UNIX独習への近道

株式会社アーヴァイン・システムズ
中島雅弘［著］

技術評論社

はじめに

Unix/Linuxに関しては、コマンドリファレンスや、OSの教科書としての良書が世の中にたくさんあります。良書との出会いは、基本的な知識や形式を身につけ技術の向上を加速します。また、ワークショップ形式のハンズオン・トレーニングはさまざまな実践が体験できるうえ、トレーナーから経験者ならではの逸話や無関係そうにみえても重要な伏線も聴くことができます。ワークショップに臨む前に基本的な知識を身につけておけば、より効果的に技術の向上が図れます。

筆者の勤めるアーヴァイン・システムズでは、社内研修として、Unix/Linuxの知識と実践のバランスを考慮したワークショップを行っています。このワークショップをもとに、月刊誌『Software Design』の連載記事「Unixコマンドライン探検隊」が企画されました。本書は、この「Unixコマンドライン探検隊」を再編したものです。

アーヴァイン・システムズでは、理工学の高等教育を受けていない人、実践の現場での経験がない人も積極的に採用し、活躍してもらいます。派遣や出向といったスタイルはできるだけ避け、開発や顧客との調整など、プロジェクト全体で大きな責任を持ってもらえるチャンスに積極的にチャレンジさせています。座学の価値は重要と知ったうえで、実践による経験をより重視しているのです。

一方、現場で10年、20年活躍した経験豊富な人に、ずっと長く第一線で活躍しつづけてもらうことは、会社にとって大切なゴールの1つです。高度に発展した現在の情報システム科学技術は広大で複雑です。残念ながら技術の「文脈」を十分に理解していない、シニアなエンジニアも見かけます。対話的なワークショップは、離散的な知識や経験の断片をつなぎ合わせる役割もあります。ですので、アーヴァイン・システムズでは、中級上級の技術者であっても、できるだけ初心者とともに研修に参加し、Unixとシェルを再確認してもらっています。

Unixを探検、さまざまなコマンドを見て歩きましょう。探検するにあたっては、コマンドと縦-横につながるシステムの関係を理解することが重要です。縦とは、コマンドを使うアクター(人や別のシステムなど)と、シェルや深層にあるOS(カーネルやライブラリファンクション、デバイスドライバなど)です。横は、シェルを媒(なかだち)としたコマンド、OS(カーネルやライブラリファンクション)の結びつきや、ネットワークプロトコルを介したプロセス同士の結びつきのことです。

さらに、時間的な変化も重要です。プログラムは「作って終わり」ではなく、た

だ「リリースがある」だけです。そのソフトウェアはその後も使い続けられ、システムはもちろん、利用者を取り巻くさまざまな環境にあわせて変化していく必要があります。ソフトウェアエンジニアリングを担当する者だけでなく、オペレーションをする人、セキュリティを担う人との役割分担や協力し合うためには、専門家としての幅広く、奥深い知識と実践での経験が不可欠です。

本書では、OSの内部のしくみや、コマンド間の連携、プロトコルや慣習、名称の歴史的な背景なども取り上げています。失敗例も取り扱い、実践的な内容になることも大切です。中級上級の方にとっては、「ここは知らなかった」「そんなことがあったね」など、新たな発見があれば幸いです。筆者が普段、社内研修で話していることをできるだけカバーできるように、紙面の許す限り書かせていただきました。

これから恐る恐る冒険に出る初心者の皆さんにとって、未開の地「Unix」を探検して、見知らぬコマンド、慣習、プロトコル、制約と、さまざまなびっくりに出くわすことでしょう。それでも本書を読み終えたときには、Unixのコマンドライン環境が安全で効率がよく、とても居心地の良い天地となることは間違いありません。読者の皆さんも、私たちと一緒にUnixのグル[注1]を目指しましょう。

2018年1月
中島雅弘

注1) ヒンドゥー語の「導師」という意味の言葉から転じて、コンピュータ関連技術について高度な知識を持った専門家で、技術の指導や啓蒙を行う人のことを指します。

1-4 ディレクトリやファイルの属性理解がセキュリティの第一歩

プログラミングが上達してくると、「他人の書いたコードが気に入らない」、「汚く見える」というシンドロームに陥ります。とくに、人のコードをデバッグ・保守しなければならないときは、堪えます。人のソースコードを読むということは、その人のそのときの思考を1つ1つ理解しながらたどっていくことです。理解し難い、受け入れ難いは当然のことかもしれません。著者も、数ヵ月して見た自身のコードが、まったく残念だと思うことは頻繁にあります。一方まれに10年以上前に書いた自分のコードに感心することもあります。いずれにしても3ヵ月もしたら、自分のコードも他人のものと変わりありません。

あれもこれも否定して、コードを書き換えてしまったり、コードを書いた人の人格まで否定するような発言をすることは賢明ではありません。戦線拡大は不利益も拡大します。優れたソフトウェアエンジニアやシステムアドミニストレータは、不愉快は「ごくん」と飲み込んで、システムがどのように機能するかに注目して、どうしたら活用できるのか、最低限何を改善(ハック)すれば目的を果たせるのかを考えます。このような建設的な考え方は近年DevOpsでの組織と文化に対する提唱(お互いの尊重、お互いに対する信頼、失敗に対する健全な態度、相手を非難しない)やAgile Modelingの価値(コミュニケーション、勇気、フィードバック、謙虚さ、簡潔さ)と合致します。

所有者と権限

Unixは、マルチユーザ・マルチプロセスのオペレーティングシステムです。自分が編集しているファイルを、同時にほかの人が見たり編集したりすることもできます[注1]。逆に権限を設定すれば、自分だけしか読めない、実行できないファイルとすることもできます。また、プロセスにも実行しているユーザとグループの情報があります。実行中のプロセスが、ファイルやディレクトリにアクセスする権限は、プロセスのユーザとグループに従います。本節はこれら、ファイルの所有者と権限、プロセスとの関係について少し掘り下げてみることにしましょう。

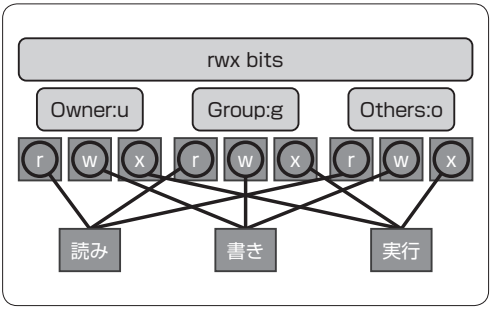
注1) 編集中の情報がリアルタイムで共有されるわけではありませんが。

ファイルの基本属性とプロセスの実行者

権限をはじめセキュリティの管理は、おもにシステム管理者の仕事です。今回説明する内容は、一般ユーザとしてだけでなく、システム管理者の視点も必要です。もちろん、自分のホームディレクトリやファイルは、あなた自身の手でしっかりと戸締まりしましょう。

所有者とグループに対してファイル、ディレクトリのアクセス権限(**図1**)の設定が基本です。実行できるプログラムに対してのセキュリティもファイルへのアクセス権限が軸です。安全にサービスが動くようにすること、セキュリティ診断でも、ここから確認していきます。プログラムを実行できないときは、権限があるかを確認してみましょう。

▼図1　rwx bits——3つの属性



アクセス権限は、ファイルの①所有者、②属するグループ、③その他の利用者、の3つの利用者クラスに対して設定できます。そして、基本の権限は“読み込み可能：r”、“書き込み可能：w”、“実行／検索可能：x”の3つです。これを、rwx bitsと呼ぶのでしたね[注2]。

プロセスは基本、実行した人・グループのもので、そのプロセスがファイルをオープンしたり実行したりできるかは、対象のファイルに設定された権限に従います。

STEP UP! setuid、setgid、sticky属性

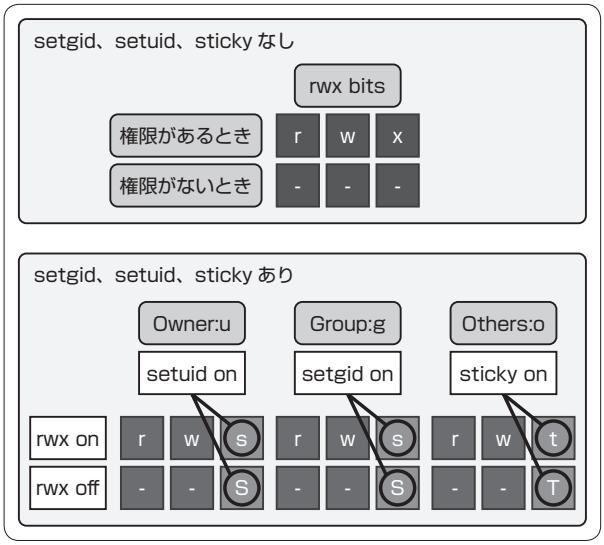
rwxに加えて、ファイルやディレクトリには、setuid(set-user-id：セットuid)、setgid(set-group-id：セットgid)[注3]、sticky(スティッキー)の3つの属性を指定することができます(**図2**)。

注2） 1-2参照。

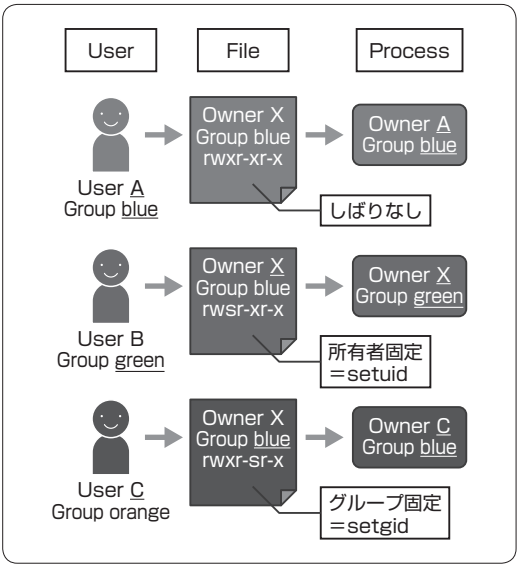
注3） set-uid、set-gidと記述することがあります。

実行可能ファイルのsetuid/setgid属性がオンなら、実行者のuid/gidではなくてそのファイルのuid/gidで実行します(**図3**)。

▼図2　lsの属性表示



▼図3　setuid/setgidのあるファイルとプロセスの所有



サービスのユーザアカウント（たとえばユーザ＝www、グループ＝www）を作ります。プログラムには所有者wwwとしてsetuidし、グループwwwに実行権限を与えておけば、誰が起動しても、サービスは常にwwwとしての権限で実行されるようにできます。

スクリプトファイルに対して、setuid/setgidを指定しても実行者はファイル所有者にはなりません。ファイルを実行するプログラムは、たとえばシェルスクリプトであれば/bin/shや/bin/bashでしょうし、Rubyスクリプトではrubyが実行されるファイルです。スクリプトは、これらシェルやRubyの動作手順を示したテキスト情報にすぎないからです。

ディレクトリのsetgidがオンなら、そのディレクトリの中のファイルとサブディレクトリを作成する場合のグループが、作成するユーザとは無関係に、そのディレクトリのグループとなります。作業ディレクトリをグループで共有して使うときに役立ちます。

ベトベト、ねばねば、貼り付くという意味のsticky。ディレクトリにsticky bitが立っている場合[注4]は、ディレクトリ中のファイルやディレクトリの削除制限をします。そのディレクトリに入っているファイルは、ディレクトリに対して書き込み権限を持っているユーザでそのファイルの所有者、ディレクトリの所有者、もしくはスーパーユーザがファイルを削除したり名前の変更ができます。つまりstickyを使えば、ディレクトリへのファイルやディレクトリ追加は許可しても、削除や名前の変更ができないようにできます。

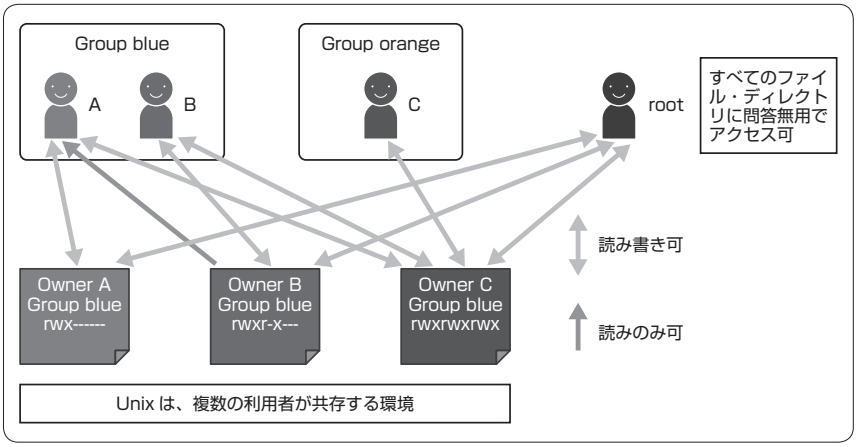
スーパーユーザ（root）は特別権限

Unixでは、特別な権限を持ったスーパーユーザが存在します。**root**というユーザで、UIDとGIDが両方0です。rootには原則アクセスできないファイルやディレクトリ、制御できないプロセスはありません（**図4**）。rootにもパスワードを設定しておけば、普通にログインすることもできます。セキュリティ上の理由から誰がスーパーユーザになったか、**記録が残るように**しなければなりませんが、rootで直接ログインしてしまうと、ログインした個人を識別できません。そのため、スーパーユーザ権限でアクセスする必要がある場合は、別の一般ユーザとしてログインして、**su**コマンドでユーザを変更したり[注5]、権限を制御・許可された

注4） 古くは、実行ファイルのstickyがオンになっていると、プロセスができるだけ物理メモリ内にとどまるようにする仕様でした。

注5） この方法もセキュリティの理由で推奨されなくなりました。

▼図4　所有者、グループ、その他によるアクセス制御とスーパユーザ権限



特定のユーザがsudoコマンドを使ってスーパーユーザ権限を発動します。

su—Substitute User identity—別のユーザidに切り替える

```
su [-] [ユーザ]
```

「ユーザ」を省略するとrootになります。-オプションを指定すると、新規にログインした状態と同じ、指定しなければ現在のシェル情報が引き継がれます。

sudo—SUしてDO—別のユーザ権限でプログラムを実行する

```
sudo [-s] [-u ユーザ] [コマンド]
```

sudoは、ユーザ対象コマンドの権限を個別に管理でき、sudoするとシステムのログに記録されるなど、rootでの直接ログインやsuに比べて安全面で優れています。-sオプションを使うと、許可されたユーザ[注6]は、rootになることができます。

sudoコマンドを実行したときに要求されるパスワードは、現在の利用者のパスワードを入力します。

注6） sudoは/etc/sudoersで別のユーザになることができるユーザを限定できます。

ログイン時などで入力を修正するときのTIPS

login、sudoなどでパスワードを求められるとき、入力の途中で間違いに気がついて、バックスペースキーで修正しようとしてもうまくいかないことがありませんか。この入力状態は、バックスペースが文字(Ctrl＋H)と入力され、前の文字の取り消しは機能しません。このときわざとログインを失敗して再挑戦するのは、次のプロンプトが出るまでに時間がかかったり、場合によっては複数の失敗でアカウントがロックされてしまうこともあり、賢明な対応ではありません。

パスワードなどの入力状態で使える編集操作キーは、**stty -a**コマンドを実行したときに出力される情報、"erase = ^?"や"kill = ^U"などなのです。Appleのキーボードでdeleteと刻印されたキーボードはDEL(アスキーコードで7F)が入力され、Windows系のキーボードのほぼ同じ位置にあるバックスペース(BSとかBACKとか←と刻印)はCtrl＋H(アスキーコードで08)になります。コマンドライン上は、DELもバックスペースも直前の文字が取り消されるので区別がしにくいかもしれません。で、"erace = ^?"の^?とは、DELのことです。Appleキーボードでは、パスワード入力で間違えた文字をDeleteキーで修正できるのですが、Windowsキーボードではバックスペースではなくて、Delキーが使えます。

またパスワード入力は、入力文字が画面にフィードバックされないので、何文字入れて、どこで間違えたかわからないというのもありますね。このときは、"kill = ^U"というところのCtrl＋Uを使えば、入力をすべてキャンセルした状態となり、1文字目から入力をやり直せます。ちなみにAppleキーボードでバックスペースを入力するには、Ctrl＋Hを入力します。

ユーザ、グループとアクセス権限

id—ID—ユーザID、グループIDなどを確認する

現在ログインしているユーザやグループを確認する必要があるときには、**id**コマンドで確認できます。ログインしているユーザには、現在のユーザ名と対応する(ユニークな番号)UIDと、現在のグループ名と対応する(ユニークな番号)GIDがありますが、idはこのいずれの情報も確認できます。ユーザ名だけを確認するのであれば**whoami**、所属グループを確認するには**groups**などのコマンドがidと同じように使えます。

id Ubuntuでの例
```
$ id ↵
uid=1000(masa) gid=1000(masa) groups=1000(masa),4(adm),24(cdrom),27(sudo),
30(dip),46(plugdev),113(lpadmin),128(sambashare)
$ id -un ↵
masa
$ id -gn ↵
masa
```

STEP UP! ユーザやグループの情報を確認・編集する

一般的なUnix系システムでは、ユーザ情報、グループ情報が、/etc/passwd注7、/etc/shadow、/etc/group/etc/gshadow注8に保存されています。

これらのファイルを直接編集することで、ユーザやグループを追加・変更・削除することができます。

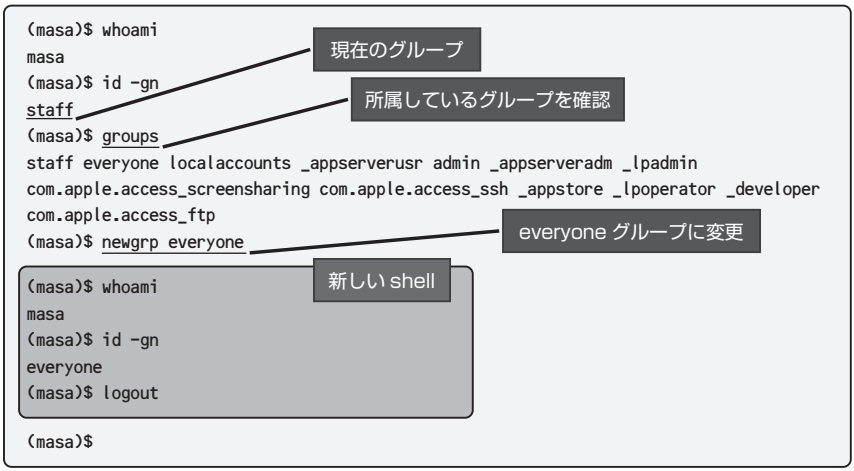
また、useradd、groupadd、userdel、usermod、groupdelなどのコマンドを使って、ホームディレクトリの管理などを併せて操作できます。

macOS(10.6以降)では、/etc/passwd、/etc/groupの情報は使わず、独自のディレクトリサービスを使うようになりました。このディレクトリサービスをコマンドラインで使うdscl注9コマンドがあります。

newgrp—NEW GRouP—新しいグループにログインする

ユーザは複数のグループに所属し、作業の役割ごとにグループを変更できます[注7]。newgrpコマンドは、同じユーザIDと新しいグループで、新しいシェルを実行します(**図5**)。ユーザは指定するグループに所属している必要があります。所属していないグループを指定すると、newgrpはそのグループに対するパスワードを要求します。パスワードがマッチしなければ、そのユーザのデフォルトグループでシェルを起動します。

▼図5 newgrp



注7) Linux系の環境では、groupに所属していても現在のグループがファイルの実行権限と異なったグループの場合、実行ができません。macOSなどBSD系の環境では、ファイルと同じgroupに所属していれば実行ができます。

chown—CHange OWNer、chgrp—CHange GRouP—ファイルやディレクトリの所有者、グループを変更する

既存のファイルの所有者、グループ情報を変更するには、**chown**、**chgrp**コマンドを使います。chownは、通常スーパーユーザ権限で実行します。chgrpは、自分が所属しているグループであればスーパーユーザでなくても変更できます。

オーナー情報は、自分が所有しているファイルでも、root権限でなければ変更できません。

rootではない一般ユーザによるchown実行の失敗例

```
$ chown root fig05.pptx ↵
chown: fig05.pptx: Operation not permitted
```

chownで所有者とグループを一度にまとめて変更することもできます。所有者名とグループ名の区切りには“**:**”を使います(**図6**)。Linuxでは名前の区切りに紛れがなければ“**.**”を使うこともできます。**:**の前にユーザ名を指定しなければ、グループのみを変更します。

▼図6　ファイル123の所有者、グループを変更してみる

```
bash-3.2# ls -l 123 ↵
-rw-r--r--  1 masa  staff  47  9 16  2015 123
bash-3.2# chown root 123 ↵
bash-3.2# ls -l 123 ↵
-rw-r--r--  1 root  staff  47  9 16  2015 123
bash-3.2# chgrp daemon 123 ↵
bash-3.2# ls -l ↵
total 40
-rw-r--r--  1 root  daemon 47  9 16  2015 123
bash-3.2# chown masa:staff 123 ↵
bash-3.2# ls -l 123 ↵
-rw-r--r--  1 masa  staff  47  9 16  2015 123
```

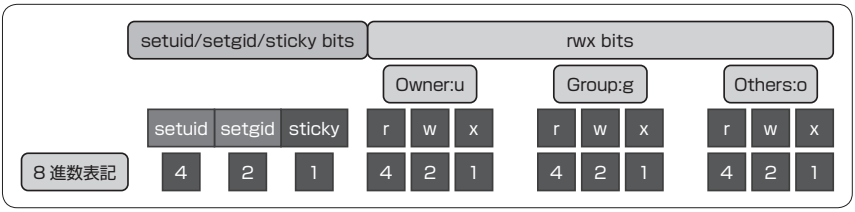
chmod—CHange MODe—属性／権限を変更する

chmodコマンドは、ファイルの属性を2種類の記法で変更します。属性の指定方法はrwxを用いたシンボル表記と、8進数表記です(**図7**)。

8進数表記(**図7**)は、setuid/setgid/stickyで1桁、所有者(Owner)で1桁、グループ(Group)で1桁、その他のユーザ(Other)で1桁の8進数(合計4桁)を、各属性をオンにしたいところの数字を足した値で表現します(**表1**)。setuid/setgid/stickyを指定しない場合は、3桁で表記することができます。

シンボル表記は、次のように“対象 演算子 権限”という式で表現します。

▼図7　rwx bits——3つの属性と8進数



▼表1　rwx bits——8進数表記例

lsの表示	8進表記
rwx rwx rwx	777
rw- rw- r--	664
r-s r-x ---	4550
rwx rws ---	2770
rwx --S ---	2700
rwx r-x r-t	1755
rws --- --T	5700

対象　　演算子　　権限

[u | g | o]　[+ | - | =]　[s | t | r | w | x]

u……所有者　　s ……setuid/setgid

g……グループ　　t ……sticky

o……その他　　r ……読み込み

+……オンにする　　w……書き込み

- ……オフにする　　x ……実行

=……指定した権限にする

chmodの使用例

fileのモードをrwxrwxrwxにする

```
$ chmod 777 file
$ chmod ugo=rwx file
$ chmod ugo+rwx file
```

fileのモードをrwsr-xr-xにする

```
$ chmod 4755 file
```

グループとその他のユーザからrを取り去る

```
$ chmod go-r
```

サブディレクトリに再帰的に適用する

chmod、chown、chgrpで、ディレクトリの下全部の属性を変更したい場合は、recursive(再帰)指定するオプション-Rで一括処理ができます。

本節の確認コマンド

【manで調べるもの(括弧内はセクション番号)】
su(1), sudo(8), sudoers(5), newgrp(1), id(1), whoami(1), chown(1), chgrp(1), chmod(1), stty(1)
- Linuxのみ -
useradd(8), userdel(8), usermod(8), groupadd(8), groupdel(8), groupmod(8)
- macOSのみ -
stiky(8), dscl(1)

Column

ディレクトリとファイルのrwx bitsと削除

ちょっと複雑なので、本文中に書ききれなかった、ファイル削除と権限の関係をまとめました。削除対象のファイルが存在しているディレクトリと、ファイルのアクセス権限によって、rmコマンドを実行したときの挙動を整理してあります。1つずつ確認してみてください。

下記の動作確認は、Ubuntu 16.04とmacOS 10.12でおこないました。

```
以下の操作はすべて、user=masa として実行
$ id -un↵
masa
```

自分の所有するディレクトリに対して

❶ディレクトリに書き込み権限があり、自分のファイルに書き込み権限がある場合

```
$ ls -la↵
total 0
drwxr-xr-x   3 masa      staff   102 10  4 11:11 .
drwxr-xr-x  75 masa      staff  2550 10  4 11:03 ..
-rw-r--r--   1 masa      staff     0 10  4 11:11 a

$ rm a↵
確認もなく、削除成功!
```

4-2 簡単な文字置換と汎用性の高い文字列演算と正規表現を使ってみる

Unixのツールは、1つ1つ独立したプログラムで、標準入出力を通して連結できることが特徴です。複数のツールを組み合わせたり、そこにシェルスクリプトによる制御を加えたりしていると、まるでオブジェクト指向でプログラムを作っているかのようです。Unixでは、オブジェクト指向という概念が生まれる前から、ソフトウェアモジュールの独立性や一貫したインターフェースがシンプルな形で実現されていました。

一歩進んだテキスト処理

4-1で、機敏で小さなテキスト処理コマンドをいくつか確認しました。今回は、正規表現を中心に、少し高度なテキスト処理を目指します。シェークスピアーのハムレットの英語文[注1]を題材にしてみます。ファイル名はhmlt.txtにしました。

手始めに4-1でも登場したwcを使って、復習がてら対象データの状況を分析します。

行数、単語数、文字数を数える
```
$ wc hmlt.txt↵
    9086   32014  179670 hmlt.txt
```

3つ並んでいる数字は、左から行数、単語数、文字数です。これらを個別に表示させるには、-l(行数)、-w(単語数)、-m[注2](文字数)のオプションを指定します。

ここから先のコマンドは、^、$、[、]、(、)など、記号を用いて動作を制御します。$、{、}、(、)、;、|、␣(半角スペースの意味)などは、文脈によっては先にシェルが解釈して、それからコマンドに渡ります。シェルにいじられたくない文字は、バックスラッシュ(\)やシングルクォート(')を使ってエスケープしなければなりません。

注1) ネット上にある、英語のテキスト(たとえば、http://shakespeare.mit.edu/hamlet/full.html)をhtmlではなく、プレーンテキストで保存して使いましょう。

注2) 以前は文字数を数えるにあたって-cを使っていましたが、こちらはバイト数を数えるオプションとして使います。

tr - TRanslate characters

trは、単純な文字の置き換えや削除をします。置き換えや削除の対象を文字の集合で表せるのが大きな特徴です。さらに、**文字クラス**は、後の正規表現でも使う重要な文字集合の表現方法です。

trや以降のコマンドで使える文字クラスには**表1**があります。

trの置換は、第1引数に与えられた文字列を1文字ずつ第2引数に対応させて置換します。第2引数が第1引数より短ければ、対応先のない第1引数中の文字には、第2引数の最後の文字を割り当てます。'-'を用いて連続する文字コードの範囲を指定することができます。

trを活用する例を見ていきましょう。

▼表1　POSIX文字クラス（これらはロケールの設定によってどの文字とマッチするかが決まります）

クラス	意味
[:alnum:]	英数文字
[:alpha:]	アルファベット
[:cntrl:]	制御文字
[:digit:]	数字
[:graph:]	グラフィック文字
[:lower:]	小文字
[:print:]	印字可能な文字
[:punct:]	句読記号
[:space:]	空白文字
[:upper:]	大文字
[:xdigit:]	16進数

```
小文字を大文字に変換する1
$ tr a-z A-Z < hmlt.txt↵
...略...
SPEAK LOUDLY FOR HIM.
TAKE UP THE BODIES: SUCH A SIGHT AS THIS
BECOMES THE FIELD, BUT HERE SHOWS MUCH AMISS.
GO, BID THE SOLDIERS SHOOT.

A DEAD MARCH. EXEUNT, BEARING OFF THE DEAD BODIES; AFTER WHICH A PEAL OF ↵
ORDNANCE IS SHOT OFF
```

次の書き方は、前の例と同じ動作になります。

```
小文字を大文字に変換する2
$ tr [:lower:] [:upper:] < hmlt.txt↵
```

trのオプション-cを指定すると、最初に指定された文字列の補集合を取ります。また、-sオプションを指定すれば、入力データ中で同じパターンが繰り返された場合1つにまとめます。\nは、C言語での表現と同様に改行を意味します。

```
アルファベット以外を改行に変換することで1行1単語の一覧を作る
$  tr -cs [:alpha:] " n" < hmlt.txt | sort | uniq
...略...
your
yours
yourself
yourselves
youth
zone
```

コントロール文字など、印字できない文字が入っているところを取り除きましょう。次の例では、改行も取り除かれます。

```
印字可能な文字だけにする
$ tr -cd [:print:] < hmlt.txt
```

シーザー暗号という、アルファベットを別のアルファベットに1対1で置き換える、簡単な暗号化をしてみます。A～MをN～Zに、N～ZをA～Mに置き換えています(小文字についても同じ)。第1引数と対応する第2引数の指定を確認してください。

```
シーザー暗号化して冒頭の8行だけ確認してみる
$ tr A-Za-z N-ZA-Mn-za-m < hmlt.txt | head -8
Unzyrg: Ragver Cynl



Gur Gentrql bs Unzyrg, Cevapr bs Qraznex
     Funxrfcrner ubzrcntr
     Unzyrg
     Ragver cynl
```

もう一度、同じ暗号化をすれば、もとの文を得られます。

```
冒頭8行の出力に同じシーザー暗号化で復号
$ tr A-Za-z N-ZA-Mn-za-m < hmlt.txt | head -8 | tr A-Za-z N-ZA-Mn-za-m
Hamlet: Entire Play



The Tragedy of Hamlet, Prince of Denmark
     Shakespeare homepage
     Hamlet
     Entire play
```

文字列の操作と正規表現

Unixでテキスト処理を自在に操るには、シェルの文字列演算子と正規表現の2つをきちんと扱えることが基礎技術です。ここからは、この2つの基本を見ていきましょう。

シェルの文字列演算

bashには、**表2**のような文字列を操作するしくみがあります。シェル変数と文字列演算${ : }演算です。

▼表2　変数内の文字列の操作

演算書式	意味
${変数#パターン}	変数の値のはじめの部分とパターンが一致した場合、最も短く一致した部分を取り除き、残りを返す
${変数##パターン}	変数の値のはじめの部分とパターンが一致した場合、最も長く一致した部分を取り除き、残りを返す
${変数%パターン}	変数の値の終わりの部分とパターンが一致した場合、最も長く一致した部分を取り除き、残りを返す
${変数/パターン/文字列}	変数の値で、パターンと最も長く一致した部分を文字列と置き換える。最初に一致した部分だけが置き換えられる。文字列がnullなら、一致した部分を削除
${変数//パターン/文字列}	変数の値で、パターンと最も長く一致した部分を文字列と置き換える。一致した部分は、すべて置き換えられる。文字列がnullなら、一致した部分を削除。変数に@か*を指定した場合、位置パラメータを順に処理して、展開結果はリストに入る

このしくみを使って、フルパスからベースネームを取り出す**basename**コマンド、フルパスからディレクトリ名を取り出す**dirname**コマンドと同じような働きをさせることができます。一連の操作を順に見ていきましょう。

```
変数の文字列操作の確認

以下'/etc/resolv.conf'を対象に試してみる
$ ls /etc/resolv.conf↵
/etc/resolv.conf

lsの結果を変数に入れる
$ i=$(ls /etc/resolv.conf) ; echo $i↵
/etc/resolv.conf

はじめに見つかった'.'までを取り除く
$ i=$(ls /etc/resolv.conf) ; echo ${i#*.}↵
conf

はじめに見つかった'/'までを取り除く
$ i=$(ls /etc/resolv.conf) ; echo ${i#*/}↵
```

```
etc/resolv.conf

フルパスからベースネームを取り出す
$ i=$(ls /etc/resolv.conf) ; echo ${i##*/}↵
resolv.conf

フルパスからディレクトリ名を取り出す
$ i=$(ls /etc/resolv.conf) ; echo ${i%/*}↵
/etc

次のように入れ子にする書き方はできない
$ i=$(ls /etc/resolv.conf) ; echo ${${i##*/}%.*}↵
-bash: ${${i##*/}%.*}: bad substitution

フルパスから拡張子を取り除いたファイル名を取り出す
$ i=$(ls /etc/resolv.conf) ; i=${i##*/} ; echo ${i%.*}↵
resolv

こうすれば、拡張子を変更できる
$ i=$(ls /etc/resolv.conf) ; i=${i##*/} ; echo ${i%.*}.def↵
resolv.def

拡張子を取り出そうとして失敗。どこがおかしいかわかりますか？
$ i=$(ls /etc/resolv.conf) ; i=${i##*/} ; echo ${i##.*}↵
resolv.conf

フルパスから拡張子のみを取り出す
$ i=$(ls /etc/resolv.conf) ; i=${i##*/} ; echo ${i##*.}↵
conf
```

ほかにもbashには、CやRubyでおなじみのprintfによる書式付き出力もあります。これらbashの変数の文字列操作機能を使いこなせれば、いちいち外部コマンドを呼び出さなくてもさまざまな文字列の加工ができて、ファイル操作やテキスト処理能力が向上します。

正規表現入門

正規表現は、その成り立ちの経緯から、基本正規表現(Basic Regular Expression)、拡張正規表現(Extended Regular Expression)や、拡張正規表現のさらに拡張されたものなどがあります。各コマンドで、どのレベルの正規表現が使えるか異なりますので注意してください。

正規表現は、とても強力で表記要素もたくさんあります。拡張正規表現(ERE)は、扱えるメタ文字が基本正規表現(BRE)に対して増えていますが、完全な上位互換というわけではありません。場面によって使い分けが必要です。

表3に加えて、**表4**の文字クラス、前出のPOSIX文字クラスも扱えます。難

▼表3　正規表現のメタ文字

文字	意味
.	改行を除く任意の1文字
^	行頭
$	行末
*	直前の正規表現の0回以上の繰り返し
?	直前の正規表現の0回か1回の繰り返し(EREのみ)
+	直前の正規表現の1回以上の繰り返し(EREのみ)
()	括弧内の正規表現をグループ化する(EREのみ)
|	選択'|'の前の正規表現か後ろの正規表現のどちらかとマッチする(EREのみ)
\メタ文字	'\'に続くメタ文字をエスケープする(特殊文字としてではなくその文字そのものとしてあつかう)
\{n\}	直前の1文字のn回の繰り返し(EREでは括弧の前の\は不要)
\{n,\}	直前の1文字が最低n回繰り返し(EREでは括弧の前の\は不要)
\{n_1,n_2\}	直前の1文字がn_1回からn_2回のうちいずれか繰り返し(EREでは括弧の前の\は不要)
\(... \)	正規表現をグループ化する(EREでは括弧の前の\は不要)
\n	グループ化したn番目の正規表現を示す

▼表4　文字クラスの表現

表現	意味	例
[～]	文字クラスのうち任意の1文字	[Ade] → Aかdかeにマッチする
[^ ～]	文字クラスの補集合のうち任意の1文字(はじめの'[' の直後に'^'を記述した場合のみ)	[^PA] → PとA以外の文字にマッチする
[○-◎]	'-'は連続する文字コードの範囲を示す	[b-d] → bかcかdにマッチ

しそうですが百聞は一見にしかず。この後も、実際にコマンドを使いながら見ていきましょう。

grep fgrep egrep - Global Regular Expression Print

データの中に、文字列とマッチする行を抽出したり、マッチしない行を抽出したりと、とにかく便利に使えるのがgrepファミリーです。

STEP UP!

歴史的経緯は、grep、fgrep、egrepは異なるバイナリで配布されていました。正規表現を使わなければfgrepは高速です。egrepは強力で高速なパターンマッチアルゴリズムを使います。grepで片付くところでも、egrepを使っていると、「こいつわかっているな」と見てもらえるかも……。

現在は、grep、egrep、fgrepともmacOSでは同じバイナリです。CentOSでは、egrepとfgrepがgrepへのシンボリックリンク、Ubuntuでは、egrep、fgrepはシェルスクリプトで、内部でgrepを起動しています。

基本正規表現(BRE)が使えるgrep。-Eオプションをつければ拡張正規表現(ERE)が使えます。egrepは、これと同じ意味です。Extendの意味ですね。正規表現を使わないなら、-Fオプションを付けます。Fixedの意味です。

```
大文字で始まる行をすべて表示する
$ egrep '^[A-Z]' hmlt.txt
...略...
Take up the bodies: such a sight as this
Becomes the field, but here shows much amiss.
Go, bid the soldiers shoot.
A dead march. Exeunt, bearing off the dead bodies; after which a peal of
ordnance is shot off
```

```
大文字で始まらない行をすべて表示する
$ egrep ^[^A-Z] hmlt.txt
...略...
[Aside]  And yet 'tis almost 'gainst my conscience.
                  Look to the queen there, ho!
                  He is justly served;
                  Let us haste to hear it,
```

次の例では、EREとBREの違いを見てみましょう。まずは、egrepを使って。

```
行がすべて大文字でできている行の行数(ERE)
$ egrep ^[A-Z]+$ hmlt.txt | wc -l
     780
```

上の正規表現をgrepで実行してみましょう。結果は、780ではなくて0になりました。+がEREでしか使えないからです。そこで、grepではこう書きます。

```
行がすべて大文字でできている行の行数(BRE)
$ grep ^[A-Z][A-Z]*$ hmlt.txt | wc -l
     780
```

grepに-vオプションを指定すると、マッチしない行を表示します。これを使って行がすべて大文字の行以外の行数を数えます。

```
行がすべて大文字の行以外の行数を数える
$ egrep -v ^[A-Z]+$ hmlt.txt  | wc -l
    8306
```

英単語は、qの後ろは必ずuが続きます。ハムレットの中には、qに続く文字がu以外のものはあるのでしょうか。

```
qの後ろにuが付かない単語はあるのか
$ egrep [Qq][^Uu] hmlt.txt
```

必ず覚えておくべきgrepのオプション

grepには、いろいろと便利なオプションがあります。ここでは、必ず覚えておきたいオプションを紹介しておきます。

grepで文字列を検索する対象は、複数のファイルを横断できます。このときに便利な、「-H ファイル名も表示する（複数ファイル指定ではデフォルト表示）、-hファイル名を表示しない」、「-n ファイル名と行番号を表示する」、「-L マッチしなかったファイル名を表示、-l マッチしたファイル名を表示」があります。

「-i 大文字小文字を区別しない」、文脈がわからないと困る場合に便利なオプションとして「-A *n* 続く*n*行表示」、「-B *n* 前の*n*行表示」、「-C *n* 見つかった行の前後*n*行」（後ろAfter、前Before、中央Centerと覚えましょう）などは、活躍の場が多く必ず押さえておきたいです。

ありませんでした。:-) 続いて文字クラスを指定して、アルファベットか数字の後にumが続くところは、何行あるのか数えてみます。

```
アルファベットの後ろにumが付かない単語を含む行の数
$ egrep [[:alnum:]]um hmlt.txt | wc -l
      74
```

74行ありましたね。

有名な「成すべきか、成さざるべきか、それが問題だ」のセリフを探してみます。大文字か小文字かわかりませんので-iを指定、何行目かも知りたいので-n、前後の文脈も知りたいので-Cも指定します。

```
有名なフレーズを探してみよう1
$ egrep -niC 2 'to be or not to be' hmlt.txtl
```

見つかりません。条件を緩めてみます。

```
有名なフレーズを探してみよう2
$ egrep -niC 2 'to be' hmlt.txt
...略...
8165-To let this canker of our nature come
8166-In further evil?
--
8282-dizzy the arithmetic of memory, and yet but yaw
8283-neither, in respect of his quick sail. But, in the
8284:verity of extolment, I take him to be a soul of
8285-great article; and his infusion of such dearth and
8286-rareness, as, to make true diction of him, his
```

たくさんマッチして、目的のセリフを確認するのは大変です。条件を変えてみましょう。

```
有名なフレーズを探してみよう3
$ egrep -niC 2 'to be or' hmlt.txt
```

これでは、見つかりませんでした。行が別れているのかもしれませんし、途中に‘,’があるのかもしれません。別のワードで探してみます。

有名なフレーズを探してみよう4
```
$ egrep -niC 2 'or not' hmlt.txt
...略...
3789-HAMLET
3790-
3791:To be, or not to be: that is the question:
3792-Whether 'tis nobler in the mind to suffer
3793-The slings and arrows of outrageous fortune,
--
5901-Deliberate pause: diseases desperate grown
5902-By desperate appliance are relieved,
5903:Or not at all.
5904-
5905-Enter ROSENCRANTZ
```

3791行目に確かにありました。やはりTo beの後ろに‘,’がありました。でも、まだたくさんの個所にマッチしすぎていますね。もう少し絞ってみると、

有名なフレーズを探してみよう5
```
$ egrep -niC 2 'or not to be' hmlt.txt
3789-HAMLET
3790-
3791:To be, or not to be: that is the question:
3792-Whether 'tis nobler in the mind to suffer
3793-The slings and arrows of outrageous fortune,
```

見事、目的の文脈が抜き出せました。

macOS限定のコマンドsayは、テキストを喋らせられるコマンドです。引数で与えた文字列を読み上げます。sayは、さまざまな国の言葉を喋る音声(ここでは声優と言っておきましょう)が登録されています(追加も可能です)。このコマンドを使って、どの声優がハムレット役に適しているのか、オーディションを開催してみましょう。

ハムレット役のオーディション macOS版
```
$ for actor in $(say -v ? | egrep en_ | egrep -o '^[A-Za-z]+'); do sleep 2; echo ${actor}; say -v ${actor} ${actor}; sleep 1; (egrep -C 4 -i "not to be" hmlt.txt| say -v ${actor}) ; done
```

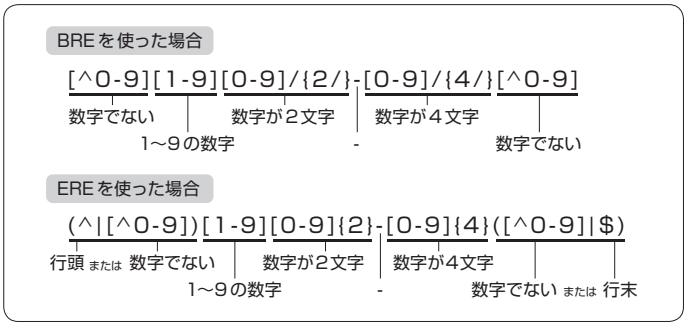
少々長いワンライナーですが、順番に読んでいけば大丈夫です。音声の一覧から、egrep en_で英語を、egrep -oで、名前部分(行頭からアルファベットの並びにマッチした個所)を抽出。名を名乗ってから、ハムレットの台詞を読み上げさせています。

Linuxにも**espeak**というコマンドがあります[注3]。オーディションを実施するスクリプトは、後に解説する**awk**を使っています。

ハムレット役のオーディション Linux版
```
$ for actor in $(espeak --voices | awk '$2 ~ /^en/{print $4}'); do sleep 2; ↵
echo ${actor}; espeak -v ${actor} ${actor}; sleep 1; (egrep -C 4 -i ↵
"not to be" hmlt.txt| espeak -v ${actor}) ; done↵
```

最後は、ハムレットを離れて、日本の郵便番号にマッチするかどうかの正規表現を作ってみましょう(**図1**)。

▼図1　BREを使った場合(上)とEREを使った場合(下)の日本の郵便番号



BREでは、行頭、行末に郵便番号が現れるとマッチしません。EREの選択'|'を使えば、この問題を克服できます。

本節の確認コマンド

【manで調べるもの(括弧内はセクション番号)】
`basename(1)`, `dirname(1)`, `tr(1)`, `grep(1)`, `awk(1)`, `say(1)`(macOSのみ), `espeak(1)`(Linuxのみ)
【以下はinfoコマンドを使って確認】
`printf`

注3) Debian系Ubuntuなどは、`$ sudo apt install espeak`で導入できるはずです。