tract; v: **v**erbose; f: **f**ile

- `$ unzip file1.zip`
- `$ gunzip file2.nii.gz`
- `$ tar xvf file3.tar`
- `$ tar xvzf file4.tar.gz`
- `$ tar xvjf file5.tar.bz2`

UNIX系OSのお作法

- ヒエラルキーがしっかりしている
- パスの指定には「絶対パス」と「相対パス」がある
- コマンドは英語の「命令文」に通じる
  - 指定するものには「引数」と「オプション」がある
  - 引数にはワイルドカードが使えることが多い
  - オプションはハイフンで指定する
- ファイル名にスペースと日本語は使わない
- 目的を達成するために「パイプ」を使う
  - 複数の工程はパイプでつなげる

シェルにとっての空白の意味

- シェルにとって、空白は「コマンド, オプション, 引数の区切り」という意味
  - 例: `cp -v fileA fileB`
- ファイル名に空白が入っていると、シェルは誤解する
  - `‘Sub 01.nii’` というファイルを dirA に移動しようとして、`mv Sub 01.nii dirA` とタイプしたとすると、「Sub と 01.nii を dirA に移動しなさい」とシェルは理解するため、エラーとなる（Sub も 01.nii も存在しないため）
- ファイル名に空白は使わない習慣をつける！
  - 空白を入れたかったらアンダースコア（ _ ）がおすすめ

日本語や全角スペースも使わない

- 日本語は変換する手間がかかる
- 全角スペースはエラーの元になる
- コマンドは「英語」の発想でいくことが大事
- 以上のことより、コマンドラインで仕事をするときには、「ファイル名に空白や日本語は使わない」ことを習慣にすることが肝要

UNIX系OSのお作法

- ヒエラルキーがしっかりしている
- パスの指定には「絶対パス」と「相対パス」がある
- コマンドは英語の「命令文」に通じる
  - 指定するものには「引数」と「オプション」がある
  - 引数にはワイルドカードが使えることが多い
  - オプションはハイフンで指定する
- ファイル名にスペースと日本語は使わない
- 目的を達成するために「パイプ」を使う
  - 複数の工程はパイプでつなげる

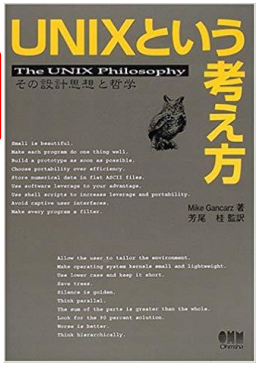
パイプ |

- パイプは、前のコマンドの結果を次に渡す役割
- UNIX系OSで使いこなせると非常に便利
- lsと単語カウントのコマンド wc -w をパイプで組み合わせると、ファイル数を表示できる
  - フォルダの中のファイル数を確認したいときに便利

```
ls              #ファイル・ディレクトリの表示
ls | wc -w      #ファイル・ディレクトリの数の表示
```

UNIXの思想

- 小さいものは美しい
- **各プログラムが一つのことをうまくやるようにせよ**
- できる限り早く原型（プロトタイプ）を作れ
- 効率よりも移植しやすさを選べ
- 単純なテキストファイルにデータを格納せよ
- ソフトウェアを梃子(てこ)として利用せよ
- 効率と移植性を高めるためにシェルスクリプトを利用せよ
- 拘束的なユーザーインターフェースは作るな
- 全てのプログラムはフィルタとして振る舞うようにせよ



UNIXという考え方—その設計思想と哲学

パイプ | を使いこなすために

- 「そのコマンドはどういう出力になるか？」を意識
- 出力に対して次のコマンドを実行する
- `cat fileA | wc -l` は、fileA の内容を表示したうえで、その行数をカウントする

本日の内容

- 第1部：UNIX系OSのお作法を知る

- 第2部：画像解析ソフトのコマンドに触れる

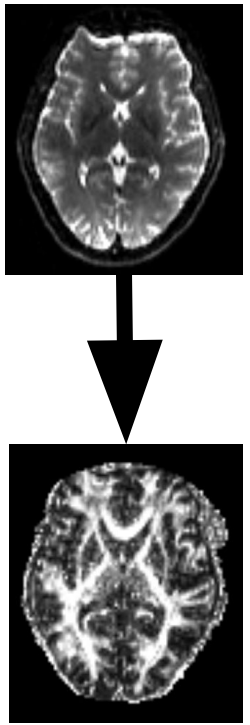
画像解析とコマンドラインの親和性

- 画像解析では、同じ作業を何度も繰り返す
  - 例: 脳MRIから脳だけを抽出することを1000人に行う
- 画像1つに対する作業はGUIでいいが、1000回の作業をGUIで行うのは苦行
  - コマンドラインで前もって準備しておけば、作業が一気にできる

- ここでは、脳画像解析ソフト FSL の中に入っているいくつかのプログラムを紹介する
  - FSLのプログラムは、画像は .nii.gz 形式もしくは .nii 形式を扱うことができる
  - 利便性のため、ファイル名から 拡張子を取り除いて指定することができる
  - 例: `fslinfo subj1.nii.gz` と `fslinfo subj1` は同じ意味

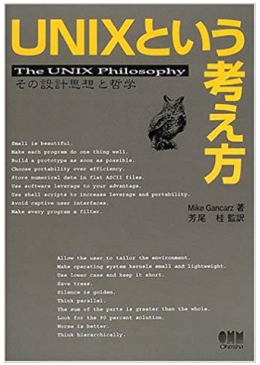
拡散MRI画像からFA画像を生成する

- 渦電流補正
  - `eddy_correct`
    (今日はとりあげない。実施済)
- B0画像の取り出し
  - `fslroi`
- 脳マスクの作成
  - `bet`
- FA画像の生成
  - `dtifit`



UNIXの思想

- 小さいものは美しい
- **各プログラムが一つのことをうまくやるようにせよ**
- できる限り早く原型（プロトタイプ）を作れ
- 効率よりも移植しやすさを選べ
- 単純なテキストファイルにデータを格納せよ
- **ソフトウェアを梃子（てこ）として利用せよ**
- 効率と移植性を高めるためにシェルスクリプトを利用せよ
- 拘束的なユーザーインターフェースは作るな
- 全てのプログラムはフィルタとして振る舞うようにせよ



UNIXという考え方―その設計思想と哲学

fsleyes で拡散MRIを確認

```
cd ~/shell_practice/nifti/
fsleyes subj1_dwi_ecc &
```

2) 画像はBrightnessとContrastで調整できる

1) Volumeを変化させることで、様々なb1000画像を見ることができる

& の意味

- コマンドの最後に & をつけると、そのコマンドは、「バックグラウンド」で実行される
- ターミナルでそのまま他の作業ができる
- もし、先の `fsleyes` で & をつけないと、ターミナルは fsleyes が起動している間は何も受け付けなくなる

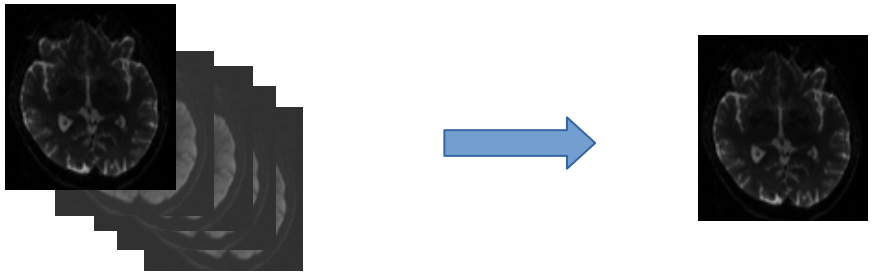
拡散MRI画像からFA画像を生成する

- B0画像の取り出し
  - `fslroi`
- 脳マスクの作成
  - `bet`
- FA画像の生成
  - `dtifit`

fslroi

- 画像から関心領域を取り出すプログラム
- 4次元画像から任意の3次元画像を取り出すこともできる
- 今は、渦電流補正が終わった4次元画像、`subj1_dwi_ecc.nii.gz` ファイルから、最初のb0画像だけ取り出し、`subj1_b0.nii.gz` として出力する
- まずはヘルプを確認する

```
fslroi -h
```



fslroi -h

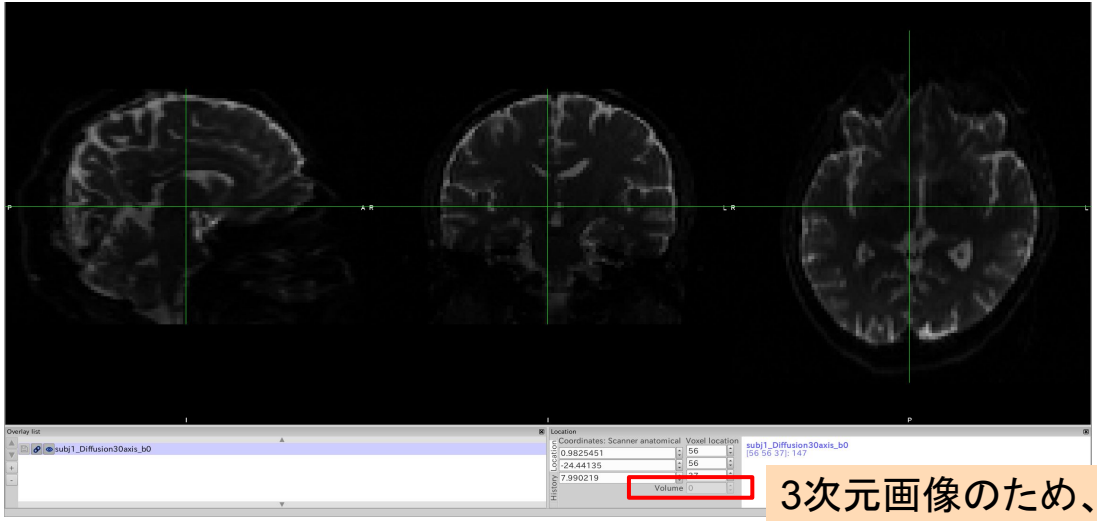
```
Usage: fslroi <input> <output> <xmin> <xsize> <ymin> <ysize> <zmin>
<zsize>

       fslroi <input> <output> <tmin> <tsize>

       fslroi <input> <output> <xmin> <xsize> <ymin> <ysize> <zmin>
<zsize> <tmin> <tsize>

Note: indexing (in both time and space) starts with 0 not 1! Inputting
-1 for a size will set it to the full image extent for that dimension.
```

- 今は、入力画像から、b0画像＝一番最初の画像だけを取り出したい
  - この場合、<tmin> が 0 となり、<tsize> が 1 となる
- 出力画像は、b0画像なので、**subj1_b0** とする

fslroi で B0画像のみ取り出す

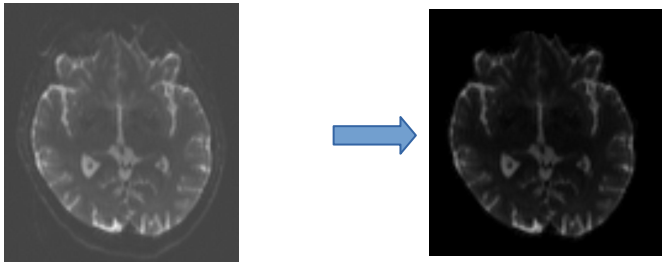
```
fslroi subj1_dwi_ecc subj1_b0 0 1
fsleyes subj1_b0 &
```

- 4次元画像ではなく、3次元画像になっている



3次元画像のため、Volumeは選べなくなっている

bet



- brain extraction tool
- 頭蓋骨除去によく使われるプログラム
- 脳領域のマスク画像も作成できる
- FA画像の計算のために、計算領域を脳にしぼることが必要なため、脳のマスクを作成する
- 入力画像には `subj1_b0` を使用し、出力画像のファイル名は `subj1_brain` としたい（マスク画像は自動で `subj1_brain_mask` となる）

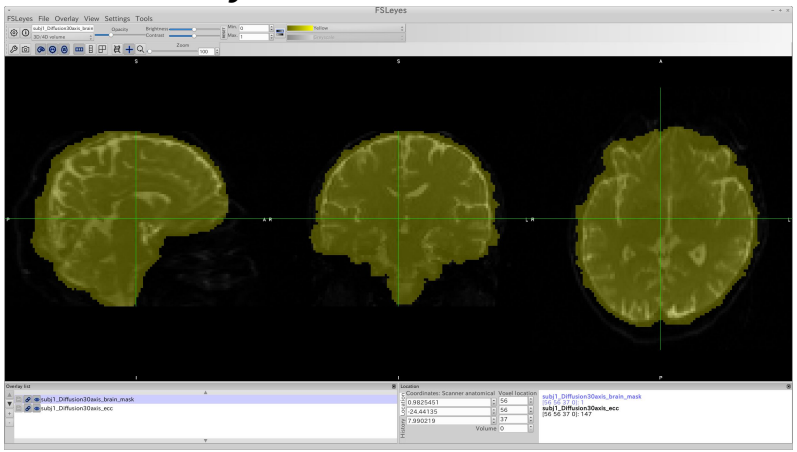
  `bet -h` で使い方を確認

bet -h

```
Usage:    bet <input> <output> [options] #input と outputが必須

Main bet2 options:

-o      generate brain surface outline overlaid onto original image

-m      generate binary brain mask

-s      generate approximate skull image

-n      don't generate segmented brain image output

-f <f>  fractional intensity threshold (0->1); default=0.5; smaller values

        give larger brain outline estimates #defaultではだめなことが多い

-g <g>  vertical gradient in fractional intensity threshold (-1→1);

        default=0; positive values give larger brain outline at bottom,

        smaller at top

-r <r>  head radius (mm not voxels); initial surface sphere is set to half of this

-c <x y z>  centre-of-gravity (voxels not mm) of initial mesh surface.

-t      apply thresholding to segmented brain image and mask

-e      generates brain surface as mesh in .vtk format
```

bet でマスク作成

```
bet subj1_b0 subj1_brain -f 0.3 -m
ls *brain*
  subj1_brain.nii.gz  subj1_brain_mask.nii.gz
fsleyes subj1_b0 subj1_brain_mask -a 40 -cm yellow &
  #-a 40 透明度40% -cm yellow カラーマップを黄色
```



拡散MRI画像からFA画像を生成する

- B0画像の取り出し
  - `fslroi`
- 脳マスクの作成
  - `bet`
- FA画像の生成
  - **`dtifit`**

dtifit

- 拡散MRI画像から、FA画像やMD画像など、拡散テンソル画像を生成するプログラム

  dtifit -h で使い方を確認

```
Usage:
dtifit -k <filename>
 dtifit --verbose
Compulsory arguments (You MUST set one or more of):
-k,--data   dti data file
-o,--out    Output basename
-m,--mask   Bet binary mask file
-r,--bvecs  b vectors file
-b,--bvals  b values file
```

dtifit の引数

```
-k,--data    subj1_dwi_ecc
-o,--out     subj1  ←ここは任意に決められる
-m,--mask    subj1_brain_mask
-r,--bvecs   subj1_dwi.bvec
-b,--bvals   subj1_dwi.bval
```

- `--out` で指定する値で出力ファイルの名前が決まる
  - `--out=taro` とすると、出力ファイルのファイル名は `taro_FA`, `taro_MD`, `taro_V1` などとなる

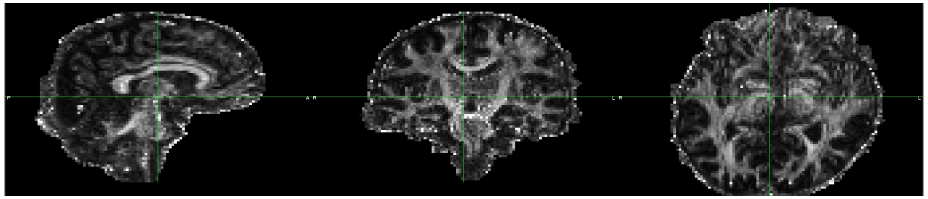
コマンドを複数行で入力

- コマンドは、￥（ \ で表されることもある）を入れると改行しても同じコマンドの続きとみなされる
- dtifit のようなプログラムのように引数が多いプログラムは、\ を上手に使って表現すると理解しやすい
  - 実際に入力する際は、1行でもよい（その場合、\ は入力しなくてよい）

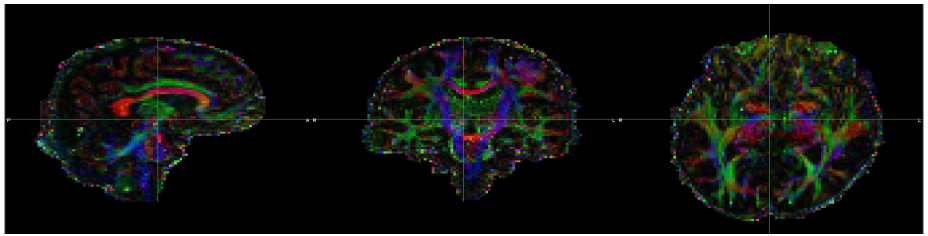
```
dtifit --data=subj1_dwi_ecc \
         --out=subj1 \
         --mask=subj1_brain_mask \
         --bvecs=subj1_dwi.bvec \
         --bvals=subj1_dwi.bval
ls
```

FA画像の確認

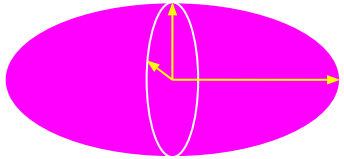
```
fsleyes subj1_FA.nii.gz &
```



```
fsleyes subj1_FA subj1_V1 \
        -ot rgbvector \
        -mo subj1_FA &
```



V1画像



- V1画像は、拡散MRIから作成されたテンソルのうち、スカラーがもっとも大きいベクトルを画像化したもの
- ベクトルが3次元であるため、x軸方向、y軸方向、z軸方向の情報をもった4次元画像として作成されている
- `fsleyes` で見る時、`3-direction vector image (RGB)` か `3-direction vector image (Line)` で表示する
- RGB: DTIベクトルは x軸方向に Red, y軸方向に Green, z軸方向に Blue で表示される。紫のような色は、RGBの混合→ベクトルの向きがx, y, z の混合で表示されているということ
- Line: DTIベクトルは、小さな線の集合として表示される。x, y, z軸方向にそれぞれ R, G, B の色が割り当てられている
- RGB モードの時は、信号強度をFA画像などで修飾することができる。つまり、FAが大きい領域は色合いが強くすることができる

スクリプト化

- 毎回手でうつのは大変
- このような時に、「スクリプト」が役立つ
- スクリプトは、シンプルにやることを書き出すだけ

- スクリプトを準備してあるので見てみる

```
cat calc_fa_subj2.sh
```

calc_fa_subj2.sh

```
#!/bin/bash

# fslroiでb0画像だけ取り出す
fslroi subj2_dwi_ecc subj2_b0 0 1


# betでb0画像から脳だけ抽出する
bet subj2_b0 subj2_brain -f 0.3 -m
```

```
# dtifitでFA画像などを計算する
dtifit --data=subj2_dwi_ecc \
       --out=subj2 \
       --mask=subj2_brain_mask \
       --bvecs=subj2_dwi.bvec \
       --bvals=subj2_dwi.bval


# 結果を表示する
fsleyes subj2_FA subj2_V1 \
          -ot rgbvector \
          -mo subj2_FA &
```

スクリプトの実行

```
bash calc_fa_subj2.sh
```

- bashというコマンド言語インタプリタにcalc_fa_subj2.shを読み込んで実行

- 他の方法:ファイルに実行権限を与える

```
chmod 755 calc_fa_subj2.sh
./calc_fa_subj2.sh
```

コンテナの終了

- Linuxが起動しているブラウザを閉じる
- ターミナルから以下をタイプ

```
docker stop l4n
docker rm l4n
```

質疑応答

- 質問、感想などお聞かせください