厳選 Unix コマンド v2.0.0

— Selected Unix Commands for Beginners —

［著］公立千歳科学技術大学 IT インフラ部
［監修］深町賢一

2025 年 2 月 28 日　v2.0.0

■免責

本書は情報の提供のみを目的としています。
本書の内容を実行・適用・運用したことで何が起きようとも、それは実行・適用・運用した人自身の責任であり、著者や関係者はいかなる責任も負いません。

■商標

本書に登場するシステム名や製品名は、関係各社の商標または登録商標です。
また本書では、™、®、© などのマークは省略しています。

まえがき / はじめに

IT インフラ修行中の三年生が製作した「厳選 Unix コマンド」第 2 版をお届けします。執筆者一同の考える「おすすめ」かつ「必須」Unix コマンド群です。

この書籍には電子版と印刷版があります。中身は同一です。どちらも、次の URL からダウンロードできます。

`https://selected-unix-commands.techbooks.fml.org/`

学内で印刷版も無料配布しているので手に取ってみてください。H205 や情報棟一階、図書館で配布しています

本書の対象読者

おおまかには Unix オペレーティングシステムのコマンド操作の初心者が対象ですが、授業の構成を考えると、情報システム工学科 3 年春学期に副読本として、おすすめです。演習授業のおともに、ご活用ください

第 1 版と第 2 版の違い

- 3 年生の授業で用いるコマンドだけに絞ってあります。いわゆる「カリカリにチューン」された内容です
  - 授業で出てこなくても知っておいた方が良い知識やオプション等が執筆者と監修者の判断で少し追加されています
  - コマンド大辞典のような書籍が必要なら、そういった商業出版物が多数あります。必要な方は、そちらを読んでください
- 第 1 章「**Unix コマンド操作とは**」(Unix 小論) と**索引**を追加
- 第 2 版はレビュー (by 深町) されているので、内容は信用して大丈夫です

問い合わせ先

メール: infra-club@cist.fml.org

学内のかたは、H205 に遊びにきてくれると歓迎されます

感想・意見等をお待ちしております

凡例

第 2 章（chapter）の各節（section: 2.1, 2.2, ...）が 1 つのコマンドを解説している。

♣ 各節の基本フォーマット

各節では、コマンドの簡単な紹介の後、「コマンドの書式」「実行例」「実行結果」を示す。「実行例」は、実際に打ちこむコマンドの例。「実行結果」には「実行例」の行と、それを実行した結果（出力）が載っている（つまり「実行例」の行は重複している）。また、実行結果のあとに、オプションの解説がある節もある。

♣ 各節の構成

「コマンドの書式」「実行例」「実行結果」すべての左端にある**$** はシェルの**プロンプト**である。打ちこむコマンドには含めないことに注意。また、実行例にはイチイチ書いていないが、コマンド入力行の最後には ENTER も必要である。

```
$ ls
```

この例で、ユーザが打ちこむのは **ls**（に続けて ENTER）だけである。

▼書式、実行例、実行結果

```
2.14.1 書式

$ ls [options] [FILE]...

2.14.2 実行例

実行例

$ ls

実行結果

$ ls
www.py
```

書式の **[ ]** は「あっても無くてもかまわない」部分つまりオプションを意味する。また、 **...** は可変長を意味している。

実行結果の中には、右端に**左矢印**に続いて**コメント**が追記されたものがある。これは Unix からの出力ではなく、本書が追記した**解説**なので注意してほしい。

▼実行結果の右端にある矢印はコメント

```
ping.re:^C                                    ← Ctrl-Cを打ち込んだ行
```

この例では「**← Ctrl-C を打ち込んだ行** 」が本書で追記した**解説**部分である。

用語

♣ 用語 (Unix)

- **Unix** ... オペレーティングシステムの名前。プロトタイプは 1969 年の夏に誕生。オリジナルの作者は Ken Thompson。C 言語は Unix を開発するために設計された（キレイに今の形にしたのは主に Dennis Ritchie なんでしょうけど:-)。ちなみに UTF-8 と Go 言語も Thomposon が作者の一人。
- **カーネル** ... OS の中核機能にあたるシステム・ソフトウエア
- **シェル** ... ユーザとの対話処理を行うプログラム
- **プロンプト** ... シェルが「入力待ち」状態を示す行左端の特殊文字。例: **$**, **>**
- **コマンド** ... 対話処理の際にユーザが打ちこむ命令（プログラム名 [引数...]）
- **CLI** ... Command Line Interface の頭文字。コマンドを入力して対話すること。CUI は同義語。反対語は GUI(Graphical User Interface)
- **ターミナル** ... ユーザが入出力をおこなうアプリケーションのこと。言葉の原義どおり（コンピュータの）末端という意味。もともとはターミナルという装置があった (「1.3 対話処理」(p.2) を参照)。文脈によっては「シェル」と同義語のこともある。
- **端末** ... ターミナルの翻訳語・同義語
- **Berkeley** ... (1) カルフォルニア大学バークレイ校 (2) この業界でバークレイと言えば BSD Unix か、その開発グループ (CSRG[*1]) のこと
- **BSD** ... Berkeley Software Distribution の頭文字。Bill Joy[*2]が考えたカッコいい用語。たぶん意識的にディストリビューションと名づけてソフトウエアを配布したのは、これが最初だと思う
- **BSD Unix** ... (1) ARPA と契約してバークレイが開発していた AT&T Unix の改良版 (〜4.3BSD Reno) (2) CSRG 後期に、フリーソフトウエアとして作られた 4.4BSD および 4.4BSD の子孫たち。たいてい BSD とは (2) の話
- **Linux** ... 1990 年代にフィンランドで作られたカーネルの名前
- **Debian** ... Debian GNU/Linux。30 年以上もコミュニティベースで開発が続けられてきた、もっとも歴史のある Linux ディストリビューションの一つ。多くの開発/製品のベースシステムとしても使われている
- **Ubuntu** ... Debian をベースに Canonical 社が開発している Linux ディストリビューションの一つ。見栄えがいいから？ か、いやに素人に人気がある。Canonical 社の商用サポートが受けられる Debian 系というところもポイント

[*1] CSRG ... Computer Systems Research Group

[*2] Bill Joy ... vi の作者、BSD Unix の主要開発者、CSRG 前期のウイザード級ハッカー

♣ 用語（ネットワーク）

- IP ... IP アドレス。コンピュータに付ける識別子。バージョン 4 と 6 がある
- IPv4 ... バージョン 4 の IP プロトコルおよび IP アドレス。IP アドレスの長さは 32 ビット。2 進数のままでは人間に辛いので、8 ビットずつ計 4 つに分割し、それぞれを 10 進数に変換した後にドット（.）でつなげた表記法を使う。例: 192.168.10.1
- Public IP ... 世界で一意に割り当てられた IP アドレス。サイバースペースにおける住所に相当する。別名グローバル IP
- Private IP ... 各組織が自由に利用できる IP アドレス。Public IP の逆
- ドメイン名 ... www.example.co.jp のように英数字をドットでつないだ文字列。必須ではないが、あったほうが便利なインターネットの識別子[*3]
- TCP/IP ... インターネットを代表するデータ転送プロトコルの総称
- ポート番号 ... TCP/IP の各通信（セッション）を識別する 16 ビットの数字
- ソケット ... ネットワークの出入り口のこと。ソケットは「電源のソケット」(電気の出入り口) と同じ単語。ソケットには IP アドレスやポート番号、プロトコルが紐づいてる

♣ 用語（クラウドコンピューティング）

- クラウド ... 「さまざまな IT リソースをオンデマンドで利用することができるサービスの総称です。必要なときに必要な量のリソースへ簡単にアクセスすることができ、ご利用料金は実際につかった分のお支払いのみといった重量課金が一般的です。」(AWS のウエブ[*4]より引用)
- AWS ... Amazon Web Services の略。Amazon が提供するクラウドコンピューティングサービスの総称
- EC2 ... AWS を代表するサービス。OS をインストールして使う仮想の PC
- S3 ... AWS を代表するサービス。オブジェクトストレージ

[*3] ドメイン名の取得は早いものがちなので、組織名などを表す部分が 2〜3 文字の短いドメインは商用インターネット初期に取られたプレミアムもの、ある意味ステータス:-)

[*4] `https://aws.amazon.com/jp/cloud/`

目次

第 1 章
Unix コマンド操作とは

1.1 なぜ今さら Unix の訓練が必要なのか？

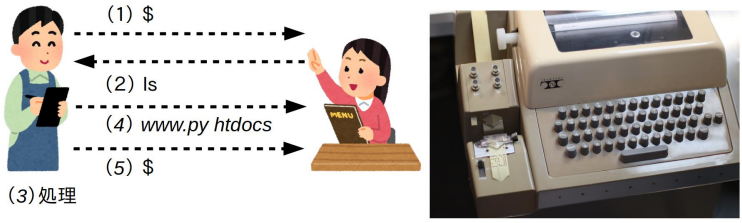
「競争力のある IT サービスを作れるようになるため」が答えである。

いまどきは、ブラウザ上でクリックすれば誰でも定番の IT サービスを立ち上げることが出来る。クラウド体験であれば、これで十分だが、定番のサービスしか作れないのでは IT のプロとして食べていけない。

それよりも、定番を構築できるライバルは世界に何千万人もいることに気づいてほしい。クラウド体験は何の競争力も産まない。クラウドサービスで、独自の IT システムを作りこんだり細かな調整をしようとすれば、どうしてもクラウドの操作画面の一皮下の OS つまり Unix を触る必要がある。

1.2 Unix は元々が研究者用 OS

Unix は素人向けではない。少人数の研究者/開発者が使う際の OS の理想像といったところだ。のちに色々なウインドウシステムやアプリを追加して、素人向けにも使えるようにしてきたが、非定番の使い方をしようとすると真の Unix が顔を出してくる。必要最小限、余計なことは言わない、いかにキーボードを打つ回数を減らすか？そういったプロ向けの設計思想が初心者の敷居を高くしている所以だろう。



▲図 1.1: 左は対話処理のイメージ。右は代表的なターミナル:Teletype Model 33

1.3　対話処理

▼対話処理（図 1.1 左側を参照）

```
$                   ← (1)「入力待ち」の意思表示としてプロンプトを表示
$ ls                ← (1)の状態から、(2)ユーザがlsと入力し、ENTERを押す（Unixにlsの依頼が伝わる）
www.py htdocs       ← (3)（lsの依頼が）処理され、(4)lsの結果が表示される
$                   ← (5)プロンプトを表示（1.の状態に戻った）
```

図 1.1（左）は、給仕（OS）と客（ユーザ）との対話イメージである。図 1.1（左）は、上述の操作と対応している。これが Unix で言うところの対話処理である。

背景事情を少しだけ説明しよう。図 1.1（右）の機材は、1970 年代に Unix が開発されていた当時のターミナル[*1][*2][*3]である。ほぼ電動タイプライターだ。タイプすると入力が Unix に伝わり、処理され、Unix からの出力が一行ずつ印刷される。これが、この当時の「コンピュータとの対話」である。

もともと、コンピュータに（一方的に）命令を伝えて、コンピュータが何時間も計算するという使い方（バッチ処理）が主流だった。それにくらべれば「対話」であって、現代語の対話とはイメージが異なるだろう。

現代では物理的なターミナルは使われず、ウィンドウシステム上のターミナルアプリ（ターミナルをエミュレーションするソフトウエア）を使うことが普通だ。しかしながら、見かけはモダンになってはいても、やっていることは図 1.1 から進化していないことを覚えておきたい。

[*1] "Teletype Model 33 - 48930224272" by stiefkind is marked with CC0 1.0
[*2] Unix の tty という用語は、この Teletype に由来する
[*3] （IT 古典部むけの脚注:-）左端は紙テープに穴 (あり/なしで 1 ビットを表現) をあける機材。この写真は Teletype model 33 の中でも、この機材付きモデルのもの

第 2 章
厳選 Unix コマンド

2.1 aws

aws コマンドを使うと、コマンドラインから AWS サービスを利用できる。

AWS 固有のコマンドなので、当然 Unix 標準ではない。AWS 提供の EC2 の OS イメージでは aws コマンドが標準搭載のようだが、OS によっては別途インストールが必要である

♣ 2.1.1 書式

```
$ aws [options] command subcommand [parameters]
```

- command 部分と subcommand 部分の指定は必須である
  - command 部分にはサービス名（例：ec2、s3）を指定する
  - subcommand 部分では、その command サービス固有の動作指定を行う
- [options] と [parameters] はオプション

♣ 2.1.2 実行例

実行例 （引数がなくエラー、ヘルプメッセージが表示される）

```
$ aws
```

実行結果

```
$ aws
To see help text, you can run:
```

```
  aws help
  aws <command> help
  aws <command> <subcommand> help

aws: error: the following arguments are required: command
```

aws 単体だとエラーとヘルプメッセージが表示される。

コマンドの使い方が分からない時

コマンドの使い方が分からない時に、「わざとエラーを起こしてヘルプメッセージを表示させる」技は、ほぼ、どんなときでも使えるので覚えておきたい

♣ 2.1.3　aws s3 subcommand

aws で使えるコマンドは様々であるが、今回はサービスの一つである S3 のみを紹介する。

S3（Simple Storage Service）は AWS を代表するサービスの一つである。きわめて耐久性が高く、**99.999999999 %（イレブン・ナイン）を超えるデータ耐久性**を唱う。S3 の代表的な使い方は「ストレージサービス（データの長期保存）」と「静的 Web サイト (事前に HTML を作成し、その通りに動くサイト) の公開」である。

S3 を利用する際には、まず**バケット**と呼ばれるファイルの入れ物を作成する。これは Unix のディレクトリや Windows のフォルダに相当するものと考えて良い。ユーザは、このバケットにファイルをアップロードしていくことになる。

aws コマンドで S3 サービスを呼び出す場合、command 部分には s3 を指定する。subcommand で様々なファイル操作が行える。例：

- バケットの内部を確認する際は ls
- データの送受信は cp もしくは sync
  - cp はファイル、sync はバケットとフォルダを同期
    注：操作方法は cp も sync も同じなので sync の実行例は省略する

前提

以下の実行例では、次の前提がある

- $ 行は、AWS の EC2 にログインしている状態での操作である
  - ユーザ admin のホームディレクトリ (/home/admin) で作業している
  - このディレクトリには www.py ファイルと htdocs ディレクトリがある
- AWS S3 の管理画面で、事前にバケットは作成済
  - バケット名は testBucket

- バケットは URL で指定する。書式はプロトコル://バケット名
  - s3://testBucket

♣ 2.1.4　実行例: S3 バケットの中身を確認する (aws s3 ls)

aws s3 ls でバケットの中身を確認できる。ls コマンドの素直な拡張といえる（「2.14 ls」(p.23) 節も参照）

書式

```
$ aws s3 ls バケットのURL
```

実行例

```
$ aws s3 ls s3://testBucket
```

実行結果

```
$ aws s3 ls s3://testBucket
hello.py
```

バケットにファイル hello.py があると表示されている

♣ 2.1.5　実行例: ファイルを S3 に送る (aws s3 cp)

EC2 上の www.py を S3 バケットにコピーする

書式

```
$ aws s3 cp ファイル名 バケットのURL
```

実行例

```
$ aws s3 cp ファイル名 s3://testBucket
```

実行結果

```
$ aws s3 cp www.py s3://testBucket
```

無事にコピーされたかどうか？ は、aws s3 ls コマンドを再度実行してみるか、AWS S3 の管理画面で確認する

```
$ aws s3 ls s3://testBucket
hello.py   www.py
```

♣ 2.1.6　実行例: S3 からファイルを取り出す (aws s3 cp)

書式

```
$ aws s3 cp s3://testBucket/ファイル名 .
$ aws s3 cp s3://testBucket/ファイル名 ファイル名
$ aws s3 cp s3://testBucket/ファイル名 ディレクトリ名
```

実行例

```
$ aws s3 cp s3://testBucket/hello.py hello.py
```

S3 バケットにある hello.py を EC2 内に hello.py という名前でコピーする

実行結果

```
$ aws s3 cp s3://testBucket/hello.py hello.py
$ ls
hello.py   www.py   htdocs
```

EC2 上に hello.py ファイルが増えたことが分かる

2.2　cat

ファイルを表示する**だけ**のコマンド。教科書やマニュアルでは「ファイルを表示する」際に **cat** コマンドを紹介しがちだが、実際の現場では、何百行もあるファイルに対して **cat** を使ってしまい、目的の行を見ることが出来ず、後悔したあげくに **less** を使うことがシバシバである。そういうわけなので、「ターミナルでは **cat** より **less**（「2.12 less」(p.22) 節を参照）を使う」と覚えておくほうが汎用性がある。

♣ 2.2.1　書式

```
$ cat ファイル名 [ファイル名...]
```

- 引数に、複数のファイル名を記述しても良い。複数の場合はコンテンツが連結 (concatenate) した状態で表示される。ちなみに、これ (concatenate) がコマンド名の由来である

♣ 2.2.2　実行例： www.py ファイルを表示する

```
$ cat www.py
```

www.py は数百行あるファイルなので、どんどん表示が流れていく。実質、最後の数十行だけが表示される（「行数」は各自のターミナルによる）。

2.3 chmod

ファイルやディレクトリの**パーミッション（アクセス権限）**を変更することができる。これにより、ファイルやディレクトリに対する**読み取り、書き込み、実行**などの権限を設定することができる。

♣ 2.3.1　書式

```
$ chmod [オプション] モード ファイル...
```

- 「モード」部分には、数値を用いた絶対値指定と、シンボルによる指定がある。
  - 数値指定では、以下の値を足し合わせた 8 進数を用いる。表 2.1 を参照
  - シンボルは、表 2.2 と表 2.3 を参照
- 「ファイル...」部分には、ファイル、ディレクトリ、複数のファイルやディレクトリを指定できる。

たとえば、所有者に読み込み・書き込み・実行を許可し、グループのメンバに読み込み・実行を許可し、 他者[*1]に読み込み・実行を許可する絶対値指定のモードは、755 (400+200+100+040+010+004+001) となる。

who シンボルの'u','g','o' はそれぞれユーザ、グループ、他者に相当する。'a' シンボルは'ugo' を指定した場合と同じになる。表 2.2 を参照

perm シンボルはモードの各ビットを表 2.3 のように表現する。

♣ 2.3.2　実行例

アクセス権限が 0664(rw-rw-r--) の text.txt に 0644(rw-r--r--) を設定する。

```
$ ls -l ←このコマンドでファイルの権限を確認できる
-rw-rw-r-- 1 admin205 group205 63 Sep 10 16:07 test.txt

$ chmod 644 text.txt
もしくは
$ chmod u=rw,g=r,o=r text.txt
```

[*1] 他者 (others) とは、ユーザつまりファイルのオーナー（所有者）でもなく、ファイルのグループにも属していないユーザ

[*2] マニュアルに searchable とあるので直訳だが、実際に何を意味するのか？ を正確に説明することは難しい。とりあえず、ディレクトリの中を見る/に移動するには **+x** が必要と覚えておいてほしい

▼表 2.1: 絶対値指定

値	説明
400	所有者の読み込みを許可。
200	所有者の書き込みを許可。
100	ファイルの場合、所有者による実行を許可。
	ディレクトリの場合、所有者による検索[*2]および（ディレクトリ内への）移動を許可。
040	グループのメンバからの読み込みを許可。
020	グループのメンバからの書き込みを許可。
010	ファイルの場合、グループのメンバによる実行を許可。
	ディレクトリの場合、グループのメンバによる検索および（ディレクトリ内への）移動を許可。
004	他者からの読み込みを許可。
002	他者からの書き込みを許可。
001	ファイルの場合、他者による実行を許可。
	ディレクトリの場合、他者による検索および（ディレクトリ内への）移動を許可。

▼表 2.2: シンボル指定 (who):操作対象を指定する

だれ	説明
u	ファイルの所有者が対象
g	ファイルのグループに属しているユーザが対象
o	上記２つに該当しないユーザ、他者が対象

▼表 2.3: シンボル指定 (perm):（who で指定した対象に対する）権限の追加・削除

権限	説明
r	読み込み許可ビット
w	書き込み許可ビット
x	実行もしくは検索/移動許可ビット

```
$ ls -l
```

実行結果

```
-rw-rw-r-- 1 admin205 group205 63 Sep 10 16:07 test.txt   ←変更前
-rw-r--r-- 1 admin205 group205 63 Sep 10 16:07 test.txt   ←変更後
```

解説

アクセス権限が 0664(rw-rw-r--) の text.txt が、0644(rw-r--r--) に変更された。「所有者 (admin205)」と「他者」の権限に変更は無いが、グループ所属のユーザは

text.txt に書きこめなくなった。

♣ 2.3.3　知っていると便利なオプション: -v もしくは --verbose[*3]

処理されたファイルごとに処理結果を出力する

実行例

```
$ chmod -v 614 text.txt
```

実行結果

```
$ chmod -v 614 text.txt
'test.txt' モードを 0644 (rw-r--r--) から 0614 (rw---xr--)へ変更しました
```

-v がない場合は何も表示されない

2.4　cp

ファイルをコピーする

♣ 2.4.1　書式

```
$ cp コピー元 コピー先
```

♣ 2.4.2　実行例

以下の実行例では、**tree** コマンドで現階層以下のファイルやディレクトリ状況を表示する。この **tree** コマンドとは、ディレクトリ内のファイル・ディレクトリをツリー形式で表示するコマンドである（詳細は「2.34 tree」(p.52) を参照）

実行例 1

いま作業しているディレクトリ以下には、次のように htdocs フォルダと test.txt ファイルがある。

```
$ tree
.
├──htdocs
└──test.txt
```

[*3] Linux と Windows のみ。正確には GNU coreutils の cp をインストールしている場合に利用できるオプション。**Unix 伝統の chmod コマンドに -v オプションは無い**

コピーを実行する

```
$ cp test.txt test2.txt
```

実行結果 1

```
$ cp test.txt test2.txt
$ tree
.
├──htdocs
├──test2.txt
└──test.txt
```

作成した test2.txt ファイルと test.txt ファイルの中身は同じである。

実行例 2

引数を次のようにすれば、htdocs 内に test2.txt をコピーできる。

```
$ cp test.txt htdocs/test2.txt
```

実行結果 2

```
$ cp test.txt htdocs/test2.txt

$ tree
.
├──htdocs
│   └──test2.txt
└──test.txt
```

2.5 curl

HTTP リクエストを利用してデータを取得する。「ブラウザの代わり」や「ファイルをダウンロードする」といった用途で利用できる。

♣ 2.5.1　書式

```
$ curl [オプション] URL
```

♣ 2.5.2　実行例

https://api.fml.org/dist/lsform.html にアクセスする

実行例

```
$ curl https://api.fml.org/dist/lsform.html
```

実行結果

```
$ curl https://api.fml.org/dist/lsform.html
<P>SHOPPING CART
<form method="POST" action="https://api.fml.org/api/lsform/v1">
        <P>item-01
                <input name="item-01"  type="text">
        <P>item-02
                <input name="item-02"  type="text">
        <P>item-03
                <input name="item-03"  type="text">
        <P>
        <input type="submit" value="buy">
</form>
```

出力の読み方：URL(https://api.fml.org/dist/lsform.html) で指定した lsform.html ファイルを取得し、html ファイルの中身を出力している。

curl を引数なしで実行するとファイルを作成しない

curl を引数なしで実行した場合、ファイルは作成しない。ダウンロードしたコンテンツを、画面に表示するだけであることに注意。

ファイルとして保存したいなら、次の-O もしくは-o オプションを使うこと

♣ 2.5.3　知っていると便利なオプション: -o, -O

-o (小文字 o、英小文字のオー、Oscar の O)

出力結果をファイルに保存する。

実行例

```
$ curl -o 保存するファイル名 URL
```

実行結果 （https://api.fml.org/dist/www.py の中身を 3w.py として保存する場合）

```
$ curl -o 3w.py https://api.fml.org/dist/www.py
        % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                       Dload  Upload   Total   Spent    Left  Speed
100  9698  100  9698    0     0  99644      0 --:--:-- --:--:-- --:--:-- 99979

$ ls  ←ディレクトリ内のファイルを確認
3w.py
```

-O (大文字 O、英大文字のオー、Oscar の O)

出力結果をリモートファイルと同じ名前で保存する。

実行例

```
$ curl -O URL
```

実行結果 （https://api.fml.org/dist/www.py を www.py として保存する場合）

```
$ curl -O https://api.fml.org/dist/www.py
        % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                       Dload  Upload   Total   Spent    Left  Speed
100  9698  100  9698    0     0  99644      0 --:--:-- --:--:-- --:--:-- 99979

$ ls  ←ディレクトリ内のファイルを確認
www.py
```

2.6　df

Unix のストレージ利用状況を確認できる。

♣ 2.6.1　書式

```
$ df
```

♣ 2.6.2　実行例

実行例

```
$ df
```

実行結果

```
Filesystem     1K-blocks    Used Available Use% Mounted on
udev              485284       0    485284   0% /dev
tmpfs              99332     488     98844   1% /run
/dev/xvda1       8025124 1944152   5651764  26% /
tmpfs             496660       0    496660   0% /dev/shm
tmpfs               5120       0      5120   0% /run/lock
/dev/xvda15       126678   11840    114838  10% /boot/efi
tmpfs              99332       0     99332   0% /run/user/1000
```

読み方は表 2.4 のとおり

♣ 2.6.3　知っていると便利なオプション: -m

MB 単位で出力する (1MB=1024KB)[*4]。

[*4] いまどきのストレージは大きいので MB どころか GB や TB 単位で表示してほしいと思うかもし

▼表 2.4: df の読み方

df の一行目	説明
Filesystem	各ファイルシステム
	（OS が割り当てた「デバイスの識別子」もしくは「ファイルシスシステムの種類」）
1K-blocks	使用できるディスク容量、基本単位は KB である
	例：（/の）/dev/xvda1 ファイルシステムの容量は 7838MB である
Used	使用中のディスク容量
Available	ディスクの空き容量
Use%	ディスクの使用率
Mounted on	ファイルシステムがどのディレクトリで使えるか
	（Unix 用語では「ファイルシステムのマウント先ディレクトリ」）
	例：/dev/xvda1 は/（ルートディレクトリ）にマウントされている

実行例

```
$ df -m
```

実行結果

```
Filesystem     1M-blocks  Used Available Use% Mounted on
udev                 474     0       474   0% /dev
tmpfs                 98     1        97   1% /run
/dev/xvda1          7838  1899      5520  26% /
tmpfs                486     0       486   0% /dev/shm
tmpfs                  5     0         5   0% /run/lock
/dev/xvda15          124    12       113  10% /boot/efi
tmpfs                 98     0        98   0% /run/user/1000
```

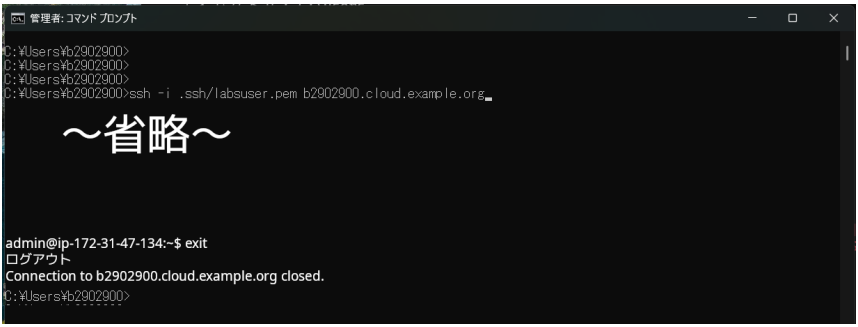
たとえば 2 行目の udev は、`485284KB/1024 = 474`（`473.9` を繰り上げ）なので、MB に変換された値が表示されている。

ストレージや IO の基本単位

df に限らずストレージや IO 統計のツールでは、表示の単位に気をつけたい。歴史的にストレージの基本単位は 1 セクタ 512 バイトであるが、ツールや OS ごとに表示単位はマチマチなので注意が必要だ。

セクタ数表示なのか？ バイト表示なのか？ 1000 で割った数字なのか？ それとも 1024 で割った数字なのか？ など気をつける点がいろいろある。正解はマニュアルを読めとしか言えないのだが...

れない。KB と MB の表示は良いのだが、それより大きい単位は Linux(GNU) 系と BSD でオプションが異なる。



▲図 2.1: Windows のコマンドプロンプトで AWS EC2 に ssh し、作業後に exit して Windows に戻ってきた様子

2.7　exit

ssh や sudo コマンドでログインしたときに、ログイン先から**ログアウト**できる（図 2.1）。Windows などのターミナルアプリ（例：tetaterm）上で exit コマンドを実行した場合、そのターミナルアプリごと閉じてしまうことに注意（注：図 2.1 は閉じる一歩手前の状態。さらに exit を入力すれば、このウインドウごと閉じてしまう）

♣ 2.7.1　書式

```
$ exit
```

♣ 2.7.2　実行例

以下、Windows のターミナルアプリ上で ssh コマンドを実行し、EC2 上に作成した Debian のサーバに SSH ログインしている前提とする

```
Windowsのターミナルアプリ ---(SSH)---> Debian GNU/Linux (EC2)
```

実行例 1 （EC2 からログアウトする）

```
admin@16.32.64.128$ exit
```

Debian 上で **exit** コマンドを実行する

実行結果 1

```
admin@16.32.64.128$ exit
Script done.
Connection to 16.32.64.128 closed.
C:\Users\b2902900>
```

ssh 先からログアウトして、Windows のターミナルに戻る。出力例の最終行 `C:\Users\b2902900>` は SSH 元の PC のターミナルに戻ってきた様子。

実行例 2 （Windows のターミナルアプリからログアウトする）

```
C:\Users\b2902900> exit
```

（実行例 1 の続きで）いまは Windows のターミナルに戻っている。その上で、もういちど exit を実行すると

実行結果 2

ターミナルアプリごと閉じてしまう

2.8 host

コマンドの引数を検索し、その引数に紐づくドメイン名や IP アドレスの情報を出力する。よく使う操作に次のものがある。

- 正引き: 引数がドメイン名 -> IP アドレスを出力
- 逆引き: 引数が IP アドレス -> ドメイン名を出力

書式

```
$ host 名前
```

「名前」の部分にはドメイン名や IP アドレスを指定する
以下に「正引き」と「逆引き」の実行例を示す。

実行例 （正引き：ドメイン名を指定する）

```
$ host ドメイン名
```

実行結果 （例：host portal.net.fml.org と入力した場合）

```
$ host portal.net.fml.org
portal.net.fml.org has address 210.128.53.254
```

出力 (portal.net.fml.org has ...) 右端の IP アドレス（210.128.53.254）部分が、回答（引数のドメイン名に紐づく IP アドレス）である。

実行例 （逆引き：ip アドレスを指定する）

```
$ host IPアドレス
```

実行結果 (例：host 210.128.53.254 と入力した場合)

```
$ host 210.128.53.254
254.53.128.210.in-addr.arpa domain name pointer portal.net.fml.org.
```

出力 (254.53.128.210.　...) 行の右端、domain name pointer の右に続く `portal.net.fml.org` という文字列部分が回答である。引数（210.128.53.254）に紐づくドメイン名になっている。

host, dig, nslookup コマンド

host と似たコマンドとして、dig と nslookup も存在する。これら 3 つのうち、どれがデフォルトで使えるか？ は OS ごとにマチマチなので注意しよう。もちろんインストールすれば、どの環境でも使用できる。

ちなみに、Windows の場合は、nslookup コマンドが標準搭載されている。

2.9 hostname

このコマンドを実行したコンピュータのホスト名を表示する[*5]。SSH 接続している場合、接続先のホスト名が表示されることに注意。また、Unix 系の OS (Debian や Ubuntu など) ではホスト名の確認に加えて、ホスト名を設定することができる。以下に、ホスト名の確認と設定の実行例を示す。

書式

```
$ hostname
```

実行例 (ホスト名の確認)

[*5] **ホスト名**とは、各機器に便宜上つける名前だ。たいてい、サーバの役割が分かる名前をつける。例：メールサーバは mail、ファイルサーバは fs

```
$ hostname
```

実行結果 (ホスト名の確認)

```
$ hostname
ip-172-31-9-130
```

この場合、「ip-172-31-9-130」がホスト名となる（注： Windows はこのホスト名の確認のみ行うことができる)。

実行例 (ホスト名の設定)（注：Unix 系 OS 限定・Windows は不可）[*6]

```
$ hostname <設定したいホスト名>
```

実行結果 (ホスト名の設定)

```
$ hostname debian.example.co.jp
```

```
$ hostname
debian.example.co.jp
```

ホスト名の設定を行う場合、実行しても設定結果が出力されるわけではないため、改めて hostname コマンドなどを実行し確認する必要がある。ただし、ホスト名の設定には管理者 (root) 権限が必要となるため、多くの場合、hostname コマンドの前に sudo コマンドをつけなくてはいけない。また、hostname コマンドで設定したホスト名は恒久的には変更されず、再起動すると設定前のホスト名に変更される。

2.10 id

ユーザ ID やユーザ名、グループ ID、グループ名などの識別情報を出力する。自分（自分のユーザ名）の役割や権限を確認するために利用する。

書式

```
$ id
$ id ユーザ名
```

[*6] hostname コマンドでホスト名をつける作業は通常おこなわない。OS の各種設定は/etc 以下の設定ファイルを適切に編集し、再起動、設定が反映されていることを確認するべきである

「ユーザ名」を指定しない場合は、このコマンドを実行している「ユーザ名」の情報が表示される。たとえば、AWS の Debian イメージであれば、通常あなたの使うユーザは admin である。よって

```
$ id
```

も

```
$ id admin
```

も同じように admin ユーザの情報を表示する

♣ 2.10.1　実行例

実行例

```
$ id admin
```

実行結果 （ユーザ admin の情報を出力する場合）

```
$ id admin
uid=1000(admin) gid=1000(admin) groups=1000(admin),4(adm),20(dialout),24(cdrom),25(floppy)›
›,27(sudo),29(audio),30(dip),44(video),46(plugdev)
```

ユーザ名 admin のユーザ ID や所属グループがわかる

- uid=1000(admin): ユーザ名が admin、ユーザ ID が 1000
- gid=1000(admin): グループ名が admin、グループ ID が 1000
- groups=1000(admin),4(adm),20(dialout): ユーザ admin は上述のデフォルトグループ admin に加えて adm と dialout グループにも所属している

実行結果 （自分のユーザ情報を出力する場合）

```
$ id
uid=1000(admin) gid=1000(admin) groups=1000(admin),4(adm),20(dialout),24(cdrom),25(floppy)›
›,27(sudo),29(audio),30(dip),44(video),46(plugdev)
```

出力がユーザ admin の場合と同じであることに注意してほしい。

実行結果 （ユーザ名が root のユーザ情報を出力する場合）

```
$ id root
uid=0(root) gid=0(root) groups=0(root)
```

実行結果 （管理者権限 (sudo) を利用し、自分のユーザ情報を出力する場合）

```
$ sudo id
uid=0(root) gid=0(root) groups=0(root)
```

出力の読み方：`sudo id` は **id** コマンドを管理者権限で実行することになるので、`id root` を実行したときと同じ出力になる

♣ 2.10.2　知っていると便利なオプション: -u

ユーザ ID のみを出力する[*7]。

実行例

```
$ id -u ユーザ名
```

実行結果 （ユーザ名が admin のユーザ ID を出力する場合）

```
$ id -u admin
1000
```

ユーザ名 admin のユーザ ID が 1000 であることが分かる

2.11　ip

ネットワークデバイスやルーティングなどの表示や変更ができる。演習で使うのは `ip a` と `ip r` だけだろうから、これくらいは覚えておきたい

♣ 2.11.1　書式

```
$ ip [ options ] OBJECT COMMAND
$ ip [ options ] OBJECT help
```

- OBJECT 部分には、**address**（IP アドレス）や **route**（経路情報）など、ネットワークの構成要素を指定する。
- COMMAND 部分では、その OBJECT ごとのコマンドが利用できる。
- COMMAND を指定しない場合は、たいてい情報の表示が行われる。たとえば、`ip address` は `ip address show` と同じ意味に解釈され、現在の IP アドレス情報が表示される

[*7] このオプションをターミナル上で利用することは少ないが、シェルスクリプトや Dockerfile を書く際には必要な知識である

- もし COMMAND が分からない場合は、OBJECT に関わらず help コマンドを指定すればヘルプメッセージが表示される

♣ 2.11.2　実行例: IP アドレス情報を調べる （ip address）

address オブジェクト（省略して a も可）を用いて、ネットワークインターフェース[*8]に関連する IP アドレス情報を確認できる。

実行例

```
$ ip address
$ ip a
```

どちらも同じ結果が出力される。

実行結果

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 100
0
    〜省略〜
2: enp4s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9001 qdisc mq state UP group default qlen
 1000
    link/ether 0a:16:41:0c:9e:43 brd ff:ff:ff:ff:ff:ff
    inet 172.31.13.152/20 brd 172.31.15.255 scope global ens5
       valid_lft forever preferred_lft forever
    inet6 fe80::816:41ff:fe0c:9e43/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group de
fault
    〜省略〜
```

最重要の確認項目は次の 2 つ。

- inet 行: ipv4 アドレス/プレフィックス（のビット数）が表示されている
  - 上の例では、172.31.13.152/20
- link/ether 行: MAC アドレス[*9]が表示されている
  - 上の例では、0a:16:41:0c:9e:43

左端の数字 (1:, 2:, ...) に特別な意味はない。`2: enp4s0:`の `enp4s0` 部分はネットワークインターフェイス名である。この名前は OS が割り当てるため、OS やハードウエア構成によって異なる値を取ることに注意。あくまでも enp4s0 は一例である。一昔前まで Linux では eth0 のような名前の付け方をしていたが、最近、enp4s0 のようなハードウエア情報に基づく名前の付け方に変わった[*10]

[*8] データを送受信するための接続ポイント
[*9] 各ネットワークインターフェースに割り当てられた一意の物理アドレス
[*10] 最初から BSD Unix に合わせておけばよかったのに！ と思ったあなたは同志です:-)

♣ 2.11.3　実行例: IP アドレス情報を調べる （ip route）

route オブジェクト（省略して r も可）を指定すると、ルーティング（ネットワークの経路）情報を出力する。

実行例

```
$ ip route
$ ip r
```

どちらも同じ結果が出力される。

実行結果

```
default via 210.128.53.201 dev enp3s8 onlink
210.128.53.200/29 dev enp3s8 proto kernel scope link src 210.128.53.204
210.128.53.208/29 dev enp1s0 proto kernel scope link src 210.128.53.209
210.128.53.216/29 via 210.128.53.212 dev enp1s0
210.128.53.224/29 via 210.128.53.212 dev enp1s0
210.128.53.232/29 via 210.128.53.212 dev enp1s0
210.128.53.248/29 dev enp2s0 proto kernel scope link src 210.128.53.249
```

- 1 行目の default 行はデフォルトルートの情報
  - デフォルトルートとは、宛先がマッチしない場合の転送先ルータ。「マッチしない」とは 2 行目以降のネットワークに該当するものがないこと
  - `default via 210.128.53.201` の `210.128.53.201` が転送先
  - dev enp3s8 は 210.128.53.201 が存在するネットワークのインターフェイス（デバイス）名。一つの PC にネットワークインターフェイスは複数ありうるので、それを区別するための情報。この enp3s8 のような名前は `ip address show` で表示されていたことを思い出せ
- 2 行目以降はネットワーク（IP アドレス/プレフィックス）ごとの情報
  - 例えば 2 行目の `210.128.53.200/29` は、`210.128.53.200/29` というアドレス範囲への通信のルーティング情報
  - dev enp3s8 の enp3s8 は、`210.128.53.200/29` が存在するネットワークのインターフェイス（デバイス）名。
  - src 210.128.53.204 の IP アドレスはネットワークインターフェイス（デバイス）の IP アドレス。 つまり enp3s8 に付いている IP アドレスと同じ

ip コマンドによる設定変更は原則おこなわない

ip コマンドによる設定変更も可能だが、これは障害対応の場合にのみ許される作業と考えておいたほうがよい。

通常、ネットワークの設定は/etc 以下の設定ファイルにもとづき OS 起動時

に行われる。ネットワーク関連の変更は/etc 以下を適切に変更後、再起動して OS に設定してもらうべきである

2.12 less

ファイルを表示するコマンドだが、画面単位で行ったり来たり、検索なども出来るところが便利。**vi** エディタのコマンドモードだけが使える状態と考えて良い。

教科書やマニュアルでは「ファイルを表示する」際に **cat** コマンドを紹介しがちだが、実際の現場では、何百行もあるファイルに対して **cat** を使ってしまい、目的の行を見ることが出来ず、後悔したあげくに **less** を使うことがシバシバである。**cat**「2.2 cat」(p.6) 節と **vi**「2.36 vi」(p.55) 節も参照。

「ファイルを見るときにエディタを使う」方法は良くない。うっかり編集してしまうかもしれないから。一つの操作/機能は一つの専用コマンドで行うべきである。この先祖伝来の Unix コマンド群の設計思想[*11] を覚えておいてほしい

♣ 2.12.1　書式

```
$ less ファイル名
```

♣ 2.12.2　実行例： www.py ファイルを表示する

```
$ less www.py
```

- **q** を叩くと終了する (もちろん quit の q)
- スペースを叩くと次の画面に変わる（スクロール）
- **b** を叩くと一つ前の画面に戻る (back の b)
- /を叩くと検索モードに入ることが出来る
  - /は画面の下に向かって検索を行う
  - **?**は反対で、画面の上に向かう検索が出来る
- **h** を叩けば、ヘルプ画面が表示される (もちろん help の h)
- **v** を叩くと、いま見ているファイルに対して **vi** エディタを起動する
  - つまり必要なら編集モードへ移行できるわけである

[*11] 最近のコマンドは、そうでもないという非肉が込められている文章:-)

more と less、less がないなら more も試してみるとよい

このような操作には元々 **more** コマンドがあったが、その機能強化版として **less** が開発された (more の反対語なのに強化版という名付け方が御洒落:-)。

いずれにせよ、more のほうが古いので less コマンドがない場合は more を試してみるとよい。たいていは、あるはずだ。

2.13　logout

exit コマンドと同様に、SSH 先の Linux/Unix からログアウトできる。
「2.7 exit」(p.14) 節も参照のこと。

♣ 2.13.1　書式

```
$ logout
```

2.14　ls

いま作業している場所（ディレクトリ）内のディレクトリやファイルの情報を表示する

♣ 2.14.1　書式

```
$ ls [option] [FILE]...
```

- 引数がない場合、現在のディレクトリの情報を表示する
- 引数のファイル（群）やディレクトリ（群）の個数は可変長
- 引数のファイルやディレクトリは混在してよい

♣ 2.14.2　実行例

実行例

```
$ ls
```

実行結果

```
$ ls
htdocs python.py test.txt
```

このディレクトリには htdocs python.py test.txt がある。

♣ 2.14.3　知っていると便利なオプション: -l (エル,lima の l)

数字の 1 ではなく英語小文字のエル、lima の l。ファイル・ディレクトリの権限が確認できる。権限の読み方は「2.3 chmod」(p.7) 節を参照

実行例

```
$ ls -l
```

実行結果

```
$ ls -l
drwxr-xr-x 2 user group  4096 Sep 08:14:44 htdocs
-rw-r--r-- 1 user group 10239 Dec 08 15:36 python.py
-rw-r--r-- 1 user group 10239 Sep 08 15:38 test.txt
```

- **d,r,w,x,-**という文字列の意味は、それぞれ以下のとおり
  - 左端の **d** はディレクトリを意味する。**-**はファイルである
  - **r** は読み取り権限、**w** は書き込み権限である。**x** はファイルとディレクトリで意味が異なる。ファイルの場合は実行権限つまり「プログラムとして実行できる」という意味。ディレクトリの場合、そのディレクトリの中を見たり/中へ移動できるという意味[*12]
- 1 行目の `htdocs` は（左端が d なので）ディレクトリ
  - ユーザは **rwx**、グループ (グループ名:group に属しているユーザ) の権限は **r-x**、他者[*13]の権限も **r-x** である
  - **r** と **x** は共通なので、**全ユーザがディレクトリの中を見ることが出来て、かつディレクトリ内に移動も可能**
  - **w** があるのはユーザだけなので、**ユーザ本人のみがディレクトリに書き込める（htdocs 内に新たにファイルやディレクトリを作成できる）**

[*12] 原語の searchable(検索〜) も不明瞭。正確な説明は難しい（「2.3 chmod」節も参照）

[*13] 他者 (others) とは、ユーザつまりファイルのオーナー（所有者）でもなく、ファイルのグループにも属していないユーザ

- 2 行目と 3 行目はファイル
  - python.py と test.txt は左端が**-**となっているのでファイル
  - ユーザの権限は **rw-**なので、ユーザ（user）は**読み書き可能**
  - group に属するユーザは **r--**なので**読み取りのみ可能**
  - 他者も同様に **r--**なので**読み取りのみ可能**

♣ 2.14.4　知っていると便利なオプション: -a

隠しファイル・隠しディレクトリを含めた全てのファイルを表示する。

実行例

```
$ ls -a
```

実行結果

```
$ ls -a
.ssh .bash_history test.txt htdocs python.py
```

隠しディレクトリである.ssh と隠しファイルの.bash_history も表示されている。

2.15　lsblk

利用可能なブロックデバイスをツリー状に一覧表示する[*14]。

♣ 2.15.1　書式

```
$ lsblk
$ lsblk [options] [device...]
```

- 引数がない場合、全ブロックデバイスの情報を表示する
- 引数で特定のデバイスを指定することも出来る

♣ 2.15.2　実行例

実行例

```
$ lsblk
```

[*14] ブロックデバイスとは、512 バイトなどのまとまったデータ量 (ブロック) を単位として読み書きする HDD や SSD などの記憶装置のこと。

実行結果

```
NAME      MAJ:MIN RM  SIZE RO TYPE MOUNTPOINTS
xvda      202:0    0    8G  0 disk
├─xvda1   202:1    0  7.9G  0 part /
├─xvda14  202:14   0    3M  0 part
└─xvda15  202:15   0  124M  0 part /boot/efi
```

実用上は、左端のデバイス名と右端のマウントポイントだけ見ていれば十分だろう。詳細は表 2.5 を参照

▼表 2.5: lsblk の表示

1 行目	説明
NAME	デバイス名（OS が自動的に割り当てている）
MAJ:MIN	MAJ はメジャーデバイス番号。どのデバイスドライバを使うべきかを示す情報
	MIN はマイナーデバイス番号で、同一ドライバを使用するデバイスを区別する番号
	(例：ここに表示されている xvda はすべて同じデバイスドライバを使用している)
RM	リムーバルディスクか否か（OS 動作中に取り外し可能であれば 1）
SIZE	デバイスの容量
RO	読み取り専用かどうかを示している。読み取り専用であれば 1 である。
TYPE	デバイスのタイプ
	disk = 物理ディスク、part = パーティション (ディスクを論理分割したもの)
MOUNTPOINTS	ファイルシステムがマウントされている場合、そのパスが表示される

2.16 lscpu

CPU の情報を表示する

♣ 2.16.1　書式

```
$ lscpu [options]
```

たいていは引数なしで実行すれば十分

♣ 2.16.2　実行例

実行例

```
$ lscpu
```

実行結果

```
Architecture:             x86_64
  CPU op-mode(s):         32-bit, 64-bit
  Address sizes:          46 bits physical, 48 bits virtual
  Byte Order:             Little Endian
CPU(s):                   1
  On-line CPU(s) list:    0
Vendor ID:                GenuineIntel
  Model name:             Intel(R) Xeon(R) CPU E5-2686 v4 @ 2.30GHz
    CPU family:           6
    Model:                79
    Thread(s) per core:   1
   〜省略〜
Virtualization features:
  Hypervisor vendor:      Xen
  Virtualization type:    full
   〜省略〜
```

そもそも `lscpu` コマンドが実行できる時点で、正常に OS が起動できているはずなので、障害対応の意味合いは薄い。実運用上は「CPU のコア数の確認」や「CPU の型番を知りたい」という場面くらいか？ 表 2.6 も参照のこと。

▼表 2.6: lscpu の読み方 (一部)

項目 (一部)	説明
Architecture	CPU のアーキテクチャ
CPU op	CPU がサポートしてるモード
	(例：64bit,32bit アーキテクチャをサポートしている)
Address sizes	物理アドレスと仮想アドレスのサイズ
	(例：物理アドレス 46bit, 仮想アドレス 48bit)
Byte Order	バイトオーダ (例：リトルエンディアンという最下位バイトが最初に来る形式)
CPU(s)	システムに搭載されてる CPU の数
On-line CPU(s) list	現在使用可能な CPU のリスト (例：0 番目の CPU1 つのみが使用可能)

2.17　lsmem

(オンライン状態になっている) メインメモリの利用可能な範囲を表示する。

♣ 2.17.1　書式

```
$ lsmem [options]
```

たいていはオプション無しで実行して良い

♣ 2.17.2　実行例

実行例

```
$ lsmem
```

実行結果

```
RANGE                                 SIZE  STATE REMOVABLE BLOCK
0x0000000000000000-0x000000003fffffff   1G online       yes   0-7

Memory block size:       128M
Total online memory:       1G
Total offline memory:      0B
```

実用上はサイズの確認くらいだろう。表 2.7 も参照のこと。

▼表 2.7: lsmem の読み方

項目	説明
RANGE	メモリのアドレス範囲 (この例では 1GB を示している)
Memory block size	1 ブロック当たりのサイズ (1GB/8=128MB)
Total online memory	使用可能な (オンラインの) メモリサイズ (例：1GB)
Total offline memory	使用不可能な（現在オフライン[*15]の）メモリサイズ

2.18　man

Unix マニュアルを見るコマンド。ただし、初心者向きではない。Unix マニュアルの記述は網羅的だが、いちから読むというより、必要なときにピンポイントでコマンドの使い方の確認に使うことが多いと思う。

♣ 2.18.1　書式

```
$ man コマンド名
```

♣ 2.18.2　実行例： ls コマンドのマニュアルを見る

[*15] オフラインとは？ メモリをブロック単位でオンライン/オフラインを設定できるのだが、超巨大な仮想基盤でもないかぎり使う機会のない機能に思える。詳細は https://www.kernel.org/pub/linux/utils/util-linux/の chmem のマニュアルとソースコードを参照したほうがよい

```
$ man ls
```

- 完全な書式、すべてのオプションの説明等が表示される（ので刮目して読め:-）

蛇足：man があるのだから、このような本は...

本来、このような技術同人誌は不要なのですね... ;-)

2.19 mkdir

ディレクトリを作成する

♣ 2.19.1　書式

```
$  mkdir [option]... DIRECTORY...
```

- 引数に、作成するディレクトリを少なくとも 1 つは指定する必要がある
- 引数は可変長。作成したいディレクトリは、引数として複数つなげて書けば良い。Unix なので、いつものように**スペース区切り**である

♣ 2.19.2　実行例

前提

現状の確認は `ls` コマンドで行える。

```
$ ls
www.py htdocs/
```

いま作業しているディレクトリには、www.py というファイルと htdocs というディレクトリがある。

実行例 1

```
$ mkdir test
```

新たに `test` ディレクトリを作成し、結果を ls コマンドで確認する。

実行結果 1

```
$ mkdir test
$ ls
www.py htdocs/ test/
```

test という名前のディレクトリが追加されていることが分かる

実行例 2

```
$ mkdir test/dir
```

さらに、test ディレクトリの下に dir ディレクトリを作成する

実行結果 2

```
$ mkdir test/dir
$ ls test
dir/
```

test 内のファイル・ディレクトリを確認すると、dir という名前のディレクトリが追加されていることが分かる

2.20　nano

Windows のメモ帳のような**シンプルな操作方法**のエディタ。

ファイル名を指定することで、指定したファイルを編集することができる。ファイル指定をせずに起動した場合や、存在しないファイルを指定して起動した時は、起動後に、あらためて新規ファイルの作成・編集ができる。

Unix のエディタ

Unix に昔からあるテキストエディタとしては vi や emacs が有名だ。vi や vim[*16]には複数のモードがあるが、nano にはモード変更などない。nano は、モード変更を使用せず、画面下部に書かれているショートカットを用いることでコピーやペースト等を行う（この点 nano の使い勝手は emacs に近い）。

♣ 2.20.1　書式

[*16] vi クローン。オリジナルの vi を遥かに超えてより強化された vi 似のエディタ。今どきは vim が vi という名でインストールされていることが多いが、たまに 4.4BSD 由来の nvi もあるので注意

▲図 2.2: nano 画面下のヘルプメッセージ（ショートカットのチートシート）

▼表 2.8: nano コマンドのショートカットキー（抜粋）

キー	意味
Ctrl x	終了する
Ctrl s	上書き保存
Ctrl 6	選択範囲の開始
Alt 6	選択範囲の終了
Ctrl u	ペースト
Ctrl k	いまカーソルのある行をまるごと削除
BackSpace	一文字消去 (右から左へ)
DELETE	一文字消去 (左から右へ)

```
$ nano [options] ファイル名
```

♣ 2.20.2　実行例

実行例

```
$ nano ファイル名
```

実行結果

- ファイル名を指定して起動する
- そのファイルが存在しない場合は、ファイルを新規作成する
- その後エディタ画面になり、文字を入力できるようになる

♣ 2.20.3　ショートカットについて

上述のとおり、nano エディタを起動した際には、図にあるショートカットの説明が確認できる（図 2.2「nano 画面下のヘルプメッセージ（ショートカットのチートシート）」を参照）。なお、ショートカットにある**^**という文字は、Windows であれば ctrl、MacOSX であれば command と読み替えてほしい。

終了する （ctrl +x もしくは command +x）

nano エディタ画面から抜けるときに使用する。これを使用した際には、保存するかどうか聞かれるので、保存するときは y を、保存しないときは n を押す。y を押し

た場合、保存したいファイル名を聞かれるので、そのままの名前でよい場合は Enter を押す。ファイルを保存後、ターミナルに戻る。もし、違う名前で保存したい場合は、ファイル名を変更して Enter を押すことで保存される。n を押した場合、保存せずにターミナルの画面に戻る（編集内容は廃棄される）。

上書き保存する （ctrl +s もしくは command +s）

変更内容を上書き保存する場合に入力する。この場合、エディタは続行される。

選択範囲の開始 （ctrl +6 もしくは command +6）

後述するカット及びコピーを範囲選択する際の始点をカーソル位置で決定する。

選択範囲の終了 （alt +6 もしくは option +6）

選択の終点を指定する。上述の「選択範囲の開始」とペアの操作である。

選択範囲の削除 （ctrl +k もしくは command +k）

（上述の操作で）選択した範囲を削除する。

選択範囲のペースト （ctrl +u もしくは command +u）

（上述の操作で）選択した範囲をペーストする。

♣ 2.20.4　マウス操作

最近のアプリはウインドウシステムとの連携も意識しているので、CUI のエディタでも GUI 操作による入出力を相当うけつける。

nano では文字列をドラックして選択して右クリックをした後、コピーを選択することで、文字列のコピーができる。同様に貼り付ける場合は、ペーストしたい位置にカーソルを移動後、右クリックしてペーストを選択することで、文字列のペーストができる。

2.21　ping

ターゲットホストの生死を確認するコマンド。ホスト間のネットワーク接続の安定性を検証する用途で利用する。

1 秒に 1 つのデータグラムを送信し、受信した応答内容を出力する。出力は無限に続くので、キーボードから中断させる (Ctrl-C を入力する) ことが必要。終了時には、往復情報とパケット損失統計を計算し、サマリを出力する。

♣ 2.21.1　書式

```
$ ping [options] ターゲット
```

「ターゲット」はドメイン名か IP アドレス

♣ 2.21.2　実行例

実行例

```
$ ping IPアドレス
```

▼表 2.9: ping コマンドの出力の読み方

キーワード	説明
icmp_seq	ping を送信した回数
ttl	ルータを最大で何個まで通過できるか？
time	送受信の往復にようした時間

実行結果 （成功パターン:10.252.0.11 から応答があったとき）

```
$ ping 10.252.0.11
PING 10.252.0.11 (10.252.0.11) 56(84) bytes of data.
64 bytes from 10.252.0.11: icmp_seq=1 ttl=64 time=0.039 ms
64 bytes from 10.252.0.11: icmp_seq=2 ttl=64 time=0.042 ms
    〜中略〜
64 bytes from 10.252.0.11: icmp_seq=7 ttl=64 time=0.018 ms
^C                                    ← Ctrl-Cを打ち込んだ行
--- 10.252.0.11 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6151ms
rtt min/avg/max/mdev = 0.017/0.027/0.042/0.010 ms
```

最終行の「0% packet loss」（パケット損失が 0%）から回線品質が良好であることが伺える

実行結果 （失敗パターン:10.252.0.127 から応答がなかったとき）

```
$ ping 10.252.0.127
PING 10.252.0.127 (10.252.0.127) 56(84) bytes of data.
From 10.252.0.11 icmp_seq=1 Destination Host Unreachable
    〜中略〜
From 10.252.0.11 icmp_seq=6 Destination Host Unreachable
^C                                    ← Ctrl-Cを打ち込んだ行
--- 10.252.0.127 ping statistics ---
7 packets transmitted, 0 received, +6 errors, 100% packet loss, time 6152ms
pipe 4
```

最終行に「100% packet loss」とあることから、パケット損失が 100%、つまりターゲット (10.252.0.127) からの応答がないことが分かる。ただし、これだけでは、ター

ゲットが稼働していないのか、ネットワーク品質が悪いのか？ は不明。さらに調査する必要がある。

ping の由来

ping という単語は潜水艦のソナーに由来する。（アクティブ）ソナーとは「潜水艦から音波を出し、障害物から反射してくる音波を分析する」ことで水中の様子をさぐる仕組みのこと。水中でのレーダーと考えれば良い。ping コマンドは、それのサイバースペース版

2.22　ps

プロセスの情報を表示するコマンド。

♣ 2.22.1　書式

```
$ ps [options]
```

- 引数なしで実行しても、あまり有益な情報は得られない
- 典型的な使い方は ps auxww である[*17][*18]。この形で覚えることを推奨

♣ 2.22.2　実行例

実行例

```
$ ps
```

実行結果

```
    PID TTY          TIME CMD
1177754 pts/1    00:00:00 bash
1177761 pts/1    00:00:00 ps
```

[*17] 書式は歴史的に 2 系統ある。Linux 搭載の ps コマンドは両系統の上位互換に見える。本書では伝統の Unix 由来のオプションのみを紹介する

[*18] 古代 Unix との互換性を保つため、ps コマンドは、オプションに-をつけてもつけなくても受け付ける数少ないコマンドである。例：ps auxww と ps -auxww どちらでもよい

自分が実行しているプロセス群が表示されている。3 行目の ps は、いま実行中の ps コマンド自身。各項目の詳細は表 2.10 を参照。

ちなみに「TTY 列が同じ値（pts/1）のみ」であることを覚えておきつつ、次の a オプション以降を読んでほしい

▼表 2.10: ps コマンドの基本出力

１行目の各項目	説明
PID	プログラムごとに割り当てられている番号（プロセス ID の略）
TTY	プロセスが関連づけられているターミナル
STAT	プロセスの状態（a オプション指定時）
TIME	プロセスが使用した CPU 時間
CMD	実行されているコマンド

♣ 2.22.3　知っていると便利なオプション: a

他のターミナルを含めたプロセスを表示する

実行例

```
$ ps a
```

実行結果

```
    PID TTY      STAT   TIME COMMAND
    656 tty7     Ssl+ 777:24 /usr/lib/xorg/Xorg :0 -seat seat0 -
    657 tty1     Ss+    0:00 /sbin/agetty -o -p -- \u --noclear
1174896 pts/0    Ss+    0:00 -bash
1177754 pts/1    Ss     0:00 -bash
1177757 pts/1    R+     0:00 ps a
```

TTY 列を見ると値が複数あることが分かる。TTY 列は各ユーザに割り当てられたターミナルである。前述の出力例で分かるとおり、pts/1 の行が自分がいる**"ターミナル"**を意味している。その他の tty1、tty7、pts/0 は別のターミナルである。この例では pts/0 行が`-bash` なので、自分以外のユーザも同じサーバ上にログインして（＝シェルを起動していて）作業中であることが分かる。

STAT 列はプロセスの状態で、表 2.11 を組み合わせた値が表示される。

♣ 2.22.4　知っていると便利なオプション: u

プロセスの所有者や CPU・メモリの使用率をわかりやすく表示する。

実行例

▼表 2.11: ps コマンドの STAT 列の読み方

STAT 列の項目	説明
R	実行中 (CPU を今使っている)
S	待機中 (例：-bash はユーザからの入力を待っている)
s	セッションリーダー (s が付くプロセスが、s が付かないプロセスを呼び出している)
l[*19]	マルチスレッドプロセスであること（複雑なプロセスにつくことが多い）
+	（ターミナルに紐づいていて）ユーザが操作できる状態のプロセス

```
$ ps u
```

実行結果

```
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
h205     1174896  0.0  0.0   9456  7328 pts/0    Ss+  13:53   0:00 -bash
h205     1177754  0.0  0.0   6408  5020 pts/1    Ss   13:58   0:00 -bash
h205     1178343  100  0.0   9388  4864 pts/1    R+   14:00   0:00 ps u
```

列の詳細は表 2.12 を参照。特に注目する列は、(1)CPU 使用率を意味する %CPU 列と、(2) メモリ使用量を意味する RSS 列だろう。システムが重い時は、このどちらかの列の値が異常に大きいプロセスが犯人である可能性が高い。

▼表 2.12: ps u で追加表示される列

列	説明
USER	（右端のコマンドを実行している）ユーザ名
%CPU	プロセスの CPU の使用率
%MEM	プロセスの物理メモリの使用率
VSZ	仮想メモリを確保しているサイズ
RSS	物理メモリを確保しているサイズ

♣ 2.22.5　知っていると便利なオプション: x

ターミナルに紐づかないプロセス（サーバとしてターミナルを切り離した状態で動作するプロセスなど）も表示する。

実行例

[*19] （IT 古典部むけ？）マルチスレッドなのに、l（エル）なことに疑問を持つあなたはエンジニア向きである。これはスレッドに相当するカーネルの lightweight process という用語に由来する。

```
$ ps x
```

実行結果

```
    PID TTY      STAT   TIME COMMAND
    800 ?        Ss     0:01 /lib/systemd/systemd --user
    801 ?        S      0:00 (sd-pam)
    816 ?        S<sl   0:00 /usr/bin/pulseaudio --daemonize=no
    〜省略〜
1174895 ?        S      0:00 sshd: h205@pts/0
1174896 pts/0    Ss+    0:00 -bash
1177753 ?        S      0:00 sshd: h205@pts/1
1177754 pts/1    Ss     0:00 -bash
1178609 pts/1    R+     0:00 ps x
```

TTY が？ のものは、ターミナル（TTY）が紐付いていないサーバプロセスである。

♣ 2.22.6　知っていると便利なオプション: aux,auxww

これまで説明したオプションの a,u,x を組み合わせたもの。しかし、aux だと長いコマンドの場合に出力が省略されるため、コマンドライン引数をすべて確認したい場合は auxww とする

実行例 （ps aux と auxww の比較）

```
$ ps aux
$ ps auxww
```

実行結果 （ps aux と auxww の比較）

```
$ ps aux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.0  0.0 169460 13824 ?        Ss   14:56   0:03 /sbin/init
root           2  0.0  0.0      0     0 ?        S    14:56   0:00 [kthreadd]
    〜省略〜
root       27816  0.0  0.0 1238196 15296 ?       Sl   15:16   0:00 /usr/bin/hogehogehogeho›
›ge-v2 -namespace dummy -id c5ca6239
    〜省略〜
```

（上の例では省略されているが）基本的に全てのプロセスが表示されている。

```
$ ps auxww
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.0  0.0 169460 13824 ?        Ss   14:56   0:03 /sbin/init
root           2  0.0  0.0      0     0 ?        S    14:56   0:00 [kthreadd]
    〜省略〜
root       27816  0.0  0.0 1238196 15296 ?       Sl   15:16   0:00 /usr/bin/hogehogehogeho›
›ge-v2 -namespace dummy -id c5cadb62398fbbad3823e8e4eaf6003e0d68f1853fed4b4cf9ff1bfe6f39d02›
›70cac8 -address /hoge/hoge/hoge/hoge
    〜省略〜
```

PID 27816 の COMMAND 列を見比べると、ps aux の場合と異なり、ps auxww の場合はコマンドの末尾まで出力されていることがわかる。

2.23 pstree

プロセスの親子関係をツリー状にアスキーアートで表示する[*20]

♣ 2.23.1　書式

```
$ pstree [options]
```

♣ 2.23.2　実行例

実行例

```
$ pstree
```

実行結果

```
systemd─┬─2*[agetty]
        ├─dbus-daemon
        ├─polkitd─────2*[{polkitd}]
        ├─sshd─────sshd─────sshd─────bash─────script─────bash─────pstree  ← (1)
        ├─sudo─────bash─┬─bash
        │               └─inotifywait
        ├─systemd─────(sd-pam)
        ├─systemd-journal
        ├─systemd-logind
        ├─systemd-network
        ├─systemd-resolve
        ├─systemd-timesyn─────{systemd-timesyn}
        ├─systemd-udevd
        └─unattended-upgr
```

- (1) の行をみると、 bash(pstree を入力したターミナルに紐づいたシェル) が pstree を呼び出していることがわかる

2.24 pwd

自分が現在作業しているディレクトリを**絶対パスで表示**する。pwd は「Print Working Directory」もしくは「Present Working Directory」の略。

♣ 2.24.1　書式

[*20] Unix の場合、プロセスにも親子関係があり、プロセスもツリー状になっている（注：学部の OS の授業では、ここまで説明しない)。そのため、親プロセス、子プロセスという言い回しもある

```
$ pwd [options]
```

♣ 2.24.2　実行例

実行例

```
$ pwd
```

実行結果

```
$ pwd
/home/admin
```

現在自分（ユーザ admin）がいるディレクトリが表示されている様子。

2.25　python3

♣ 2.25.1　書式

```
$ python3
$ python3 ファイル名
```

- python3 と単体で入力した場合は対話モードとなり、コードを入力すると実行される。quit() と入力すれば、対話モードは終了する
- 引数があれば、その引数のファイルを 1 行ずつ実行していく（スクリプト言語、下記ノートを参照）

♣ 2.25.2　実行例

以下のように `print` 一行だけの hello.py ファイルがあるとする

```
$ cat hello.py
print("Hello World")
```

ちなみに、ファイルの中身を確認するときは cat コマンドを使うと表示できる、「2.2 cat」(p.6) 節も参照。

実行例 1

```
$ python3 hello.py
```

実行結果 1

```
$ python3 hello.py
Hello World
```

hello.py が実行され、結果が出力されている。

実行例 2

```
$ python3
〜省略〜
>>>
```

対話モードに入ったため、>>>と表示される。

実行結果 2

```
>>> print("Hello World")
Hello World

>>>quit()
$
```

- `print("Hello World")` を入力したあと ENTER を押すと、入力した行が実行され、結果が出力された様子
- `quit()` を入力し ENTER を押すと、対話モードが終了する。シェルに戻ってきたので`$` が表示されている。これで Unix コマンドが打てる状態に戻った

コンパイラ言語とスクリプト言語

プログラミング言語は大きく分けて二種類ある。コンパイラ言語とスクリプト言語だ。

コンパイラ言語の代表例は C 言語である。(a) まずソースコードを書き (b) それをコンパイラで実行形式 (`a.out`) に変換する。OS に実行してもらうのは `a.out` ファイルである。

一方、python はスクリプト言語である。www.py のようなプログラムは実行ファイルではなくソースコードにあたる。python の場合、実行ファイルは作成しない。python プログラムがソースコード (www.py) を一行ずつ読みながら逐次実行する。こういったプログラムをインタープリタと呼ぶ。「python のプログラムを動かす」という表現は上のような意味である。よって、プログラム (www.py) 自体に実行権限がなくても www.py を実行できる（注意：www.py に実行権限は不要だが、python が www.py を読める権限は必要である）。

2.26 scp

SSH プロトコル[*21]を利用して、自分のコンピュータのファイルを他のコンピュータに転送するコマンド。scp は「Secure Copy」の略。

♣ 2.26.1　書式

```
$ scp [options] SOURCE... TARGET
```

- 基本的に cp コマンドの SSH 版。「scp コピー元ファイル コピー先ファイル」のように使う。コピー元/コピー先ファイルはネットワークの向こう側でも良いところが cp コマンドと異なる。「2.4 cp」(p.9) 節も参照
- cp コマンドと同じく、TARGET 部分はディレクトリでも良い
- cp コマンドと同じく、TARGET がディレクトリの場合、「SOURCE...」の部分は複数のファイルでも良い

♣ 2.26.2　実行例

実行例

sample.txt ファイルを、example という名のサーバ上の/home/test ディレクトリへコピーしたい

```
$ scp sample.txt testuser@example:/home/test/
```

- 「sample.txt」がコピー元のファイル名
- 「testuser@example:/home/test/」がコピー先のファイルパス
  - testuser がコピー先のユーザ名
    (example サーバにログインする際のユーザ名)
  - example がドメイン名（サーバ名）
  - ホスト名 (この例では example) の部分は IP アドレスでもよい

♣ 2.26.3　知っていると便利なオプション: -r

コピーしたい対象がディレクトリの場合に使用する。指定したディレクトリ以下すべてをコピーする（再帰的（recursive）の r)。

[*21] SSH (Secure SHell) protocol

-r 再帰的（recursive）オプション

-r オプションは「再帰」を意味するコマンドが多いので覚えておくと良い。もっとも chmod のように再帰に-r ではなく-R を使うものもあるのだが...
そのうち体が覚えるので、いくつかの例外以外は-r は再帰と覚えておいて間違いない。ようするに定番オプションの付け方には規則性があるという話である。

実行例

sampledir ディレクトリ以下すべてを、サーバ example の/home/test ディレクトリへコピーする

```
$ scp -r sampledir testuser@example:/home/test/
```

scp SOURCE... TARGET/

TARGET がディレクトリの場合、コピー先がディレクトリであることを強調するために右端に/を付けることが多い。つまり右端の/は付けても付けなくても良いが、(a) ドキュメントを読む人にコマンドの意図を伝えようとする心がけは大事 (b) (たまに、うっかりエラーを回避できることもある <- 生きる知恵)

2.27 ss

コマンド名は socket statistics の頭文字。ネットワーク通信で利用しているソケットの情報を出力する。

♣ 2.27.1 書式

```
$ ss [options]
```

- オプションなしでは情報過多
- 定番の `ss -l4n` の形で覚えることを推奨

♣ 2.27.2 実行例

実行例

```
$ ss
```

実行結果

```
Netid State  Recv-Q Send-Q Local Address:Port       Peer Address:PortProcess
udp   UNCONN 0      0           0.0.0.0:123       0.0.0.0:*
      〜 省略 〜
tcp   LISTEN 0      128         0.0.0.0:22        0.0.0.0:*
tcp   LISTEN 0      511         0.0.0.0:80        0.0.0.0:*
```

実用上は、開いているポート番号（**:数字**部分）の確認に使うことが多い。「IP アドレス:数字」形式の部分が重要である[*22]。その他の詳細は省略する[*23]。

♣ 2.27.3　知っていると便利なオプション: -l

数字ではなく英語小文字のエル、lima の l （listen の l に由来する）。接続待ち状態 (State が LISTEN) のソケットのみを表示する。サーバプロセスだけを表示したいときに使用する。

実行例

```
$ ss -l
```

実行結果

```
Netid State      Recv-Q Send-Q Local Address:Port                 Peer Address:Port
u_str LISTEN     0      128    /root/.pm2/pub.sock 24878          * 0
u_str LISTEN     0      128    /root/.pm2/rpc.sock 24879          * 0
      〜 省略 〜
tcp   LISTEN     0      128     *:56958                           *:*
```

State が LISTEN の行のみが表示されている。

♣ 2.27.4　知っていると便利なオプション: -4

IPv4 のソケット情報のみを表示する。

実行例

```
$ ss -4
```

実行結果

[*22] 0.0.0.0 は「すべて（の IP アドレス）」という意味。「すべて」意味で*を使うツールも多い。
サーバやルータは複数のネットワークインターフェイス（IP アドレス）を持ちうることに注意

[*23] きちんとマニュアルに説明がないので、正解はソースコードを参照のこと。
https://www.kernel.org/pub/linux/utils/net/iproute2/

```
Netid State      Recv-Q Send-Q Local Address:Port               Peer Address:Port
udp   ESTAB      0      0      127.0.0.1:57674                 127.0.0.1:57674
tcp   ESTAB      0      0      127.0.0.1:6081                  127.0.0.1:35634
tcp   ESTAB      0      0      127.0.0.1:35000                 127.0.0.1:48549
      ～ 省略 ～
```

Address が IPv4 表記 (数字. 数字. 数字. 数字) の行のみが表示されている。

♣ 2.27.5　知っていると便利なオプション: -n

名前解決を行わない (プロトコル名やドメイン名が表示されない)

♣ 2.27.6　知っていると便利なオプション: -l4n

これまで説明したオプション l,4,n すべてを適用する。おすすめのオプション

実行例

```
$ ss -l4n
```

実行結果

```
Netid State      Recv-Q Send-Q     Local Address:Port       Peer Address:Port
tcp   LISTEN     0      128        *:59707                 *:*
tcp   LISTEN     0      128        *:40603                 *:*
      ～ 省略 ～
```

2.28　ssh

他のコンピュータ・サーバに SSH(Secure Shell) プロトコルでログインする。非対称鍵暗号[*24]による「認証」と「通信路の暗号化」を行う商用インターネット時代のデファクトスタンダードツール。サーバ運用管理の必須ツール。

認証には、非対称鍵暗号と対称鍵暗号の両方式が利用できるが、インターネットからアクセスできるサーバは非対称鍵暗号認証で運用するべきである。

♣ 2.28.1　書式

```
$ ssh  [options] DESTINATION
$ ssh  [options] DESTINATION [command [argument...]
```

[*24] 本文中では「非対称暗号」に統一しておいたが、「公開鍵暗号」という言いまわしの方が広く使われていると思う。「公開鍵」と「秘密鍵」という鍵のペアを用いる。「公開鍵」という表現は「方式」「鍵」どちらの話か？ と素人を混乱させるので、非対称鍵という言い回しをしている。

- 「ssh サーバ」の形式でサーバへ遠隔ログインできる
  - 引数が「サーバ」形式の場合、サーバにログインするユーザ名は（今 ssh コマンドを実行しているユーザ名と）同じと想定される
  - 「ユーザ@サーバ」つまりメールアドレス形式で、サーバのユーザ名とサーバ名を同時に指定できる
- 「ssh サーバ コマンド」の形式でコマンドを遠隔実行できるが、演習では使わないため説明は省略

♣ 2.28.2　実行例

実行例

```
$ ssh user@16.32.64.128
```

実行結果

```
$ ssh user@16.32.64.128
    〜省略〜
user@16.32.64.128$
```

最終行は（ログイン先サーバの）シェルのプロンプトである。この例では$ の左側に「ユーザ名@ip アドレス」が表示されている。

♣ 2.28.3　知っていると便利なオプション: -i （i, india の i)

-i オプションには、引数として「(SSH 接続に使用する非対称鍵暗号方式の）**秘密鍵**ファイル」を指定する。ちなみにオプション名は **identity の i** である。

実行例

```
$ ssh -i .ssh/id_rsa.pem user@16.32.64.128
```

.ssh/id_rsa.pem の部分が秘密鍵ファイルである

実行結果

```
$ ssh -i .ssh/id_rsa.pem user@16.32.64.128
    〜省略〜
user@16.32.64.128$
```

説明は同上なので省略

2.29 sudo

ルート権限[*25]を持たない一般ユーザが、sudo コマンドを使うと、コマンド単位でルート権限を利用できる

♣ 2.29.1　書式

```
$ sudo [options] command [arg ...]
```

♣ 2.29.2　実行例

実行例

```
$ sudo id
```

実行結果

```
$ sudo id
uid=0(root) gid=0(root) groups=0(root)
```

♣ 2.29.3　ルート権限の特長

ID が変わる

コマンドは実行時のユーザ ID（uid）と紐付いている。ルート権限を持つユーザ (root) の uid は 0 なので、sudo を利用して実行したコマンドは uid が 0 で動作する。

実行例

id コマンドに、コマンド実行時の uid を表示させてみよう。

id コマンドにより自身の uid について確認できる。（「2.10 id」(p.17) 節を参照）

```
$ id -u
```

sudo をつけて実行する場合は次のようにする。

```
$ sudo id -u
```

実行結果

[*25] ほぼすべての操作が可能な管理者の権限

```
$ id -u
1000
$ sudo id -u
0
```

sudo を使う時と使わないときで uid が異なることがわかる

♣ 2.29.4　特権ポートでサーバを起動する

ネットワークでは、通信を識別するために、ネットワークの入出力口（ソケット）に識別番号（ポート番号）をつけている。当然サーバが待ち受けするポートもある。広く利用される（公益性が高い）サーバが使うポート番号には 1024 未満が割り当てられており、**特権ポート**と呼ばれている。このポートを利用する際、つまりサーバを起動する際には、ルート権限が必要である

sudo コマンドを利用することでルート権限を取得し、特権ポートでサーバを起動することが出来るようになる

実行例

ポート番号 80 番で Python の WWW サーバを起動する例を見てみよう。

▼一般ユーザが普通にサーバを起動する場合

```
$ python3 -m http.server 80
```

▼sudo コマンドをつけて、サーバを起動する場合

```
$ sudo python3 -m http.server 80
```

実行結果

HTTP デフォルトのポート番号 80 番で Python の WWW サーバを起動しようとすると、次のように `PermissionError: [Errno 13] Permission denied` というエラーメッセージが表示され、異常終了する。

```
$ python3 -m http.server 80
Traceback (most recent call last):
    〜省略〜
PermissionError: [Errno 13] Permission denied
```

sudo コマンドをつければ起動できるようになる

```
$ sudo python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
    〜省略〜
```

sudo なしだと、**80 番ポートを開く権限がない**ため、WWW サーバを起動できない。sudo を使うことで、**80 番ポートを開く権限得られる**ため、WWW サーバを起動することができる。

♣ 2.29.5　注意点

root 権限を持つユーザはセキュリティ上ねらわれやすいユーザであるため、直接ルートユーザにログインするのではなく、一般ユーザでログインし、必要な場合にのみ sudo コマンドを利用してルート権限を利用することを心がける。

2.30　telnet

他のコンピュータ・サーバにログインできる。暗号化は行わない。暗号通信をしたい場合は、ssh コマンドを利用する。今日では、「デバッグ」か「LAN でネットワーク機器の設定」に利用するくらいだが、何かと便利なコマンドである。

♣ 2.30.1　書式

```
$ telnet [options ...] [HOST [PORT]]
```

- 引数なしで `telnet` コマンドを起動すると、対話モードに入ってしまうので注意。quit で抜けることが出来る
- 「`telnet HOST`」形式で起動すると、HOST へ telnet プロトコルで接続する
- 「`telnet HOST PORT`」形式で起動すれば、HOST のポート番号 PORT とのあいだに TCP 論理回線が構築される。アプリケーションプロトコルのデバッグで用いる

♣ 2.30.2　実行例

実行例

アメリカ国立標準技術研究所 (NIST) の時刻サーバへ接続する

```
$ telnet time.nist.gov 13
```

実行結果

```
$ telnet time.nist.gov 13
60642 24-11-04 15:18:00 00 0 0 762.9 UTC(NIST) *
```

NIST から時刻の返信が来ている。読み方は次のとおり

- 60642 が 1970/1/1 から現在 (記録日 2024/11/4) までの日数
- 15:18:00 はアメリカの現在時刻
- 00 はエラーが発生してないことを確認する。(00 以外ならエラー)

- `0 0 762.9` は時間の補正に関する値である。
- UTC(NIST) は、NIST の協定世界時である UTC(いわゆるグリニッジ標準時) で時刻表示していることを示している

2.31 top

top コマンドは、OS のステータスのサマリを表示する。

- 表示は**数秒おきに自動的に更新**される
- 8 行目以下は ps コマンドに似たプロセス一覧表
  - いま活発に動いているプロセスが上の方に表示される
  - 同じコマンドが表示され続けているなら、そのコマンドはシステムを重くしている犯人候補だ

♣ 2.31.1 書式

```
$ top
```

```
top - 01:29:59 up 1 min,  1 user,  load average: 0.09, 0.04, 0.01
Tasks:  86 total,   1 running,  85 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.3 sy,  0.0 ni, 89.0 id,  0.6 wa,  0.0 hi,  0.0 si, 10.1 st
MiB Mem :    970.0 total,    748.8 free,    229.1 used,    123.9 buff/cache
MiB Swap:      0.0 total,      0.0 free,      0.0 used.    740.9 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
     36 root       0 -20       0      0      0 I   0.3   0.0   0:00.01 kworker/0:1H-kblock›
›d
      1 root      20   0  102128  12068   9184 S   0.0   1.2   0:00.79 systemd
      2 root      20   0       0      0      0 S   0.0   0.0   0:00.00 kthreadd
   ... 以下略 ...
```

特に注意するべき項目は次の 2 ヶ所だろう。

- `load average: 0.09, 0.04, 0.01`
  - どれくらいシステム（OS）が忙しいのか？ の目安[*26]
- `MiB Mem :    970.0 total,    748.8 free`
  - メモリ使用量。この例では、トータルが 970MB、未使用が 748MB（つまり 220MB ほど使用中）なので、まだまだ余力があることが分かる

表示の詳細は Unix マニュアルを参照のこと。

[*26] おおむね CPU コア数以下なら気にしなくてよいだろう。load average とは、プロセスの待ち行列の長さなので、直感的な忙しさとは微妙に異なるが、システム負荷の目安とされている。

2.32 tracepath

宛先にパケットを送信し、宛先までに通過する各ルータの IP アドレスを表示する。宛先のホストまでのネットワーク経路を可視化するために利用する。

♣ 2.32.1　書式

```
$ tracepath [options] DESTINATION
```

DESTINATION はホスト名か IP アドレス

♣ 2.32.2　実行例

実行例

```
$ tracepath IPアドレス
```

実行結果　（192.159.190.35 nintendo.com）

```
$ tracepath 192.159.190.35
1?: [LOCALHOST]                      pmtu 1280
1:  LAPTOP-A860U3T2.mshome.net                            1.443ms
1:  LAPTOP-A860U3T2.mshome.net                            0.383ms
2:  unifi.localdomain                                     1.811ms
3:  10.252.0.1                                            1.910ms
4:  118.23.89.234                                         4.303ms
5:  118.23.89.77                                          5.230ms
6:  221.184.5.205                                         6.879ms
7:  122.1.245.193                                        18.817ms
8:  122.1.245.206                                        21.827ms
9:  ae-9.a03.tokyjp05.jp.bb.gin.ntt.net                  19.898ms              ← NCOM,東京
10:  ae-5.r32.tokyjp05.jp.bb.gin.ntt.net                 19.778ms asymm 11     ←ここまで日本国内
11:  ae-4.r27.snjsca04.us.bb.gin.ntt.net                 128.045ms             ← NCOM,サンノゼ
12:  ae-0.a02.sttlwa01.us.bb.gin.ntt.net                 120.781ms asymm 13
13:  ae-0.comcast.sttlwa01.us.bb.gin.ntt.net             116.382ms             ←ここまでNCOM
14:  be-1211-cs02.portland.or.ibone.comcast.net          147.274ms asymm 16    ←ここからcomcast
15:  be-36131-arsc1.burien.wa.seattle.comcast.net        136.669ms asymm 16
16:  ae-501-ar01.seattle.wa.seattle.comcast.net          121.145ms asymm 17
17:  be-2-sur02.bellevue.wa.seattle.comcast.net          152.747ms asymm 20
18:  be-1-sur03.bellevue.wa.seattle.comcast.net          149.624ms asymm 19
19:  50.231.28.130                                       128.459ms asymm 18
20:  no reply
   〜省略〜
30:  no reply
               Too many hops: pmtu 1280
               Resume: pmtu 1280
```

出力の読み方：日本のサーバ〜NCOM[*27]〜アメリカの Comcast[*28]のルータ群を

[*27] NTT コミュニケーション。旧 Verio を買収して世界規模の長距離回線を擁する Tier-1 のキャリア。業界人は NTT COM ではなく NCOM（えぬこむ）と呼ぶ。電車の中で「NCOM が〜」とか電話してる同業者がいるとドキドキするよね？（w）

[*28] アメリカのケーブルテレビ大手

経由していることが分かる。14 番目のルータのドメイン名にはポートランドとある。サンノゼもポートランドもデータセンター銀座である。NCOM の海底ケーブルで太平洋を超え（10 番目〜11 番目）、サンノゼで陸揚げし、アメリカ西海岸を北上して、ポートランドで comcast のアメリカ内陸ネットワークに入ったことが読み取れる。

♣ 2.32.3　知っていると便利なオプション: -n

DNS の検索をせずに処理を実行する。

実行例

```
$ tracepath -n IPアドレス
```

実行結果 （192.159.190.35 nintendo.com）

```
$ tracepath -n 192.159.190.35
1?: [LOCALHOST]                      pmtu 1280
1:  172.29.208.1                                           1.404ms
1:  172.29.208.1                                           1.343ms
2:  192.168.1.1                                            1.767ms
3:  10.252.0.1                                             3.226ms
4:  118.23.89.234                                          9.273ms
5:  118.23.89.77                                          11.126ms
6:  221.184.5.205                                         12.700ms
7:  122.1.245.193                                         24.420ms
8:  122.1.245.206                                         24.939ms
9:  120.88.53.21                                          25.712ms
10:  129.250.5.109                                         19.804ms asymm 11
11:  129.250.5.78                                         125.279ms
12:  129.250.5.227                                        133.718ms asymm 13
13:  129.250.66.106                                       115.329ms
14:  96.110.44.93                                         128.149ms asymm 15
15:  68.86.93.2                                           129.075ms asymm 17
16:  96.216.153.6                                         144.728ms
17:  69.139.164.126                                       130.598ms asymm 19
18:  68.86.96.142                                         150.784ms asymm 19
19:  50.231.28.130                                        131.109ms asymm 18
20:  no reply
   〜省略〜
   30:  no reply
                Too many hops: pmtu 1280
                Resume: pmtu 1280
```

ドメイン名ではなく IP アドレスで表示されていることが分かる。

-n オプションの意義

ネットワーク系のコマンドは、-n オプションをつけて実行することを心がけたい（その手のコマンドには普通-n オプションがあるはず）。なぜなら、この手のコマンドを実行するのは障害対応時であり、DNS も確認項目の一つだからだ。DNS 障害に引きづらずにネットワークの動作確認をする必要がある。

2.33 traceroute

宛先にパケットを送信し、宛先までに通過する各ルータの IP アドレスを記録する。記録された IP アドレスによって、宛先のホストまでのネットワーク経路を可視化するために利用する。

歴史的には **traceroute** コマンドが初出で、ほとんどの Unix/Linux で利用できるのは **traceroute** コマンドである。当然 Debian/GNU Linux にも存在する（デフォルトでは入っていないかもしれないが、その時は apt install traceroute)。ただし、なぜか Ubuntu には **traceroute** ではなく **tracepath** コマンドが入っているらしい[*29]。

traceroute と **tracepath** の基本的な使い方は一緒なので、詳細は **tracepath**「2.32 tracepath」(p.50) 節を参照

♣ 2.33.1 書式

```
$ traceroute [options] DESTINATION
```

- DESTINATION はホスト名か IP アドレス
- **-n** オプションあり

Windows でもネットワーク障害切り分けに ping と tracert が使える

Windows のコマンドプロンプトでも **ping** と **tracert** コマンド (Windows 版 traceroute) が利用できる。ただし Microsoft が再実装したものらしく、オプションが Unix と異なるため注意が必要だ。それでも非常時の障害きりわけには有効なので、Windows にもパチモノが搭載されていることを覚えておこう。

2.34 tree

ファイルとディレクトリをツリー状に表示する。ファイルやディレクトリの整理状況を把握するために利用する。

♣ 2.34.1 書式

[*29] 少なくとも AWS Academy(vocareum) の環境では tracepath しか使えない

```
$ tree [options] [ディレクトリ]
```

♣ 2.34.2　実行例

実行例 （admin ディレクトリを出力する場合）

```
$ tree admin
```

実行結果 （admin ディレクトリを出力する場合）

```
$ tree admin
admin
├── __pycache__
│   └── bws.cpython-311.pyc
├── bws.py
├── cc-04.py
├── htdocs
│   ├── cart.html
│   ├── index.html
│   ├── lsform.html
│   ├── rek_janken.html
│   └── upload.html
├── rekognition.py
└── www.py

3 directories, 10 files
```

♣ 2.34.3　知っていると便利なオプション

♣ 2.34.4　-d オプション

ディレクトリのみをツリー状に出力する。

実行例

```
$ tree -d ディレクトリ
```

実行結果 （admin ディレクトリ内のディレクトリのみを出力する場合）

```
$ tree -d admin
admin
├── __pycache__
└── htdocs

3 directories
```

♣ 2.34.5　-a オプション

隠しファイルを含むすべてのディレクトリとファイルを出力する。

実行例

```
$ tree -a ディレクトリ
```

実行結果 （admin ファイル内のすべてのディレクトリとファイルを出力する場合）

```
$ tree -a admin
admin
├── .bash_history
├── .bash_logout
├── .bashrc
├── .local
│   └── share
│       └── nano
├── .profile
├── .ssh
│   └── authorized_keys
├── .sudo_as_admin_successful
├── .viminfo
├── __pycache__
│   └── bws.cpython-311.pyc
├── bws.py
├── cc-04.py
├── htdocs
│   ├── cart.html
│   ├── index.html
│   ├── lsform.html
│   ├── rek_janken.html
│   └── upload.html
├── rekognition.py
└── www.py

7 directories, 17 files
```

.ssh など.(ドット) から始まる隠しファイルや隠しディレクトリも表示されている。

2.35　unzip

zip アーカイブ（ファイル）から、アーカイブ内のファイルを取り出す（解凍する）コマンド。zip コマンドの反対（「2.37 zip」(p.59) 節も参照）

♣ 2.35.1　書式

```
$ unzip [options] 解凍したいファイル名
```

♣ 2.35.2　実行例

（test1.txt と test2.txt が格納されている）example.zip を解凍することを考える。

実行例

```
$ unzip example.zip
```

実行結果

```
$ unzip example.zip
 Archive:  example.zip
  extracting: text1.txt
  extracting: text2.txt
```

2.36 vi

Linux を含む Unix ベースの OS で利用できるテキストエディタ。vi は、キーボードのみの編集を前提にしたコマンドラインベースのエディタであり、テキスト編集にはキーボードショートカットを使用する。

♣ 2.36.1　書式

```
$ vi [options] [file ...]
```

♣ 2.36.2　実行例

実行例

```
$ vi ファイル名
```

実行結果

- ファイル名を指定して起動する
- そのファイルが実行したディレクトリに無ければ、ファイルを新たに作成する
- その後エディタ画面になる。コマンドモードである

♣ 2.36.3　コマンドモードと入力モード

vi には**モード**という考え方があり、モードごとに挙動が異なる。**コマンドモード**と**入力モード**の 2 つがある。vi 起動時のモードはコマンドモードになっている。

エディタとして期待される「文字を入力できる状態」が「入力モード」だ。コマンドモードは、その他の制御を行うモードで、Microsoft Word でいえば画面上にある様々なボタン/リボンに対応する。

♣ 2.36.4　よく使うコマンドの紹介

▼表 2.13: vi のコマンドモードで知っておくと便利なコマンド

コマンド	説明
ZZ	上書き保存しつつ終了
:wq	上書き保存しつつ終了
:q!	保存せずに（変更を廃棄して）、終了
:w	上書き保存（保存のみ）
yy	カーソルのある行をコピー (クリップボードへコピー)
p,P	ペースト
dd	カーソルのある行を削除
x	カーソル下の一文字を削除

モード変更

i,a

コマンドモードから入力モードへ移行する。入力位置は i キーでは現在のカーソル位置に文字が挿入される（insert の i)。a キーではカーソル位置の右から文字が挿入される（append の a)。

esc

入力モードからコマンドモードへ移行する。

ファイルの保存と終了

コマンドモードから以下のコマンドを実行することで保存及びエディタを終了することができる。コマンドは画面下部で入力できる。

ZZ

shift を押しながら z を二回たたくというシンプルな方法。なぜかメジャーな方法と認識されていないらしい。不思議。

- **変更があれば上書き保存**後に終了
- **変更がないなら即終了**。変更を捨てたい場合は、次の`:q!`

:q!

変更内容を保存せずに終了する場合には`:q!`だけを入力する。

:w

「**上書き保存**」。変更内容を保存したい場合に利用。エディタは終了しない。

:wq

「上書き保存」しつつ「終了」。変更内容を保存して終了したい場合に入力。

コピー＆ペースト

コマンドモードで以下のコマンドを実行するとコピー及びペーストができる。

入力モードの場合には、マウスでコピー＆ペーストも可能だ。マウスで文字列をドラックして選択し、右クリック後、コピーを選択すると文字列のコピーができる。同様に貼り付ける場合は、ペーストしたい位置にカーソルを移動後、右クリックしてペーストを選択することで、文字列のペーストができる。

yy

カーソル位置の行をコピーする。

数値 +yy

カーソル位置以下の数値分の行を一括でコピーする。

p,P

コピーした行をペーストする。p であればカーソルの一つ下の行に、P であればカーソルの一つ上の行にペーストする。

♣ 2.36.5　実行例 （コマンドモードでの操作）

次のような 4 行のファイルがあるとする。今、カーソルは 1 行目の右端にある

```
siritori|←カーソル位置
ringo
gorira
raion
```

説明の都合上、以下では左端に行番号をつけるが実際には無いことに注意されたい。

```
1 siritori|←カーソル位置
2 ringo
3 gorira
4 raion
```

実行例 1

```
1 siritori|←カーソル位置
2 ringo
3 gorira
4 raion
```

ここで、コマンドモードに移行し、4yy を入力する（1 行目〜4 行目がコピーされた、いわゆるクリップボードへコピーされた状態）

実行例 2

```
実行例1の続き
1 siritori
2 ringo
```

```
3 gorira
4 raion
5 |←カーソル位置
```

入力モードで 5 行目を作成（当然カーソルも移動している）し、コマンドモードに戻り、p を入力すると

実行結果

```
1 siritori
2 ringo
3 gorira
4 raion
5
6 siritori
7 ringo
8 gorira
9 raion
```

6 行目から 9 行目に（クリップボードから）1〜4 行目の内容がコピーされた。
まとめると次のような操作をしたことになる。

- 実行例 1
  - 4yy でカーソルの位置を含めて 4 行分 (1 行目〜4 行目) をコピー
- 実行例 2
  - カーソルが 5 行目にあるところで p を実行した
  - p は「カーソルから一つ下の行へのコピー」なので、5 行目には何も書かれずに、6 行目から実行例 1 でコピーしたテキスト (1〜4 行目) が書かれる (コピー＆ペーストされた)。

vi は使えたほうが良いのか？

実運用環境では、エディタが vi しか入っていない Unix/Linux 環境もありうるので、最低限の vi 操作方法は知っておいたほうが良い。

ただ、vi の学習コストを考えると、すなおに nano をインストールして使う方が良いと思う（客先の場合には nano を入れさせてもらう交渉をする)。nano については「2.20 nano」(p.30) 節を参照。

コマンドに:が付いたり付かなかったりするのは何故？

:で始めるコマンドは ex コマンドを呼び出す呪文だ。ex とは何か？ というと vi の一つ前に開発されていたエディタで、実際のところ vi と ex は同じプログラムへの異なるインターフェイスである。

こういうインターネット古代史は BSD のマニュアルにはきちんと書いてあるが:-)、vi(vim) のマニュアルには書いていないようである (少なくとも Debian 同梱の vi(vim) には無い)。まぁ古代史を知らなくとも困らないが、我々の道具が何故このような形に落ち着いているのか興味ないだろうか？ (「わたし、気になります！」という IT 古典部気質の人は、エンジニア適性があると思う)

コンピュータ開発史については YAPC::Hakodate で少しだけ語っているので、speakerdeck のスライドを参照してほしい (URL は脚注[*30]を参照)。動画も近々 Youtube に上がると思う (まだ準備中らしい)。

2.37　zip

ファイルを zip 形式でアーカイブするコマンド。反対は unzip コマンド (「2.35 unzip」(p.54) も参照)

♣ 2.37.1　書式

```
$ zip [options] アーカイブファイル名 アーカイブしたいファイルの名前...
```

「アーカイブしたいファイルの名前...」は可変長である。

♣ 2.37.2　実行例

text1.txt と text2.txt を example.zip にアーカイブする。

実行例

```
$ zip example.zip test1.txt test2.txt
```

実行結果

```
$ zip example.zip text1.txt text2.txt
   adding: text1.txt (stored 0%)
   adding: text2.txt (stored 0%)
```

[*30] YAPC::Hakodate 2024 "シェルと Perl の使い分け、 そういった思考の道具は、どこから来て、どこへゆくのか？ "(https://speakerdeck.com/fmlorg/sierutoperlnoshi-ifen-ke-souitutasi-kao-nodao-ju-ha-dokokaralai-te-dokoheyukunoka-v1-dot-1-0)

広告

深町および IT インフラ部が製作した技術同人誌（PDF）群は電子版を配布しています。ものによっては印刷版も無料で配布しています（在庫かぎり）

- 電子版の配布場所: `https://distribution.techbooks.fml.org/`
- 最新情報 (github): `https://github.com/techbooks-fmlorg`

既刊（2024 年度刊行, 監修あり）

- 厳選 Unix コマンド 第 2 版 (本書)
  - 2025/02/28 刊行、インフラ部の最新刊！
  - 公立千歳科学技術大学 IT インフラ部 [著]、深町賢一 [監修]
- IT インフラ演習環境 hands-on-base 0.3.0 の設計と実装
  - 技術: Unix コマンド (シェル)、docker-compose
  - LPI Webinar[*31]用に開発した環境です。これはインフラ部の成果物ではなく、作者は深町のみ。ただ、このシステムがインフラ部のシステムや研修の元ネタとして再利用されているので、ぜひ御一読を。
  - 企業の新人研修やインターンシップに最適と思います、どうでしょう？

近刊予定 (鋭意執筆中)

- 冴えない PaaS(へろく) の育て方 by 深町
  - 技術: HTML 1.1, CGI, Web API, Go 言語, docker-compose
  - PaaS の基本原理を学習する研究室兼インフラ部の研修用教材

[*31] 2024-06-08: `https://lpic-2024q2.demo.fml.org/`

あとがき / おわりに

「厳選 Unix コマンド」(第 2 版; v2.0.0 〜届けたいコマンド〜) は、いかがだったでしょうか。初版 (〜公立千歳科学技術大学 IT インフラ部へようこそ〜) と長さは同程度ですが、授業内容とシンクロするようにコマンドと内容を厳選したので、収録コマンド総数は 3.5 倍になりました。ちなみに、本書の全面改訂の予定は無く、マイナーアップグレード (v2.0.0 -> v2.1.0 -> v2.2.0, ...) が続いていく予定です。

監修の深町です。「前書き」「第 1 章」「あとがき」「ノート」、そして校正と組版もやっています。第 2 章の校正と組版をしていたら半分くらい書き直してしまいました:-) 脚本を発注したのに、それを叩き台に全然ちがうものを作ってしまい、「それなら発注するなよ！」と脚本家に怒られる監督さんみたい (苦笑)

執筆は、4 人でコマンド群を 7〜8 個ずつ担当し、メンターの 4 年生がそれらをレビューした上で開発ブランチに commit しています。第 2 章は、それらの原稿を合体し、大幅に改訂したものです。そのため、たまに文体にゆれがありますが、それもオムニバス執筆の味ということで、ひとつよろしくお願いします。

「厳選 Unix コマンド」の大規模改修は、いったん終わりですが、他の技術ネタでもいい（インフラでなくてもいい）から、この「技術同人誌を書こう！ 友の会（？）」の部活動が継続するといいな〜と思います。つねに、大絶賛、執筆者を募集中です！！

♣ 著者紹介 (第 2 版)

大黒浩貴、小坂健人、山平竜世、野上航汰、(五十音順)
中西和音 (メンター、第一次レビュー担当)

♣ 著者紹介 (第 1 版)

川合志門、北川武、後藤駿介、田村跳飛、中西和音、栁田颯太 (五十音順)

ラピダス建設中のクレーンが工場の向こうに見える千歳市美々より (左端が大学)

索引

厳選 Unix コマンド v2.0.0

Selected Unix Commands for Beginners

2025 年 2 月 28 日　v2.0.0

著　者　公立千歳科学技術大学 IT インフラ部
監　修　深町賢一
発行者　深町賢一
連絡先　infra-club@cist.fml.org
　　https://selected-unix-commands.techbooks.fml.org/
印刷所　株式会社イシダ印刷

(powered by Re:VIEW Starter)