

UNIX系OSのお作法

筑波大学医学医療系精神医学

根本 清貴

本日の内容

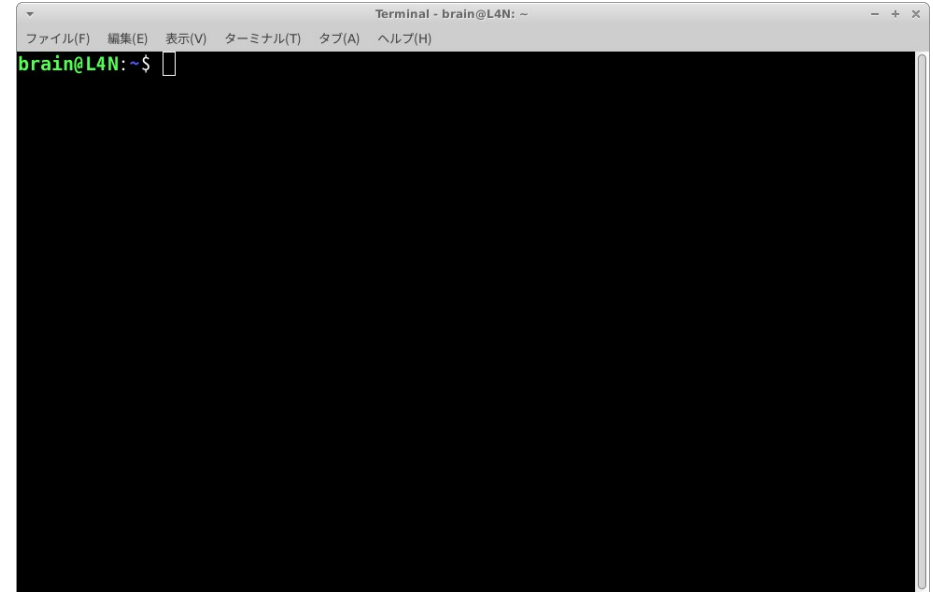
- 第1部:UNIX系OSのお作法を知る(兼予習)
- 第2部:テキストを処理してみる
- 第3部:画像解析ソフトのコマンドに慣れる
- 第4部:明日から使えるシェルスクリプトを知る
 1. 繰り返し
 2. 条件分岐

勉強会のルール

- 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

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
 - いくつかの方言
 - 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

- お作法1: ヒエラルキーがしっかりしている
- お作法2: パスの指定には「絶対パス」と「相対パス」がある
- お作法3: 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
- お作法4: コマンドについて知りたかったら **man** を使う
- お作法5: ファイル名にスペースと日本語は使わない
- お作法6: Tabキーを使って補完する
- お作法7: 自分の打ったコマンドの履歴は上下矢印で再現できる
- お作法8: 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
- お作法9: 動かなくなったら、**Ctrl** + **C** で中断

お作法1

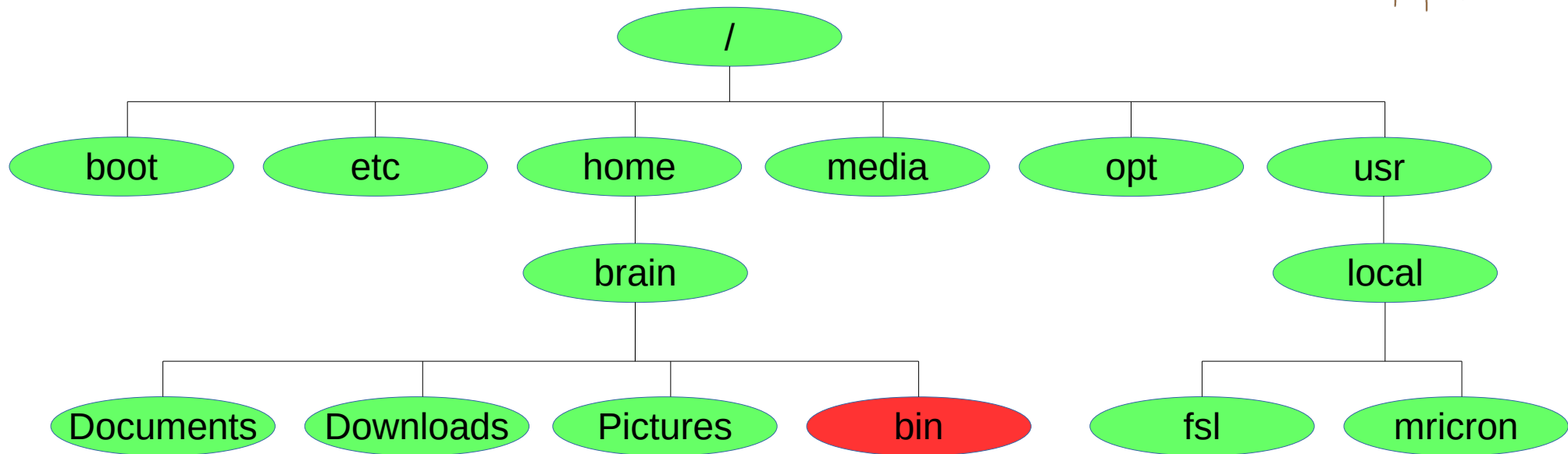
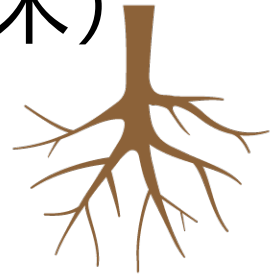
「ヒエラルキー」がしっかりしている！

ルートが一番上！

すべて「パス」で表示できる！

システムのヒエラルキーとパス

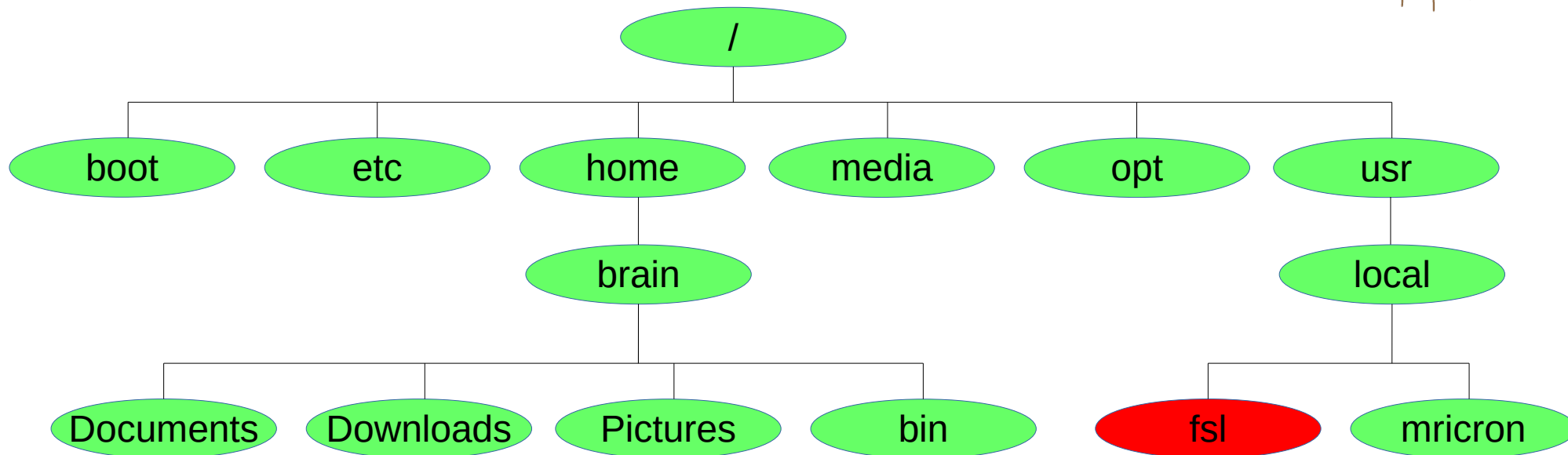
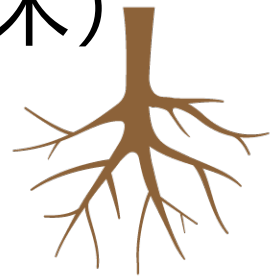
- UNIX系OSでは、頂点を root (/) といい、その下に様々なディレクトリが位置する (ひっくり返した木)
- ディレクトリまでの経路をパス (path) という



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



このパスは
`/usr/local/fsl`

お作法2

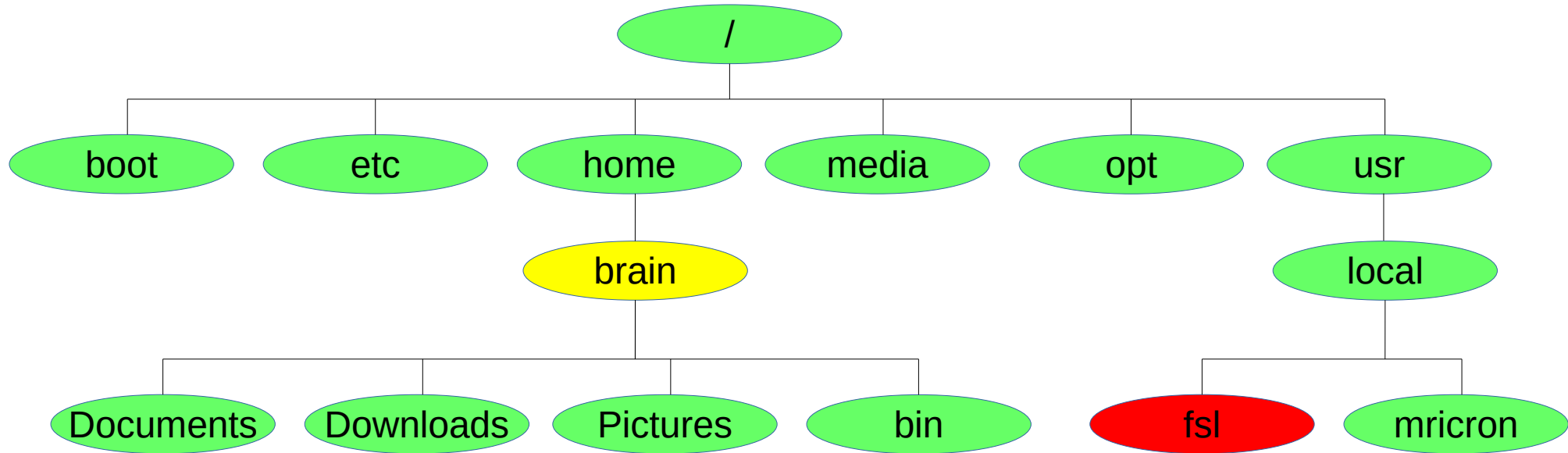
パスには「絶対パス」と「相対パス」がある！

パスを表示するための記号が3つある！

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localの下
のfsl

お作法3

コマンドは「命令」である！

英語での命令文をイメージする！

指定するものには
「引数」と「オプション」がある

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: $S V O \rightarrow$ 引数は1つ
 - `cd`, `mkdir`, `rmdir` など
- 第4文型: $S V O O \rightarrow$ 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	list	ファイル一覧を表示
cd	change directory	ディレクトリを移動
pwd	print working directory	現在のディレクトリを表示
cp	copy	コピー
mv	move	移動/リネーム
mkdir	make directory	ディレクトリの作成
rm	remove	削除
rmdir	remove directory	ディレクトリの削除
chmod	change mode	ファイル・ディレクトリの権限を設定
cat	concatenate	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	word count	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/ディレクトリ一覧が表示される
- 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
- 【演習】以下のコマンドをタイプしてください(\$はタイプしません)

```
$ ls /usr/local/fs1
```

```
$ ls -al /usr/local/fs1
```

ls -al

```
Terminal - brain@L4N: ~
ファイル(F) 編集(E) 表示(V) ターミナル(T) タブ(A) ヘルプ(H)
brain@L4N:~$ ls /usr/local/fsl/
LICENCE  bin      build.log  data  etc      fslpython  lib      refdoc  tcl
README   build    config     doc   extras  include    python   src
brain@L4N:~$
```

```
Terminal - brain@L4N: ~
ファイル(F) 編集(E) 表示(V) ターミナル(T) タブ(A) ヘルプ(H)
brain@L4N:~$ ls -al /usr/local/fsl
合計 2288
drwxr-xr-x 15 root root    4096 10月 28 11:33 .
drwxr-xr-x 22 root root    4096 10月 28 11:32 ..
-rw-r--r--  1 root root      55  9月  5  2012 LICENCE
-rw-r--r--  1 root root      70  9月 18  2004 README
drwxr-xr-x  3 root root   12288 10月 28 11:47 bin
-rwxr-xr-x  1 root root    1365  4月 18  2017 build
-rw-r--r--  1 root root 2257065 10月 24 04:03 build.log
drwxr-xr-x 35 root root    4096 10月 24 00:23 config
```

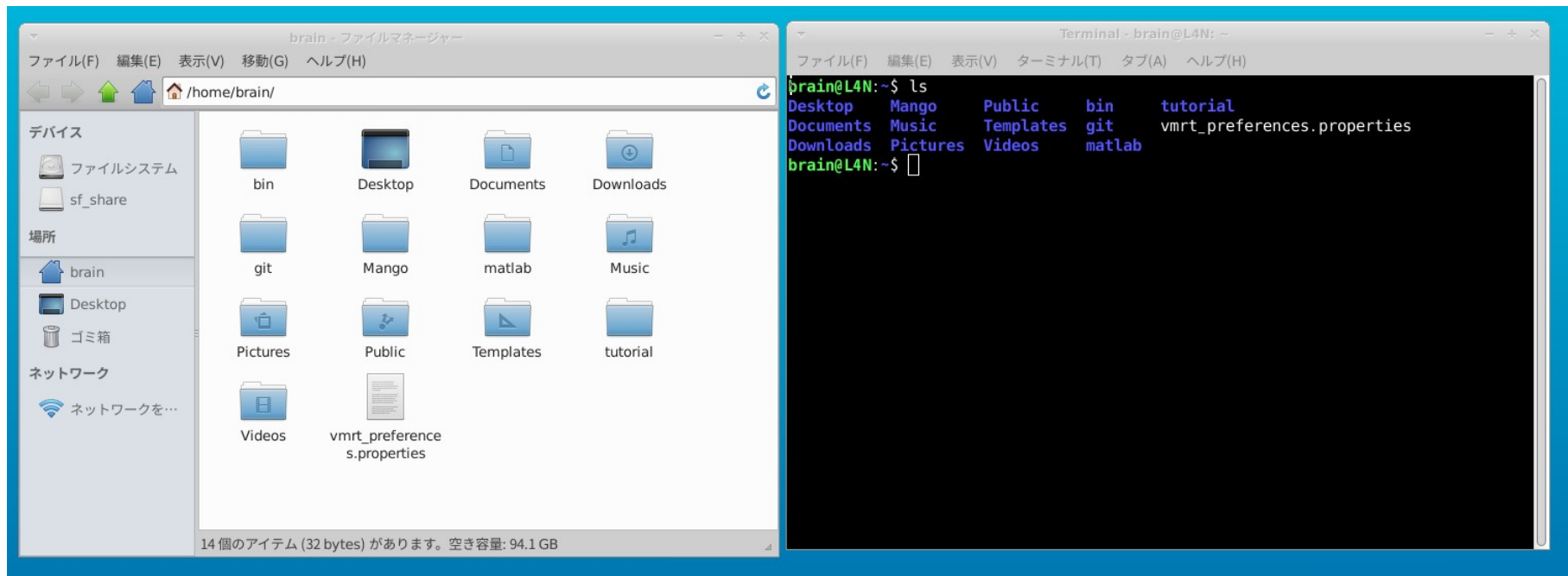
パーミッション 所有者 ファイルサイズ ファイル/ディレクトリ名

cd

- **change directory** 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！

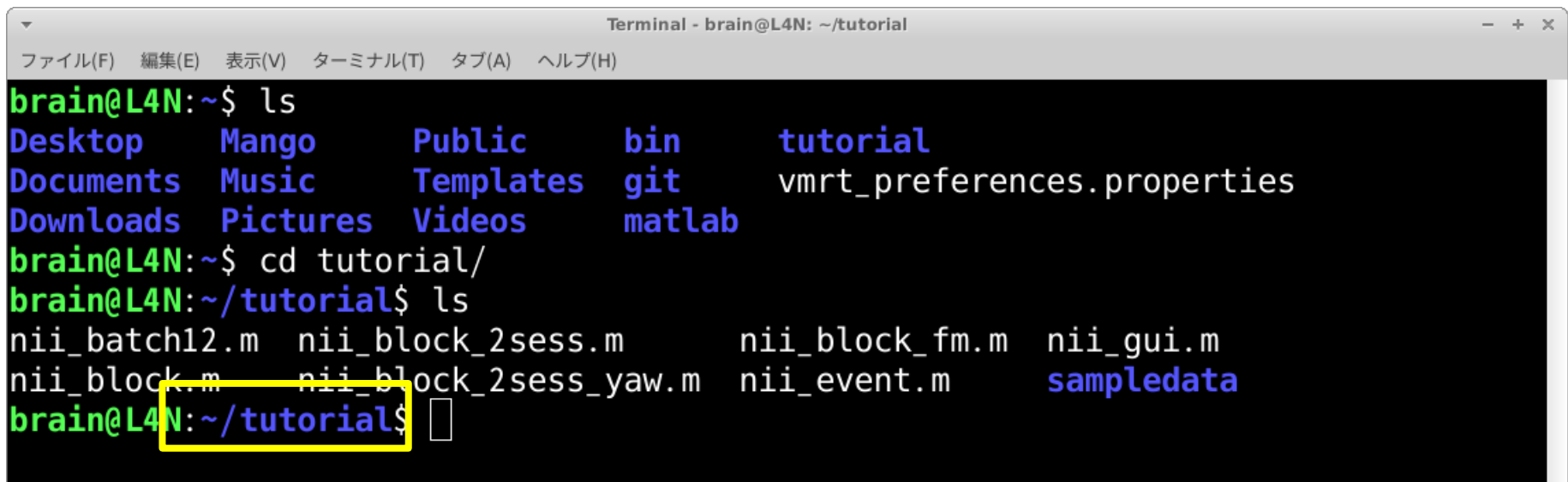
cd と ls の組み合わせ

- 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



pwd

- **print working directory** 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

A terminal window titled "Terminal - brain@L4N: ~/tutorial" with a menu bar (File, Edit, View, Terminal, Tabs, Help). The user runs 'ls' in the home directory, showing a list of folders and files. Then they run 'cd tutorial/' to move into the 'tutorial' directory. Finally, they run 'ls' again, and the prompt changes to 'brain@L4N: ~/tutorial\$'. The 'tutorial' part of the prompt is highlighted with a yellow box.

```
brain@L4N:~$ ls
Desktop    Mango      Public     bin         tutorial
Documents  Music      Templates  git         vmrt_preferences.properties
Downloads  Pictures   Videos    matlab

brain@L4N:~$ cd tutorial/
brain@L4N:~/tutorial$ ls
nii_batch12.m  nii_block_2sess.m      nii_block_fm.m  nii_gui.m
nii_block.m    nii_block_2sess_yaw.m  nii_event.m     sampledata

brain@L4N:~/tutorial$
```

現在のディレクトリは ~/tutorial

cp

- **copy** 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) しないで (move fileA as fileB)」
 - `mv fileA /path/dirC` で「fileA を /path/dirC に移動しないで (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

- **make directory** 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

rm

- **remove** 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
 - **-I** 3 個を超えるファイルを削除するか、ディレクトリを再帰的に 削除する場合は一度だけ確認をする。(Linuxのみ)
 - **-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

- **remove direcotry** 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

- **change mode** 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

- concatenate 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

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

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
- ***** 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

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: extract; v: verbose ; f: file

```
$ unzip file1.zip
```

```
$ gunzip file2.nii.gz
```

```
$ tar xvf file3.tar
```

```
$ tar xvzf file4.tar.gz
```

```
$ tar xvjf file5.tar.bz2
```


お作法4

コマンドについて知りたかったら
man を使う!

man

- **manual** の略語
- **man** コマンド名で「コマンド名のマニュアルを表示しなさい」という意味
 - **man cp** で cp に関するマニュアルが表示される
- マニュアルは、**less** を使って表示される
 - 終了したいときは、**q**をタイプ
- Linuxは日本語版のmanが準備されている
 - 以下でインストール
 - **\$ sudo apt install manpages-ja**
(Lin4Neuroの場合、パスワードは **lin4neuro**)

お作法5

ファイル名やディレクトリ名に
空白と日本語は使わない!

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

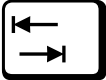
- 日本語は変換する手間がかかる
- 全角スペースはエラーの元になる
- コマンドは「英語」の発想でいくことが大事
- 以上のことより、コマンドラインで仕事をするときには、「ファイル名に空白や日本語は使わない」ことを習慣にすることが肝要

お作法6

Tabキーを使って補完する!

1回うちで補完、2回うちで候補表示

Tabキーは補完

- タイプするとき、長い文字を入れるのは苦痛
- UNIX系OSでは、数文字入れて Tabキー  を押すことで、後は補完してくれる
- これは便利だけでなく、エラーを回避するためにも重要

- 例

```
$ cd
```

```
$ cd bi 
```

```
$ cd bin/ #自動で補完される
```

Tabキーは2回うちで候補が出る

- Tabキーを2回うつと補完される候補が出る
- その候補を見ながらまたタイプすればよい

```
$ cd ~/git/shell-basic
```

```
$ cd d  
```

```
data/ docs/
```


お作法7

↑ ↓ は履歴が表示される
過去のコマンド一覧は **history** を使う

コマンドは記憶されている

- Linuxでは、デフォルトでは過去にうったコマンド2000個まで保存している
- このコマンドは、↑↓キーで確認できる
- また、`history` コマンドで一覧が表示される
- `history` コマンドの結果の最初に出てくる数字を！とともに使うことで、そのコマンドを再利用できる

`$ history`

(中略)

1258 `sudo apt install dkms`

1259 `sudo /sbin/vboxconfig`

`$!1259` `#sudo /sbin/vboxconfig` が実行される

お作法8

パイプを使ってコマンドを組み合わせ、
ほしい結果を得る！

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`

|

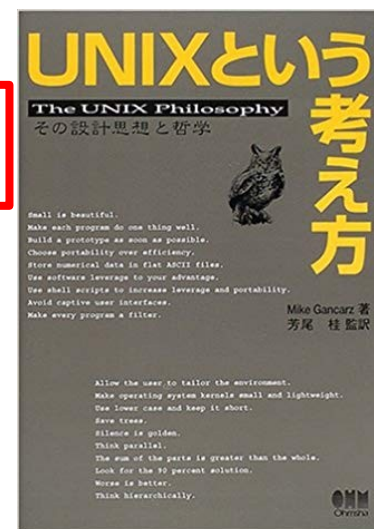
`wc`

`-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
- ソフトウェアを槌子(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



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

パイプの1例：自分がうったコマンドをカウントする

- 以下で、これまでに自分の打ったコマンドを頻度順にカウントできる
- (*いくつかのコマンドはまだ出てきていません。今はあくまでもパイプの役割を知るための例です)
- `$ history | perl -pe 's/\n//' | sed 's/ /\n/g' | sort | uniq -c | sort -nr | head`

381 ls

235 cd

129 |

59 ..

49 sudo

37 rm

33 mv

- 根本のワークステーションでは、過去にタイプしたコマンド2000個のうち、
 - ls を 381回
 - cd を 235回 タイプしている
 - パイプは 129回 使っている

お作法9

困ったら、Ctrl + C !

ターミナルで困ったら

- Windowsでは、`Ctrl` + `C` は、「コピー copy」のショートカットだが、Bash では、`Ctrl` + `C` は、「キャンセル cancel」のショートカット
- macOS では、コピーは `command` + `C` 、キャンセルは `control` + `C`
- タイプミスなどでターミナルがうまく動かなくなったら、`Ctrl` + `C` をタイプする
- スクリプトを実行した時に、中止したい時にも、`Ctrl` + `C` をタイプする
- 止め方を知っていると安心して作業できる

無限ループを止める

- ターミナルから以下をタイプ

```
$ num=0
```

```
$ while true
```

```
> do
```

```
> echo $num
```

```
> num=$((num + 1))
```

```
> done
```

- このコマンドは、数字に1をずっと足していく
- Ctrl** + **C** をタイプすることでキャンセルできる

演習(1)

1. ホームディレクトリの下に `shell_practice` というディレクトリを作成してください
2. `~/git/shell-basic/data/UnixIntro` を `~/shell_practice` にコピーしてください
3. `~/shell_practice` に移動してください
4. `~/shell_practice` の下にある `UnixIntro` を `UnixIntro_backup` としてコピーしてください
5. `UnixIntro` の中にあるファイルの一覧を表示してください
6. `UnixIntro` の中にある `stim1.txt` の内容を `cat` で表示してください

演習(2)

7. `UnixIntro` の中にある `design.fsf` の内容を `less` で表示してください
8. `UnixIntro_backup` を `backup` という名前に変更してください
9. `backup` を確認のプロンプトを出したうえで削除してください
10. `~/shell_practice/UnixIntro` の中にあるファイルの数をカウントしてください
11. ホームディレクトリに戻ってください
12. `cp` には「存在するファイルを上書きしない; Do not overwrite an existing file」オプションがあります。`man` でそのオプションを探してください

演習の回答例(1)

1. ホームディレクトリの下に `shell_practice` というディレクトリを作成してください

```
$ cd #ホームディレクトリに移動
```

```
$ mkdir shell_practice
```

```
$ ls #確認する習慣を!
```

2. `~/git/shell-basic/data/UnixIntro` を `~/shell_practice` にコピーしてください

方法1: 先に `~/git/shell-basic/data` に移動してから、`UnixIntro` を `~/shell_practice` にコピーする (Tabを上手に使って!)

```
$ cd ~/git/shell-basic/data/
```

```
$ ls
```

```
$ cp -r UnixIntro/ ~/shell_practice
```

方法2: `~/git/shell-basic/data/UnixIntro` をそのまま `~/shell_practice` にコピーする (回答省略)

演習の回答例(2)

3. ~/shell_practice に移動してください

```
$ cd ~/shell_practice/
```

```
$ ls
```

4. ~/shell_practice の下にある UnixIntro を UnixIntro_backup としてコピーしてください

```
$ cp -r UnixIntro/ UnixIntro_backup #Tabを使って!
```

5. UnixIntro の中にあるファイルの一覧を表示してください

```
$ ls UnixIntro/
```

6. UnixIntro の中にある stim1.txt の内容を cat で表示してください

```
$ cd UnixIntro/
```

```
$ ls #cd したら ls する習慣をつける
```

```
$ cat stim1.txt
```

演習の回答例(3)

7. UnixIntro の中にある `design.fsf` の内容を `less` で表示してください

```
$ less design.fsf (Qで終了)
```

\$ `cat design.fsf` では全ページが一度に表示されてしまう

8. UnixIntro_backup を backup という名前に変更してください

```
$ cd .. #上のディレクトリに移動
```

```
$ ls
```

```
$ mv UnixIntro_backup/ backup
```

9. backup を確認のプロンプトを出したうえで削除してください

```
$ rm -ir backup/
```

`-i` は削除の際に確認プロンプトを出すオプション、`-r` は再帰的に削除するオプション。2つを同時に `-ir` で指定できる

10. UnixIntro の中にあるファイルの数をカウントしてください

```
$ ls UnixIntro/ | wc -l #wc -wでもよい
```

演習の回答例(4)

11. ホームディレクトリに戻ってください

```
$ cd
```

12. `cp` には「存在するファイルを上書きしない; Do not overwrite an existing file」オプションがあります。`man` でそのオプションを探してください

```
cp -n
```

`man cp` の結果

-n, --no-clobber

存在するファイルを上書きしない

Do not overwrite an existing file.

- `~/git/shell-basic/textbook/UnixHandout2017.pdf` はFSL courseで配布されるハンドアウトです。本スライドはこのハンドアウトの大部分をカバーしています

質疑応答