コマンドライン入門

ウインドウが開くとファイルの一覧が表示され、それをクリックするとプログラムが立ち上がるのは 現在では常識になっているが、そのような直感的な仕組みは、1990年代以降に普及したにすぎない。 その前のおよそ20年間、コンピューターはキーボードからコマンドを入力して、操作するものだった。 現在のコンピューターにもその機能が残っている。

なお、マウスやタッチで操作する仕組みをGUI(Graphical User Interface)といい、コマンドラインを CUI(Character User Interface)という。理系の大学生は 1 年目に必ず習う。

マウスで絵を描いたり、写真を見ながらトリミングしたり、動画を見ながらカットするような 「本質的にマウスで操作する」作業は、コマンドラインではできない。

学習する理由

- オープンソースソフトを使うため
  オープンソースのソフトや公開プログラムは、（特定のOSでしか使えないGUIを避けて）コマンドラインが標準になっている。便利なプログラムを作ってくれる人の流儀に合わせるしか、恩恵は受けられない。
- 速い
  ファイルの特定の場所に書いてある数値を抜き出したいが、そのファイルは3000本ある場合。3000回ファイルを開いてコピペするのか？
- 自動化
  毎日決まった時間に特定のサイトをダウンロードして保存し、印刷したいような場合。

PowerShellとは

コンピューターに文字で命令を伝えるコマンドラインの画面自体もアプリで、「シェル」と呼ばれる。 Windowsには「コマンドプロンプト」というMS-DOS時代から使われてきた仕組みがあるが、（なにせ古いので）現在では、発展型としてWindows PowerShellという強力な仕組みが導入されている。 マイクロソフトが開発したが、オープンソース化され、MacやLinuxでも使用できる。

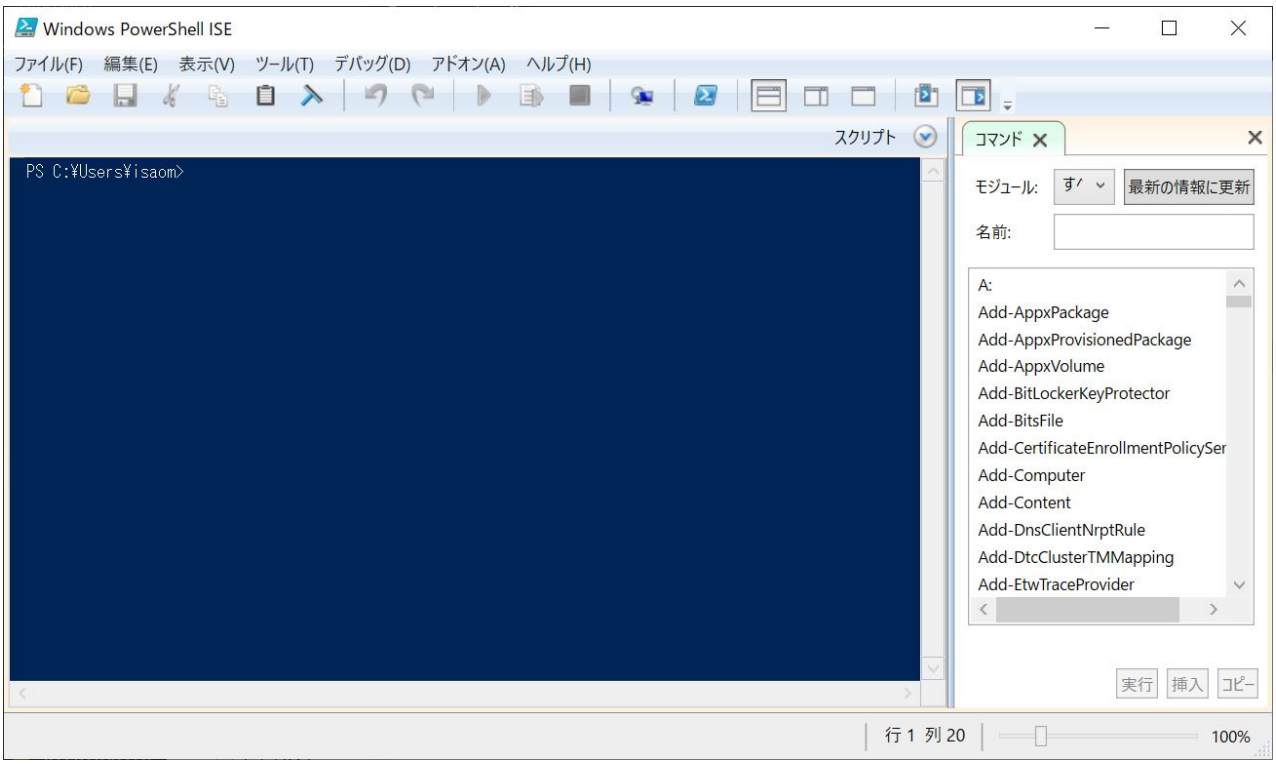
Macにインストールする方法

```
$ brew install --cask powershell
$ pwsh  # PowerShellをスタート
```

なお、Mac/Linuxは通常はBashというシェルを使っている。こちらが「シェルの本家」だが、 機能は、後発のPowerShellの方が断然進んでいる。

ウインドウがない世界

Windowsメニューから Windows PowerShell- Windows PowerShell ISE をクリックすると、 以下のようなウインドウが立ち上がる。



表示されているのは、現在の位置（カレントディレクトリ）と、点滅するカーソル（プロンプト）だ。おそらく、C:¥Users¥<自分のアカウント名>¥が表示されている。

```
$ ls (OR Get-ChildItem)
```

と入力すると、ファイル一覧が表示される（lsはlistの略）。 ウインドウズでいえば、このディレクトリが表示されている状態だ。

カレントディレクトリを移動するにはcd（change directory）で指示する。デスクトップに移動してみよう。

```
$ cd (OR Set-Location) Desktop
```

もちろん、「cd デスクトップ」と日本語で指定してもいい。再び、lsを入力すると、

```
$ ls (OR Get-ChildItem)
```

あなたの（全く整理されていない）デスクトップにあるファイルやフォルダが表示される。

デスクトップに展開した配布データ「DJN2021」のフォルダの中身をみるには、lsで直接ディレクトリを指定することもできる。

```
$ ls DJN2021
```

しかし、これからここで作業するので、移動しておこう。

```
$ cd DJN2021
$ ls
```

上のフォルダに移動したい場合もある。ひとつ上に階層に移動する場合には「..」、さらに上なら「..¥..」になる。

```
$ cd ..
```

時々、自分がどこにいるのか分からなくなる場合がある。 lsを叩くと中身が表示されるフォルダという意味で、pwd(present working directory?)を使う。

```
$ pwd (OR Get-Location)
```

いまの位置に新しいフォルダ（ディレクトリ）を作る場合はmkdir(make directory)で指示する。(削除はrmdir)

```
$  mkdir (OR New-Item -Type Directory) myFolder
```

このように、21世紀の人間にとって、ウインドウとマウスがない不便は著しい。

相対パスと絶対パス

ファイルやフォルダがアイコンで表現されるウインドウズシステムと違って、コマンドラインではファイル名を指定しなければならない。（というより、それが面倒なのでウインドウズシステムが発明された）

絶対パスとは、ファイルの場所と名前を完全に指定する方法で、例えば「C:¥Documents¥Newsletters¥Summer2020.pdf」などのように、ドライブからフォルダの階層、ファイル名まで¥でつないで表現する。この方法は、ファイルを移動させただけで全部書き換える必要がある。

相対パスとは、現在の位置からの相対位置で表現する。もし、現在の位置が「C:¥Documents」ならば、上のファイルは相対的に「Newsletters¥Summer2020.pdf」と表す。CDを使って、現在の位置を「C:¥Documents¥Newsletters」に移動すると、「Summer2020.pdf」になる。

Windowsの場合は、ドライブという概念がある(Macにはない)。ドライブ名を省略すると、現在のドライブの意味になる。パスが「¥」で始まっていれば絶対パス、始まっていなければ相対パスということになる。

通常、絶対パスは使わない。もし、ファイルをDドライブに移動させたらすべて修正する必要があるし、フォルダを多くの人に配布した場合、「C:¥Users¥<ユーザーアカウント名>¥Documents¥Newsletters¥Summer2020.pdf」のように、絶対パスが人によって違うことになるからだ。

パスを示すとき、特別な意味を持つ記号がある。これは絶対覚えよう。

- .(点一つ) = 現在の位置
- ..(点２つ) = 一つ上のディレクトリ
- ~ (チルダ) = ユーザーのホームディレクトリ（C:¥Users¥<ユーザーアカウント名>）

コマンドの約束事

lsに「-Recurse」を追加すると「フォルダの中のフォルダの中の...と再帰的に」がファイルが表示される。例えば、

```
$ ls -Recurse
```

このように、英語の命令形と同じようにコマンドに続いて、 対象ファイル（あるいはディレクトリ）やオプション（ **大抵は-で始まる** ）を指定する。

コマンドの（オプションを含めた）使い方は、本やネットで調べる必要がある。 ただし、一度使い方を覚えてしまえば、簡易のマニュアルは以下のようにhelp(OR man)で表示することができる。

```
$  help ls
```

の使い方が表示される。「q」を押せば終了する（quitという意味）。このhelpというコマンド自体もコマンドだ。

ただし、help(man)が知らないコマンドもある。あるいは、

```
$  info pwd
$  pwd -?
$  pwd --help
```

のようなオプションで使い方が表示されるかもしれない。 残念ながら、すべてのコマンドに簡易マニュアルが装備されているという保証はない。

一般的なファイル操作

デスクトップでできることはコマンドラインでもできる。

ファイルの中身が見たい場合

```
$ cat(OR Get-Content) test.txt
$ Get-Content lines.txt -TotalCount 10  # 最初の10行
$ Get-Content lines.txt -Tail 10        # 最後の10行
$ Get-Content utf8.txt -Encoding Utf8   # 文字コード指定
```

ファイルの移動は

```
$ mv test.txt myFolder
$ Move-Item test.txt -Destination myFolder
$ Move-Item test.txt -Destination myFolder -Force # 強制上書きしたいとき
```

ファイル名の変更は

```
$ mv(OR Rename-Item) test.txt new_test.txt  （ファイル名変更）
```

ファイルの複製は

```
$ cp(OR Copy-Item) new_test.txt myFolder¥copied_text.txt
```

ファイルの削除は

```
$ rm(OR Remove-Item) new_test.txt
$ rm -rf(OR Remove-Item) myFolder  # フォルダごと削除する場合
```

オプションとして、「*」（ワイルドカードという）が使えることがあり、例えば 「*.txt」は「.txt」で終わるすべてのファイルを意味する。

```
$ rm *.txt（注意！実行すると全部のテキストファイルが削除されてしまう）
```

OSに対する操作

psは、パソコン上で動いているプログラムの一覧を表示する。killはプロセスIDで指定したプログラムを強制終了する。

```
$ ps ax  (axはすべてのプロセスを表示せよというオプション)
3273   ??  S      0:00.03 /System/Library/Frameworks/AddressBo.....(略)
1283 s001  S      0:00.14 -bash
3274 s001  R+     0:00.00 ps ax
$ kill 1283  (1283番の-bashを強制終了する)
```

とにかく、デスクトップ上でできることはコマンドラインでもできる。

コマンドは他にもあるが、複雑なオプションを丸暗記するほど暇な記者はいないので、 適宜、参考書を開くか、ネットで調べればよい。多用するコマンドは自然に覚えてしまう。

「パスを通す」「パスを追加する」とは？

ところで、あるコマンドがそのコンピューターに実在するのか、打ち間違いなのか、コンピューターはどうやって判断するのか？

コンピューターには、環境変数（システム共通環境変数とユーザーごとの環境変数）にパス(Path)という「フォルダのリストの変数」がある。 シェル(コマンドプロンプト)は、コマンドが入力されると、Pathにあるフォルダに、コマンド名のプログラムがあるかどうかを検索する。 そこに見つからなければ、「そんなコマンド知りません。スペルミスではないですか」というエラーを出す。数TBもある巨大なハードディスク全体を検索すると数時間かかるからだ。

その結果、新しいソフトをインストールする場合、①すでにPath変数にあるフォルダにプログラムを設置するか、②プログラムを設置したフォルダをPath変数に追加する必要がある。 この「Path変数に追加する」作業を「パスを通す」とか「パスを追加する」という。

ImageMagickのインストール

ImageMagick <https://imagemagick.org> は、1987年に米DuPont社のJohn Cristyが開発を始めたオープンソースソフトで、 GIF、JPEG、JPEG 2000、PNG、PDF、Photo CD、TIFF、DPXなど200種類近いフォーマットに対応している。 「コマンドラインのPhotoshop」と呼ばれる。

Windowsの場合、ダウンロードのページから <https://download.imagemagick.org/ImageMagick/download/binaries/ImageMagick-7.0.10-58-Q16-HDRI-x64-dll.exe>をダウンロードする。

インストール画面の途中で、英語で「パスを追加する」というオプションがあることに注意すること！（つまり、このインストールソフトは気を利かせて自動で「パスを通し」てくれる）

Windowsの場合、デフォルトのインストールによって、ffmpegもインストールされる。 ffmpeg <https://ffmpeg.org> は、2000年にフランスのFabrice Bellardが開発を始めたオープンソースソフトで、動画のフォーマット変換、カットやトリミング、フィルター処理、字幕の焼き込みなど様々なことがコマンドラインで処理できる。「コマンドラインのPremiere」と呼ばれる。

なお、Macの場合は、以下のコマンドでインストールできる。

```
$ brew install imagemagick
$ brew install ffmpeg
```

ImageMagickを体験

最新のImageMagick（バージョン7）は、コマンド名がmagickに変更されている。 ネットにある解説の多くはバージョン６のものなので、convert/display/animateなどのコマンド名になっている。 それらは、magick convert/magick display/magick animateのようにmagickを前につけると動くはず...

サイズ変換

```
cd images
magick convert marathon01.jpg -resize 50% marathon01_half.jpg
magick convert marathon01.jpg -resize 128x marathon01_width128.jpg
```

フォーマットを変換

```
magick convert marathon01.jpg marathon01.png
```

文字から画像を作ることもできる。

```
magick convert -size 256x50 canvas:none -pointsize 32 -font "Meiryo-&-Meiryo-Italic-&
```

写真の合成

```
magick convert marathon01.jpg JNPC.png -gravity southeast -compose over -composite ma
```

写真のexif情報

```
magick identify -verbose marathon01.jpg
```

アニメーションGIFも可能

```
magick convert marathon02.jpg -resize 256x marathon02_width256.jpg
magick convert marathon03.jpg -resize 256x marathon03_width256.jpg
# delayは100分の1秒単位、最初に指定すること
magick convert -delay 500 marathon02_width256.jpg marathon03_width256.jpg -loop 10 -d
```

長く開発されているので、写真のトリミング、フィルタ、変形など、思いつくことはできる。 詳細は、公式マニュアル <https://imagemagick.org/script/command-line-tools.php> を参照のこと。 日本語のブログも多数あるが、version6に関する解説が多いので要注意。

ffmpegの利用法は次回「Visual Forensic入門」で行う。 詳細は公式マニュアル <https://ffmpeg.org/documentation.html> を参照のこと。

リダイレクトとパイプ

リダイレクトとパイプは、単純な作業しかできないコマンドを繋げて、 複雑なことを実現する仕組み。

出力リダイレクト

コマンドラインは通常、キーボードから入力を受け付け、画面に結果を出力する。

echoという、文字列を出力するだけのコマンドもある。

```
$ echo(OR Write-Output) "Hello"
Hello
```

この出力を、「>」を使ってファイルにリダイレクト（出力先を変更する）することができる。

```
$ echo "Hello" > hello.txt
```

これだけで、カレントディレクトリにhello.txtというファイルができる。中身は当然Helloである。

追記リダイレクト「>>」を使用すると、もし既にファイルが存在する場合は追記するようになる。

catという、ファイルの中身を出力するだけのコマンドもある。

```
$ cat (OR Get-Content) japanse.txt
(ファイルの中身がテキストとして表示される)
```

これを頭にいれて、シェル操作の華、パイプに移ろう。

パイプ

パイプはコマンドをつなげる方法で、コマンドを「｜」（vertical bar = 縦棒）で繋ぐ。

```
あるコマンド | 次のコマンド
```

これで、あるコマンドの出力がそのまま、次のコマンドの入力になる。

それがどうしたと思うかもしれないが、たとえば、japanse.txtが、日本語Windows標準のshift-jisでつくられたファイルの場合、ユニコードに変換するには...

```
cat japanese.txt | Set-Content -Encoding UTF8 japanese_UTF8.txt
# 文字コードを明記するなら
Get-Content -Encoding shift-jis japanese.txt | Out-File -Encoding utf-8 japanese_UTF8
```

スクリプト

コマンドラインで一行一行、対話的に操作するだけでなく、テキストファイルにコマンドを書いておき、一気に実行することができる（バッチ処理ともいう）。

（ところでダウンロードした（他人が書いた）スクリプトファイルは直接実行できないように保護がかかっているので、どうしても実行したいなら以下のコマンドで解除してください。おすすめはscriptsフォルダにあるファイルをメモ帳で開いて、自分で作ったファイルにコピペすることです）

```
Set-ExecutionPolicy RemoteSigned  # 自分で作ったファイルは実行できる
Set-ExecutionPolicy Unrestricted  # 注意: 保護が効かなくなる
```

ここから本番。例えば、sample01.ps1(scriptsフォルダにある)という、以下のようなファイル（バッチファイル）をメモ帳で作成する。

```
echo "hello" > test.txt
Rename-Item test.txt new_test.txt
```

このファイルを実行する。（./は現在のディレクトリを表す）

```
./sample01.ps1
```

一瞬でファイルを作り、それを改名するので、new_text.txtが残る。

このスクリプトはキーボード操作の決め打ちに過ぎないが、PowerShellには、変数やループなど、スクリプトを賢いプログラムに進化させる機能がそろっている。それを説明する時間も能力もないし、ネット上にアドバイスは溢れているので、典型的な利用例を紹介する。

スクリプトの応用例

scripts/collectFiles.ps1: フォルダにある特定の拡張子のファイルを集める例

```
# フォルダにある.txtファイルを探し
# そのコピーをcollectedDirectoryに集める

$targetDirectory = "scripts"    # 対象ディレクトリ（書き換え可）
$collectedDirectory = "collectedFiles"  # 回収フォルダ

# 回収ディレクトリを作る
mkdir $collectedDirectory

# すべてのファイルを（フォルダがあればさらにその中まで）集める
Get-ChildItem -Path $targetDirectory -Recurse |
        # 拡張子がtxtと同じものだけに限る（フィルタリング）
        Where-Object {$_.Extension.ToLower() -eq ".txt"} |
        # それぞれのファイルについて
        ForEach-Object {
                # 画面に現在のファイル（$_）の名前を表示する
                Write-Output ("copying...  " + $_)
                # 現在のファイルを回収ディレクトリに同じ名前でコピーする
                Copy-Item $_.FullName ("./" + $collectedDirectory + "/" + $_.Name)
        }
```

注意: $targetDirectoryの中に$collectedDirectoryがあると、コピーしたものが再びコピーの対象になるので エラーが出るかもしれない。コピー先のディレクトリは対象ディレクトリの外に置くべき。

scripts/fetchRemoteFiles.ps1: ネット上にある規則的な名前のファイルを一気に集める例

```
# 日本卸電力取引所が公開している電力料金のスポット価格は
# http://jepx.org/data/20210114.csv のように日付名のcsvファイルとして
# 翌日にはネットで公開されている。

# ファイルを保存するディレクトリを作る
mkdir -Force "PowerPrices";

# 収集したい初日
$dt = [DateTime]::ParseExact("2021-01-01","yyyy-MM-dd", $null)
$today = Get-Date
while ($dt -lt $today) {
        # 取得したいファイルのURL
        $url = "http://jepx.org/data/" + $dt.ToString("yyyyMMdd") + ".csv"
        # 保存したい名前
        $outputFilename = "PowerPrices/" + $dt.ToString("yyyyMMdd") + ".csv"
        # 進行状況を表示
        echo ("calling Invoke-WebRequest " + $url + " -OutFile " + $outputFilename)
        # URLのファイルをダウンロードしてファイルに保存する
        Invoke-WebRequest $url -OutFile $outputFilename
        # エチケットとして５秒待つ
        Start-Sleep -s 5
        # 日付を１日進めてwhileの先頭に戻る
        $dt = $dt.AddDays(1)
}
```

scripts/concatenateFiles.ps1: フォルダにある複数のファイル（txtやcsv）を連結する例

```
# フォルダにある.csvファイルを探し
# mergedCSVに文字通り「一本化」する

$targetDirectory = "PowerPrices"
$mergedCSV = "power_prices_2021.csv"

Get-ChildItem -Path $targetDirectory |
        Where-Object {$_.Extension.ToLower() -eq ".csv"} |
        ForEach-Object {
                # 進行状況を表示
                Write-Output ("appending...  " + $_.FullName)
                # >>は追記リダイレクトなので、同じファイルの最後に書き加える
                cat $_.FullName >> $mergedCSV
        }
```

power_prices_2021.csvをエクセルやLibreOfficeで確認してください。

なんのためにImageMagickを紹介したか？

scripts/largePNG2JPG.ps1:

```
# フォルダにある.pngファイルを探し
# サイズが1MB以上ある写真だけ
# jpgに変換してcollectedDirectoryに保存する

$targetDirectory = "./"
$collectedDirectory = "jpegFiles"

mkdir $collectedDirectory

Get-ChildItem -Path $targetDirectory -Recurse |
        Where-Object {$_.Extension.ToLower() -eq ".png"} |
        ForEach-Object {
                # ファイルサイズが1MBより大きいなら
```

```
                if($_.Length -gt 1024*1024) {
                        # 進行状況を表示
                        Write-Output ("saving as jpg ...  " + $_.Name)
                        # ImageMagickのコマンド
                        # ちょっと難しいが、-replace演算子で拡張子の.pngを.jpgに置換
                        magick convert $_.FullName ($collectedDirectory + "/" + ($_.Na
                }
        }
```

コマンドラインで動くソフトは、このようにスクリプトで大量のファイルに対して酷使されることを想定して作られている。だからこそ、エンジニアはいまでもコマンドラインを使う。

我々がコンピューターに期待すること、例えば、定時にあるサイトで発表されるデータを自動で保存してほしいとか、事件の舞台となった団体のサイトのすべてのページを（サイトがダウン・閉鎖される前に）一気に保存しておきたいとか、コマンドラインの活用方法はいくらでもある。

参照:コロナが変えた航跡 <https://static.tokyo-np.co.jp/tokyo-np/pages/feature/corona/how_our_sky_changed.html>　　:審判を審判する2020 <https://static.chunichi.co.jp/chunichi/archives/ee/feature/figureskating2020/judging_the_judges.html>

参考になるサイト

- 本家PowerShell の公式製品ドキュメント: <https://docs.microsoft.com/ja-jp/powershell/?view=powershell-7.1>
- 多数のブログがある。例えばこのようなサイト: <https://bayashita.com/p/category/show/22>