地球惑星物理学科　計算機演習

～ Unix2 ～

森 晶輝 *

令和 3 年 4 月 8 日

はじめに

計算機演習も 3 回目となりました．基本的な部分は前回習ったと思いますので，今回は少し発展的だが知っておくと非常に便利な事柄について学んでいきます．また，慣れないことが続き消化不良に陥っている人も居るかと思いますので，コマンドラインでのファイル操作の復習も兼ねて行います．

Unix 端末は，最初はとっつきにくい部分もありますが，使いこなせるようになることで，他の汎用 OS 以上の機能やカスタマイズ性を実現することができます．そのためにシステムの理解は不可欠であり，講義の後半ではそのための基礎的な知識を説明したいと思います．

目次

* 宇宙惑星講座 杉田研究室 修士 1 年

1 遠隔ログイン

遠隔ログインを用いれば，リモートマシン（目の前にないマシン）を使うことができます．具体的な用途として，「**自宅から計算機室の端末にログインして演習で出題された課題に取り組む**」などが挙げられます．便利，というより今後時間のかかる課題に取り組む上で必須になる用途ですのでぜひ身につけましょう．

1.1 SSH

通信を暗号化した遠隔ログイン方法として広く使われているのが SSH (Secure SHell) です．やり方は以下の通りです．

SSH を利用してリモートホスト (遠隔地にあるサーバ) にログインするには,

```
$ ssh username@hostname
```

とします．ホスト名の部分には IP アドレスをそのまま書いても構いませんが，普通はドメイン名を入力します．例えば,

```
$ ssh s2126??@dover.eps.s.u-tokyo.ac.jp
```

といった具合です．入力後，パスワードを聞かれたら入力してリターンしてください.

なお，ローカルホスト (本人の手元にある端末) とリモートホストとでユーザ名が一致している場合は,

```
$ ssh hostname
```

と username を省略してもＯＫです.

初めて ssh でログインするホストにアクセスしたときは,

```
The authenticity host ‘edu 05 (192.168.1.105)’ can’ t be established.
DSA key fingerprint is cb:27:ec:3c:ff:02:f6:fc:9e:7b:39:80:e7:0f:9e:bf.
Are you sure you want to continue connecting (yes/no) ?
```

などとたずねられますので,

```
yes
```

と打ってください.

リモートホストで作業が終わってログアウトする際には,

```
$ exit
```

と打てばもとのローカルホストでの作業に戻れます.

ここで大事なことですが，ssh の後に「-X」(大文字であることに注意！) というオプションをつけないと，リモートホストでウィンドウを開く系の作業ができません．このオプションをつけることで，X window system (ウィンドウを開いてくれるシステム) も自動的に転送されるようになり，リモートで開いたウィンドウがローカルで見えて操作できるようになるのです[*1]．試しに「-X」なしでログインしてから,

```
$ emacs &
```

と入力して Emacs を起動しようとしてみてください．新しいウィンドウを開けずに，エラーとなってしまうはずです[*2]．その他のアプリケーションについても，ウィンドウを開くことができないはずです.

なお，この機能を利用するためにはローカルのマシンに X サーバという種類のソフトがインストールされている必要が

[*1] たまに**-X** をつけてもうまくいかない場合があります．その場合は**-XY** とすると解決することがあります.

[*2] **$ emacs -nw &**のように **nw** オプションをつけると，現在のシェル上に Emacs が起動するためエラーにはなりません.

あります．もちろん計算機室の端末 (**edu**) のマシンでしたら既にインストールされているので問題ありませんが，自宅からリモートログインする際には注意してください (後述).

1.2 自宅から SSH をするには

自宅のパソコンからでも，インターネットに接続されていれば，今みなさんがやっているのと同じように，演習室にリモートログインして作業ができます．これは非常に便利です．Windows にも標準で OpenSSH がバンドルされるようになりましたので，いずれの OS もそれぞれ適当なターミナルからそのまま SSH ができます．ただし，GUI なソフトを使用したい場合は X サーバーというソフトウェアをローカルホストにインストールする必要がありますので，使用している OS に合わせて適切な環境設定をしましょう.

1.2.1 eduvm の場合

標準で X サーバーまでインストール済みですので，追加でなにかをインストールする必要はありません.

1.2.2 Windows の場合

Windows の場合，X サーバーには無料の VcXSrv や有償の X410 などがあります．インストール方法については初回の環境構築で用いた wiki を参考にしてください.

1.2.3 Mac の場合

Mac の場合，XQuartz という X サーバーがあります．559 室の edu には標準でインストール済みです.

なお Mac から edu にログインした場合に，日本語が文字化けすることがあります．ログイン後

```
$ export LANG=ja_JP.UTF-8
```

とすることで解決するかもしれません.

1.3 設定

`~/.ssh/config` というファイルが SSH の設定ファイルです．たとえば,

```
Host dover
        HostName dover.eps.s.u-tokyo.ac.jp
        User s212601

Host edu01
        HostName edu01.eps.s.u-tokyo.ac.jp
        User s212601
        ProxyCommand ssh -X dover nc %h %p

Host www-geoph
        HostName www-geoph.eps.s.u-tokyo.ac.jp
        User s212601
        ProxyCommand ssh -X dover nc %h %p
```

のように記述しておくと便利です．こうしておけば

```
$ ssh -X s212601@dover.eps.s.u-tokyo.ac.jp
```

とわざわざ入力しなくても

```
$ ssh -X dover
```

と入力するだけで済むようになります．また edu や www-geoph にログインする場合も dover に一旦 SSH する必要はなく，直接

```
$ ssh -X edu01
```

とするだけで済みます．SCP や SSHFS[*3]にも有効なのでぜひ設定しましょう．

1.4 公開鍵認証

~/.ssh/には config 以外のファイルも配置されます．その中で最も重要なものが，鍵ファイルです．

これまで皆さんは，SSH で遠隔ログインする際にパスワードの入力を求められてきました．これは「背後に人が立っていてパスワードを盗み見られる」とか「パスワードが (暗号化されているとはいえ) インターネットを通ってしまう」とかいった脆弱性があります．そこで，これを解決する方法の一つとして公開鍵認証[*4]が広く用いられています．

設定方法は次の通りです．まず，ローカルホストで /.ssh/へ移動し，

```
/.ssh$ ssh-keygen
```

を実行します．すると

```
Generating public/private rsa key pair.
Enter file in which to save the key (/home/xxx/.ssh/id_rsa):
```

と表示されるので，鍵の名前を設定します．未入力で進むと id_rsa という名前が自動的に設定されます．このテキストでは鍵の名前は id_rsa とします．

次に，

```
Enter passphrase (empty for no passphrase):
```

と表示されるため，鍵を開くためのさらなるパスワードを設定します．空でも構いませんが，その場合は盗難等に気をつけましょう．入力すると確認されるので，同じ文字列を再度入力します．

すると，~/.ssh/に id_rsa というファイルと id_rsa.pub という 2 つのファイルが作成されます．見ての通り，id_rsa.pub が公開鍵ですので，これをサーバーに転送して使います．

```
ssh-copy-id -i ~/.ssh/id_rsa.pub username@hostname
```

[*3] 本テキストでは説明しません．リモートマシン上のファイルをローカルのファイルと同じように扱えるため便利なので中〜上級者は使ってみてください．

[*4] 暗号化に用いる鍵と復号化に用いる鍵が異なっているもの．この場合鍵ペアのうち暗号化に用いるものは公開しても問題ない．

とすれば公開鍵がサーバーに転送され適切に配置されます.

最後に，鍵認証で接続する設定を行います．ssh コマンドのオプションを用いて

```
ssh -i id_rsa username@hostname
```

として接続することも可能ですし，~/.ssh/config に

```
Host asano
        HostName asano.eps.s.u-tokyo.ac.jp
        User s212601
        ProxyCommand ssh -X dover nc %h %p
        IdentityFile ~/.ssh/id_rsa
        IdentitiesOnly yes
```

のように IdentityFile 行を追加することによって鍵を指定することも可能です.

1.5 練習

自分が使用している edu 以外の edu に SSH でログインしてログアウトしてください．dover に公開鍵認証を設定してください.

2 ファイル転送

遠隔ログインを使うようになると，ただログインするだけでなく，ファイルの転送もしたくなります.

2.1 SCP

ファイルの転送を暗号化するには，SCP (Secure CoPy) [*5]を用います．SSH を利用して通信を行う方式なのでセキュアです．dover や演習室の端末と自分のパソコン間でファイルをやり取りする場合にはこれを使うことになるので，使用頻度はかなり高いと思います．使い方は以下の通りです.

ローカルからリモートにファイルを送る場合には

```
$ scp filename1 [username@]hostname:filename2
```

[username@] となっているのは，ユーザ名がローカルとリモートで同じ場合に username@ を省略できるということで，SSH の時と同様です．例えば，手元にあるレポート課題を dover に送りたい時は

```
$ scp report.pdf dover.eps.s.u-tokyo.ac.jp:enshu_report.pdf
```

[*5] 2019 年 4 月より OpenSSH 公式より SCP は非推奨と宣言されています．本講義では cp コマンドとの類似性から引き続き SCP を用いてファイル転送の説明をしますが，余裕のある方は代替である sftp や rsync を調べてみてください.

のようになります.

反対にリモートからローカルにファイルをとってくる場合には

```
$ scp [username@]hostname:filename1  filename2
```

とします．さらに，-r オプションをつけると，ディレクトリのコピーもできます．recursive（再帰的）の r です.

いずれにせよ,「送るもの」を先に指定してから,「送り先」を指定します．前回扱った cp コマンドとよく似ていますね.

2.2 練習

1. hoge1.dat というファイルを dover に送り，hoge2.dat という名前で保存してください.
2. 1. で送ったファイルを edu に取ってきて，hoge3.dat という名前で保存してください.

2.3 WGET

例えば，`http://www.eps.s.u-tokyo.ac.jp/access.html` というファイルをダウンロードしたいとき，どのような方法があるでしょうか．GUI が好きであれば Firefox で右クリックして保存しても良いですが…CUI も使えるようにして欲しいですね….

このファイルは地球惑星科学専攻の場所のアクセスマップです．この地惑専攻のウェブサーバには皆さんのアカウントはありませんから SCP は使えません．このような時は **wget** コマンドを利用すると良いでしょう．wget は Web からファイルをダウンロードするコマンドで，HTTP・HTTPS・FTP のプロトコルに対応しています．上の例だと,

```
$ wget http://www.eps.s.u-tokyo.ac.jp/access.html
```

のようにして使います．“-x”オプションを用いれば，ディレクトリ構造を保ってダウンロードできます．つまり,

```
$ wget -x http://www.eps.s.u-tokyo.ac.jp/access.html
```

とすると，カレントディレクトリに www.eps.s.u-tokyo.ac.jp というディレクトリができ，その中に access.html が保存されます.

また，“-r”オプションを用いれば，指定したページに含まれるリンク先を再帰的に取得できます.

2.4 練習

`http://www-geoph.eps.s.u-tokyo.ac.jp/~s122621/kadai/index.html` とそのページに書かれているリンク先を全てダウンロードしてください.

3　パッケージマネージャ

UNIX にソフトウェアを追加でインストールする方法には複数ありますが，最も簡単なものはパッケージマネージャを使うものです．eduvm には標準で apt(Advanced Package Tool) がインストールされており，Mac では Homebrew が事実上の標準パッケージマネージャです．また，Python では pip というパッケージマネージャが使われており，Anaconda を使って Python をインストールした場合は conda という pip より高機能なパッケージマネージャもインストールされます．

他にも様々なパッケージマネージャが存在しますが，それらに共通する機能は次の通りです．

- インストールされているパッケージを把握する．
- オンラインのサーバーと，パッケージの情報を同期する．
- インストールされているパッケージをアップデートする．
- 新しいパッケージをダウンロードし適切なディレクトリに配置する．
- パッケージ間の依存関係を解消する．

これらの機能により，我々はほぼ自動で新しいプログラムをインストールし，かんたんにシステムを最新に保つことができます．

3.1　apt

前述の通り，eduvm には apt というパッケージマネージャが標準でインストールされています．自分のパソコンが Windows で，WSL 上に Ubuntu をインストールしている場合も同様です．これを使って新しいソフトをインストールするには次の通りの手順を踏みます．

まず，

```
$ sudo apt update
```

として，パソコンが持っているパッケージの情報をオンラインの最新のものへと更新します．このときパスワードを要求されますが，入力するのは今ログインしているユーザーのパスワードです．(eduvm のデフォルトなら，ユーザー名 chikyu のユーザーのパスワードを入力します．) asano や sakura や dover でこれを行おうとすると “You are not in sudoers.” と怒られます．これは「あなたは apt でこのシステムのプログラムを変更する権限がありません」ということを意味しています．もしこれらのマシンにインストールしてほしいソフトウェアがある場合は admin210 までお問い合わせください．

次に，

```
$ apt search 欲しいソフトウェアのキーワード
```

としてパッケージ名を検索します．すでにパッケージ名を知っている場合は飛ばしても構いません．

最後に，

```
$ sudo apt install パッケージ名
```

としてインストールします．

他にも様々な機能がありますので，興味がある方は man ページを見るなりインターネット検索を駆使するなりしてみる

といいでしょう．

3.2 Homebrew

macOS の場合は標準ではパッケージマネージャはインストールされていません．そのため自力でパッケージマネージャをインストールすることになるのですが，この際にほとんどすべての人が使っているものが Homebrew です．`https://brew.sh/index_ja` を参考にインストールしましょう．

使い方は大まかには apt と似ています．ただし，パッケージのリストは常にオンラインから最新のものを取得するようになっていますので，都度 update する必要はなくなっています．

つまり，まず

```
$ brew search 欲しいソフトウェアのキーワード
```

としてパッケージ名を検索し，次に

```
$ brew install パッケージ名
```

としてインストールすればよいことになります．

ここで，`brew install` の前に `sudo` が無いことからわかるように，Homebrew では管理者権限が無くともユーザーエリアにパッケージをインストールできるようになっています．559 室の edu も例外ではありませんので，皆様で好きなソフトウェアをインストールしてみてください．

4 基本的なコマンドとオプション

ターミナル上で次のようなコマンドを利用して，ファイル操作に慣れましょう．

表 1 基本的なコマンド

コマンド名	語源	概要	重要オプション
`cd`	Change Directory	カレントディレクトリの移動	-
`pwd`	Print Working Directory	カレントディレクトリの表示	
`ls`	LiSt	ファイル一覧の表示	-l -a -d -h -F
`cp`	CoPy	ファイルの複製	-R -f
`mkdir`	Make Directory	ディレクトリの作成	-p
`mv`	MoVe	ファイルの移動 / リネーム	
`rm`	ReMove	ファイルの削除	-r -f -i
`rmdir`	ReMove DIRectory	ディレクトリの削除	-p

前回の講義でオプションについても習いました．例えば，ターミナルで`ls -R`と入力してみましょう．どうなりましたか？ これはどんなオプションでしょうか．`man ls`として，確認してみましょう．また，一度に複数のオプションも使用できましたね．`ls -la`と入力してみてください．これは`ls -a -l` と同じ結果となるはずです．コマンドの機能を忘れてしまったときは`man <コマンド名>`とするか，それでもわからなければ`firefox &`などでブラウザを呼び出して Google などで検索してみましょう．

```
合計 108
drwxr-xr-x  12 ueda student  4096  4月 10 14:24 ./
drwxr-xr-x 305 root root    12288  3月 31 17:59 ../
-rw-------   1 ueda ta         51  4月 10 14:24 .Xauthority
-rw-r--r--   1 ueda ta       5837  3月 31 17:47 .Xresources
-rw-r--r--   1 ueda ta        543  3月 31 17:47 .aliases
drwx------   2 ueda ta       4096  4月  5 18:20 .anthy/
-rw-------   1 ueda ta       1571  4月  5 18:49 .bash_history
-rw-r--r--   1 ueda ta        416  3月 31 17:47 .bash_profile
-rw-r--r--   1 ueda ta        973  3月 31 17:47 .bashrc
drwx------   3 ueda ta       4096  4月  5 16:07 .dbus/
-rw-r--r--   1 ueda ta       4120  3月 31 17:47 .emacs
drwx------   3 ueda ta       4096  4月  5 17:49 .emacs.d/
drwxr-xr-x   5 ueda ta       4096  3月 31 17:47 .fluxbox/
drwx------   2 ueda ta       4096  4月 10 14:24 .gconf/
drwxr-xr-x   3 ueda ta       4096  4月  5 16:07 .local/
drwxr-xr-x   3 ueda ta       4096  4月  5 18:40 .texmf-var/
drwx------   3 ueda ta       4096  3月 31 17:47 .uim.d/
-rw-r--r--   1 ueda ta       5230  3月 31 17:47 .vimrc
-rw-------   1 ueda ta        577  4月  5 18:47 .xdvirc
```

ここに載せている以外にも様々な隠しファイル (ピリオドで始まるファイル) や前回までに作成したファイルが見えると思います．`ls -l` の出力に関しては前回 (UNIX その 1) の講義で習いましたが，ちゃんと意味が理解できるでしょうか．一列目にある文字はパーミッションを表していますが，その意味を忘れてしまっていたら前の資料を確認しておいてください．また，隠しファイル (.bashrc など) は主にユーザ毎の設定ファイルということでした．隠しファイルの設定に関しては，後ほどもう少し詳しく説明したいと思います．

4.1 練習

いくつかのコマンドとオプションを使って，遊んでみましょう．例えば，`sl`としてみましょう．これは何のためのコマンドでしょうか？ このコマンドはインストールされていますか？ インストールされていない場合はパッケージマネージャを使ってインストールしてみてください．そして，`man sl`としましょう．また，そこで見つけたオプションを試してみましょう．`cal -jy`とかもやってみましょう．

また`du`は便利なコマンドの 1 つです．`du -sh`として，それぞれのオプションの意味を`man du`で確認してみてください．また，`du -sh *`と入力してみましょう．

(時間が余った人は `finger`や`last`, `who` なども調べてみると面白いかもしれません)

また，ファイル操作に関するコマンドは非常に重要です．別の場所にあるディレクトリを自分のホームディレクトリにコピーするにはどうすればよいでしょうか？`/home2/mori2021/TA2021/`というディレクトリに `exercise` というディレクトリが置いてあり，その中には今日使うファイルが入っています．これを自分のホームディレクトリにコピーしてきてください．なお，ファイル・ディレクトリの作成・削除，移動の仕方などが分からない人は TA の人に聞きながら練習して

みてください.

5　リダイレクトとパイプ

さて次も前回習った話です．パイプは UNIX の哲学の根幹にあるという話でした．使えるようになっているでしょうか？ **標準出力をファイルに書き出したり，ファイルの中身を標準入力として読み込んだりする際に使うのがリダイレクト**，**標準出力を別の操作の標準入力として送り込むのがパイプ**でした．それぞれについて，前回の講義で簡単な例を用いて学習したと思います．ここでは，`head` と `tail` というコマンドを使って，パイプの使い方を少し練習してみましょう．

先ほどコピーしてきてもらったディレクトリに `students.txt` というファイルが入っていたはずです．ファイルの中には皆さんのログインネームと名前が学生証番号の順に並んでいます．これを開いてみましょう．まずホームディレクトリから`cd exercise`でディレクトリを移動して，`cat -n students.txt`としてみてください (`-n` は行番号を付加するオプションです).

```
1  s172601 Hiromasa Akadama
.
.
.
```

上のような出力が得られるはずです．ターミナルからはみ出して見づらい場合は，前回習ったようにパイプで `less` などに送りましょう．`cat -n students.txt | less`です.

さて，ファイルの中身を見ることができたら，今度は自分の名前の行だけ抜き出してターミナルに表示させてください．方法はいろいろとあるかと思いますが，ここでは `head` と `tail` というコマンドとパイプを用いてみましょう．`cat`，`head`，`tail` の基本的な使い方は Table 2 の通りです．オプションなどの詳しい情報は `man` で調べてください.

表 2　ファイルの中身を出力する基本的なコマンドとその使い方

コマンド名	出力	基本的な書式[*6]
`cat`	ファイルの中身	`cat <file>`
`head`	ファイルの先頭から指定した行数	`head -`*lines* `<file>`
`tail`	ファイルの末尾から指定した行数	`tail -`*lines* `<file>`

さて，解答編です．例えば，4 行目を抜き出したいとします．`head -4 students.txt`だと，上から 4 行が出力されてしまいます．`tail -20 students.txt`だと，下から 20 行です．自分の名前が 1 行目や最後の行にある人は良いですが，これでは任意の行を抜き出すことができません．そこで，`head -4 students.txt`の出力をながめてみましょう.

```
s172601 Hiromasa Akadama
s172602 Kosuke Ikehata
s172603 Kensuke Ishikawa
s172604 Sohei Ishizuka
```

この出力の一番最後の行を抜き出してやれば，`students.txt` の 1 行目のみを出力することができますね．最後の行を

抜き出すには`tail -1`としてやればよいので，あとはパイプで`head -4 students.txt`の出力を送ってやればよいわけですね．

```
~/exercise$ head -4 students.txt | tail -1
```

または少し長くなってしまうやり方ですが，`cat` も使って

```
~/exercise$ cat students.txt | head -4 | tail -1
```

としても同じ結果になるはずです．

機能としては，先頭から任意の行数を抜き出すコマンドと末尾から任意の行数を抜き出すコマンドがそれぞれ存在するだけですが，パイプを使って組み合わせることで任意の行を抜き出すことができるツールになるわけです．任意の行を抜き出すコマンドを作らず，簡単な道具の組み合わせによって必要な処理を実現させる．それぞれの簡単な道具は，パイプという装置を通して，必要かつ十分な機能を持っており，それが **KISS (Keep It Simple and Smart)** を実現しているわけです．これが，パイプが UNIX の哲学の根幹と言われるゆえんです．

さて，任意の行が取り出せたので，次は任意の列を取り出す方法を見てみましょう．

`cat students.txt | cut -f 1,3 -d " "`と打ってみましょう（ダブルクオーテーション” ”の間は半角スペースです）．すると

```
s172601 Akadama
s172602 Ikehata
.
.
.
```

のように，1 列目と 3 列目のみが表示されたはずです．ここで行った操作は，cut コマンドを使って半角スペースで区切られた（d オプション）1 列目と 3 列目を（f オプション）切り出したということです．cut コマンドは他にもいろいろな便利なオプションがついているので，調べてみるとよいでしょう．

5.1 練習

`students.txt` の 6 行目から 10 行を抜き出してみましょう．また，6 行目から 10 行目と 16 行目から 20 行目を `exercise.txt` というファイルに書き出してみてください．さらにそこから名前部分だけを `exercise2.txt` に書き出してみてください．リダイレクトの ‘`>`’ と ‘`>>`’ の違いに気をつけて，操作を行いましょう．

この程度の作業であれば，わざわざターミナル上で行わなくても Emacs などのエディタで十分だと思うかもしれませんが，本格的に計算を始めると，100 個，1000 個という規模のデータファイルを扱う機会も出てきます．Bash などのシェルを用いた，さらに高度な操作については 5 月後半の講義でシェルスクリプトというものを習います．特殊な文法などもありますが，基本はターミナル上でできることを組み合わせて作るプログラムのようなものになります．

コマンドというと何やら難しそうですが，イメージとしてはちょっとしたことができるプログラム群と考えれば良いわけです．`ls` はカレントディレクトリのファイルやディレクトリを表示してくれます．`head` や `tail` はファイルの先頭や末尾から指定した行だけ抜き出してくれます．パイプやリダイレクトを利用することで，処理を重ねて行わせて複雑なことをしたり，結果をファイルに書き出したりすることもできます．**「一つ一つはちょっとしたこと，しかし組み合わせること**

でいろいろなことができる」，この辺りは UNIX の哲学に関係する部分です．詳しくは前回のレジュメを読み直してみてください．

6　コマンド呼び出しの仕組み

6.1　コマンドはどこからくるのか？

さて，コマンドがちょっとしたことができるプログラムということは良いでしょうか？ そう考えると，ここで少し疑問が出てきます．Windows でも Mac OS X でも，アプリケーションなどのプログラムはアプリケーションが入っている場所までいって起動をしたり，ショートカットなどから起動したりしますね．しかし`ls`をしても分かるように，皆さんのホームディレクトリにコマンドに関係しそうなファイルはありません．一体コマンドというもの達はどこからどのように呼び出されているのでしょうか？ ここから少し UNIX のシステムの話になります．

まず，前回の講義に載っていた UNIX における代表的なディレクトリとその中身という図をもう一度見てみましょう．

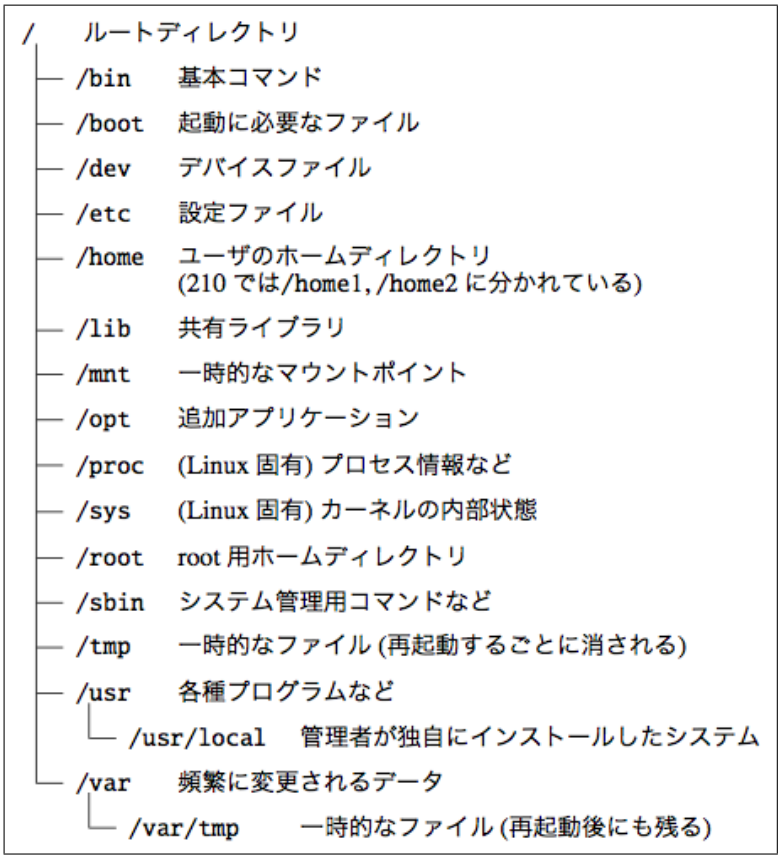

図 1　UNIX における代表的なディレクトリとその中身

Fig. 1 を見ると，基本コマンドは`/bin` にあるようです．`find /bin -name ls`としてみましょう．

```
~$ find /bin -name ls

/bin/ls
```

とお返事がいただけますね．どうやら/bin というディレクトリに ls が存在しているようです．head についても/bin にあるのでしょうか．同じように`find /bin -name head`としてみましょう．お返事が返ってこないのではないかと思います．こういうときは`whereis head`と入力してみましょう．

```
~$ whereis head

head: /usr/bin/head /usr/bin/X11/head /usr/share/man/man1/head.1.gz
```

というお返事が返ってくるかと思います．後ろにいろいろと付いているかと思いますが，いまは気にせずに流してください．どうやら head は/usr/bin にあるようですね．Fig 1 を見てみると，/usr は各種プログラムなどが入っているディレクトリです．ちなみに，`man whereis`とすれば分かりますが，whereis はコマンドのバイナリ・ソース・man ページの場所を示してくれるコマンドです．その他のコマンドについても，いくつか試してみましょう．

6.2 コマンドはどうやってくるのか？

さて, コマンドがどこにいるのかは分かってきました. どうやら ls は/bin というディレクトリにいて, head は/usr/bin にいるようです．試しに，`/bin/ls -l`と入力してみましょう．

```
~$ /bin/ls -l
```

すると，ls と全く同じ出力が得られるはずです．場所さえ知っていれば，このように使うことができるのは Windows や Mac OS X などのアプリケーションと同じですね.

しかし，先ほどいくつかのコマンドで確認したように，皆が同じ場所にいるわけではないようです．にも関わらず，我々はコマンドがどこにいるのかを全く知らないまま使うことができています．不思議ですね．この節では，コマンドが呼び出される仕組みについて説明します.

ヒントは，ログイン時に読み込まれるファイル/etc/profile にあります．起動時のファイル読み込みについては後ほど詳しく説明しますが，とりあえず今は中を確かめてみましょう．ターミナル上で`lv /etc/profile`と入力してみてください．何やら暗号めいた内容が表示されるかと思いますが，一つ一つの解読はそれぞれの興味にお任せします．各自調べてみたり，TA に聞いてみたりしてみてください．ここでは，先ほどからコマンドのありかとして現れている bin という単語を検索してみましょう．lv や less 内での検索には`/<search term>` ↵ と入力します (つまりスラッシュを打った後探したい言葉を入力する，n と打つと次の検索結果に移る)．この場合，`/bin` ↵ です．最初の方の PATH=... と書かれた箇所を探してください.

```
if [ "`id -u`" -eq 0 ]; then
  PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
else
  PATH="/usr/local/bin:/usr/bin:/bin:/usr/games"
fi
```

このような記述が見つかりましたか？ さらにこのような記述が最後の方にあったと思います.

```
export PATH
```

export は bash において環境変数を設定するコマンドです．そしてこの**環境変数** PATH が，使えるコマンドの場所を指定しています．つまり，いちいちそのコマンドの置き場を指定しなくても，PATH で指定されているディレクトリについては自動的に探してくれるということです．“:” で区切られて，いくつかのディレクトリが設定されていますが，前に書かれているものほど**優先的**に呼び出されます．つまり，上の例では，仮に/usr/local/bin/ls と/bin/ls が存在していた場合，普通に ls と入力したときには前の方にある/usr/local/bin/ls が実行されるということです．環境変数 PATH において設定され，そのディレクトリのコマンドなどをパスの指定なしに利用できる状態のことを**パスが通っている**というような言い方をします．CUI に限らず，計算機を便利に利用するためには，パスを通すという概念が非常に重要です．現在パスが通っているディレクトリがどうなっているかは，環境変数 PATH を表示する以下のコマンド

```
~$ echo $PATH
```

を実行すれば確認できます．すると，(ちょっと見づらいかもしれませんが) 確かに/usr/local/bin:が入っていることがわかると思います．先ほどの ls と head について，上の PATH で確認してみましょう．今は下段の方の PATH の設定が効いています．確かに/bin にも/usr/bin にもパスが通っていますので，ls も head も取り立ててパスの指定をすることなく利用できていたわけですね．

ではここで少し試してみましょう．先ほどの~/exercise ディレクトリに line.sh というコマンドを用意しておきました．とりあえず拡張子.sh については気にしないでください．これは cat コマンドなどで中をみてみれば何となくわかると思いますが，要するに<ファイル名>の<整数> 行目を抜き出してください，という意味です．適当に試してみましょう．

```
~$ ~/exercise/line.sh 10 ~/exercise/students.txt
```

お返事がありましたよね？ でも

```
~$ line.sh 10 ~/exercise/students.txt
```

とだけしても，そんなコマンドないよって怒られるはずです．たとえ今いるディレクトリに line.sh が入っていてもやはり実行できません．そこで

```
~$ PATH=$PATH:~/exercise

~$ line.sh 10 ~/exercise/students.txt
```

としてみましょう．今度は怒られないはず．~/exercise にパスを通すことにより（これまでのパスに~/exercise を追加）line.sh コマンドが使えるようになったわけです．

6.3 環境変数について

環境変数について，少しだけ説明しておきます．環境変数とは，その名の通り，端末の環境に関する情報です．ターミナルで<u>env</u> (environment の略) と入力してみて下さい．

```
MANPATH=/usr/local/intel/idb/man:/usr/local/intel/ifc/man:/usr/local/intel/icc/man:/usr/local/
intel/idb/man:/usr/local/intel/ifc/man:/usr/local/intel/icc/man:/usr/local/intel/idb/man:/usr/
local/intel/ifc/man:/usr/local/intel/icc/man:/usr/local/intel/icc/man:/usr/local/man:/usr/loca
```

```
l/share/man:/usr/share/man
INTEL_LICENSE_FILE=/usr/local/intel/icc/licenses:/opt/intel/licenses:/home2/koshin/intel/licen
ses:/Users/Shared/Library/Application Support/Intel/Licenses:/usr/local/intel/ifc/licenses:/op
t/intel/licenses:/home2/koshin/intel/licenses:/Users/Shared/Library/Application Support/Intel/
Licenses:/usr/local/intel/icc/licenses:/opt/intel/licenses:/home2/koshin/intel/licenses:/Users
/Shared/Library/Application Support/Intel/Licenses:/usr/local/intel/ifc/licenses:/opt/intel/li
censes:/home2/koshin/intel/licenses:/Users/Shared/Library/Application Support/Intel/Licenses:/
usr/local/intel/icc/licenses:/opt/intel/licenses:/home2/koshin/intel/licenses:/Users/Shared/Li
brary/Application Support/Intel/Licenses:/usr/local/intel/ifc/licenses:/opt/intel/licenses:/ho
me2/koshin/intel/licenses:/Users/Shared/Library/Application Support/Intel/Licenses
TERM=xterm-256color
SHELL=/bin/bash
.
.
.
```

このようにさまざまな環境変数があります．これらの環境変数が，現在皆さんが使っている端末において定義され，端末の環境を規定しているわけです．今後色々なアプリケーションをインストールするとき，インストールガイドなどで「○○という環境変数を設定するように」という指定がなされることがあると思います．そのときはここでやったことを思い出してみるとよいでしょう．また，興味のある方は，それぞれの意味について調べてみると良いと思います．

7 起動のはなし

さて，話を本筋に戻しましょう．これまでの話をまとめると，**コマンドは主に /bin や /usr/bin などのディレクトリ内にしまわれており，それらのディレクトリにパスを通すことによって，パスを指定しなくてもコマンドが利用できる**という話でした．**環境変数 PATH は /etc/profile の中で定義されていました**．ここまでは良いですね．

少しずつ，端末の中がどうなっているのかが分かってきましたね．しかし，まだ大きな疑問が残っています．本来パスの設定などは個人設定として，自分で自由に追加したり削除したりしたいところです．でも先ほどの/etc/profile というディレクトリは，いかにも個人で変えられそうもないところにあります．実際にemacs /etc/profile &などとして何か書きかえようとしてみればわかります．権限がないと怒られるはずです．ではどのようにすれば自分で好きなように個人用の環境変数を設定すればよいのでしょうか．それを理解してもらうために，この章では，端末の起動の仕組みについて説明します[*7]．

7.1 ログイン

もう問題無いかと思いますが，ユーザ名とパスワードを入力して，端末にログインをします．ここでユーザの認証が行われて，それぞれの環境に向かい入れてもらえます．ちなみに現在の 210 はパスワード入力時に * が表示されると思いますが，UNIX 系の端末ではパスワードの流出を防ぐため，何も表示されないこともあります．ネットワークの回で学んだ ssh を用いて，外部ログインをすでに試した人は，ログインするときパスワード入力のとき何も表示されないことに気づいた

[*7] 使っているシェルやディストリビューションによって，異なる部分がありますが，ここでは 559 の環境を想定し，典型的な一例を示します．

と思います.

7.2 /etc/profile の読み込み

ログインに成功すると，bash が起動し，/etc/profile が読み込まれます．これは全てのユーザに共通の設定ファイルであり，ログイン時に一度だけ読み込まれます.

7.3 ∼/.profile の読み込み

/etc/profile の次に読み込まれるのが，ホームディレクトリにある.profile[*8]です．ホームディレクトリに存在することからも分かるように，このファイルは個人用の設定ファイルです．.profile も**ログイン時に一度しか**読み込まれません．/etc/profile との違いは，**全ユーザに共通か，個人にのみ適用されるか**，という点です．つまり，全ユーザについて設定しておくべきことは/etc/profile に書き込まれます.

7.4 ∼/.bashrc の読み込み

次に読み込まれるのが，.bashrc です．.bashrc は**シェルを起動する毎 (ターミナルなどを新たに立ち上げる毎)** に読み込まれます．もちろん，これも個人毎の設定ファイルになります．.profile とのもっとも大きな違いは，**.bash_profile がログイン時に一度だけ読み込まれるのに対し，.bashrc はシェルを起動するたびに読み込まれる**という点です．つまり，一度だけ設定すればよいことは.profile に書かれ，シェルを起動する毎に設定すべきことが.bashrc に書かれます．先ほどの PATH の設定などは.profile と.bashrc のどちらに書くこともできます. ただ，.bashrc の中で設定した方がターミナルを立ち上げ直すだけですぐに変更が反映される (いちいち再ログインしなくてよい) ので，そちらの方が一般的です．皆さんの場合は

```
# PATH etc の設定
export PATH=/usr/local/intel/idb/bin:/usr/local/intel/ifc/bin:
/usr/local/intel/icc/bin:/usr/local/bin:/usr/bin:/bin:/usr/games
```

などとしておいて[*9]これに適宜足していくのがよいかと思います．また，一度に書かずに

```
# PATH etc の追加
export PATH=$PATH:<追加したいパス>
```

などとして，これまでのパスに順次追加することもできます.

その他にも，/etc/.profile.d というディレクトリや，~/.bash_login，~/.bash_logout といったファイルが存在する場合もありますが，概ねの流れは 3.1 節から 3.4 節のような感じです．Fig 2 に流れを示しておきます.

また，予想できるかと思いますが，.bash_history はコマンドのヒストリ機能を助けるためのファイルです．less .bash_historyで中をのぞいてみると，皆さんがこれまでに入力したコマンドが並んでいるはずです.

[*8] .bash_profile のような名前になっている場合もある.

[*9] 見やすさのために改行していますが，実際にはこの書き方はできません．1 行で書くか，改行するところにバックスラッシュを付ける必要があります.

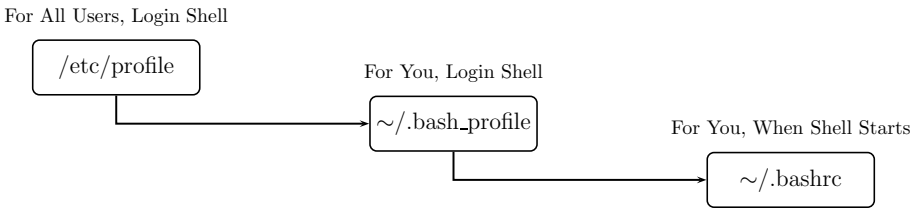

図 2　ログイン後に読み込まれるファイル

7.5　ファイルの中をのぞいてみよう

さて，今までに出てきたファイルの中身を少しのぞいてみましょう．
まず，/etc/profile をのぞいてみましょう．lv /etc/profileとしてください．

```
 1 # /etc/profile: system-wide .profile file for the Bourne shell (sh(1))
 2 # and Bourne compatible shells (bash(1), ksh(1), ash(1), ...).
 3
 4 if [ "`id -u`" -eq 0 ]; then
 5   PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
 6 else
 7   PATH="/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games"
 8 fi
 9 export PATH
10
11 if [ "$PS1" ]; then
12   if [ "$BASH" ] && [ "$BASH" != "/bin/sh" ]; then
13     # The file bash.bashrc already sets the default PS1.
14     # PS1='\h:\w\$ '
15     if [ -f /etc/bash.bashrc ]; then
16       . /etc/bash.bashrc
17     fi
18   else
19     if [ "`id -u`" -eq 0 ]; then
20       PS1='# '
21     else
22       PS1='$ '
23     fi
24   fi
25 fi
26
```

```
27 # The default umask is now handled by pam_umask.
28 # See pam_umask(8) and /etc/login.defs.
29
30 if [ -d /etc/profile.d ]; then
31   for i in /etc/profile.d/*.sh; do
32     if [ -r $i ]; then
33       . $i
34     fi
35   done
36   unset i
37 fi
```

シェルスクリプトについては，今回は触れませんが，意味の分かる部分も出てきたかと思います．4 行目の if はプログラミングをやったことがある人なら分かるかもしれませんが，条件によって異なる処理を行う際に用いられる構文です．その後の id という部分は，man idとしてみましょう．どうやらユーザの情報を参照して，それによって処理を変えているようです．ユーザの種類によって，PATH の値を変更しているようです．PS1 に関しては，後でもう少し詳しく説明します．というように，何となく意味が分かっていくかと思います．繰り返しになりますが，9 行目の export というのは，シェルスクリプトの中で定義したシェル変数を環境変数にするという操作をしています．少しややこしいですが，export をしないと，代入された値などはそのシェルの中でしか定義されたことにならず，export が行われることで環境変数として，そのプロセス (ここでは/etc/profile が実行されたシェル内) から起動されるプロセス (例えば，新たに立ち上げたシェル) にも値が引き継がれるということになります．

次は~/.bash_profile の中をのぞいてみましょう．先ほどと同様，less /.bash_profileとしてください (文字化けするときはless /etc/profile | nkf -w としてみてください)．

```
 1 # ---- language-env DON'T MODIFY THIS LINE!
 2 # .bash_profile は、ログイン時に実行される。
 3 if [ -f ~/.bashrc ]
 4 then
 5   # ただし、すでに .bash_profile が .bashrc を実行していたら、
 6   # 重複しては実行しない。
 7   if [ -z "$BASHRC_DONE" ]
 8   then
 9     . ~/.bashrc
10   fi
11 fi
12 # ---- language-env end DON'T MODIFY THIS LINE!
```

コメントの通り，.bash_profile はログイン時に実行されます．3 行目は.bashrc が存在するかを確かめる条件文です．コメントを付けているので大体分かるかと思いますが，コメント (# 印) の 5 行目と 6 行目に書かれている内容が

7 ～ 11 行目に書かれているという感じです．しかし実はこの部分は役に立っていません．それについては後ほど．

内容としては，.bashrc を実行してね，ということですね．皆さんの.profile は条件付き (.bashrc の有無や，すでに実行されたかどうか) で実行するように調整されていますが，人によっては，.profile の記述は，

```
source ~/.bashrc
```

のみという人もいます．

ここまでがログイン時に設定されることです. つまり，もし.bashrc という個人設定ファイルがなくても，/etc/profile の中に書かれていたことは設定された状態になるということです．PATH や umask などの基本的な部分はここまでですでに定義されていますね．.profile を見れば分かるように，別に.bashrc が無くても，端末が起動しないわけではありません．なので，自分が新たにパソコンを使い始める時に.bashrc がないからといって焦る必要はありません．.bashrc は基本的には自分で作るものなので，もしなければ新たに作ればよいというだけです．.bashrc の中での設定は，端末を便利に使う上で非常に重要なものばかりです．それでは中をのぞいてみましょう．.bashrc はそれなりに内容が多いので，今回はその中の一部を説明します．行数を利用して説明しますので，cat -n .bashrc | lvなどとしてください[*10]．その後，何となく分かるような分からないような記述が続きますが，最後までみても先ほどの BASHRC_DONE という環境変数が現れないことが分かると思います．.bashrc を起動したか否かを判断するための変数なのに.bashrc の中に記述がない．これでは.profile でわざわざ重複を避けている意味がありませんね．役に立っていないと言ったのはそういう理由です．気になる人は

```
# .bash_profile で使う。
 BASHRC_DONE=1
```

などと足しておけばいいでしょう (別にこれをやらなくても何も問題はありません).

11 ～ 20 行目の PS1 については，後の練習で利用します．下に進んでいくと，27 行目からは，aliases という名前がみえると思います．

```
28  if [ -f ~/.aliases ]; then
29      .  ~/.aliases
30  fi
```

エイリアスというのは覚えていますか？.aliases に関しては後で説明しますが，どうやらホームディレクトリ下の.aliases の存在を確認し，読み込んでいるようです．

また 22 行目に書かれている/etc/bash_completion はシェルの補完機能[*11]を拡張するためのファイルです．補完機能の拡張は端末を便利に使うためには不可欠です．興味があれば，中をのぞいてみてください．

その他，気になる部分は各自調べてみたりしましょう．おそらく，最初に自分で設定ファイルを作る際は，ネット上に転がっているファイルを拾ってきて，カスタマイズしていったり，便利そうな機能をコピペしていくという形になるのではないかと思います．その時のためにも暗号解読のスキルを身に付けておきましょう．

[*10] もう意味は分かりますよね！

[*11] Tab を押すとコマンド名やらファイル名やらが補完される機能．ちゃんと使っていますよね．使わないと非常に不便です．

7.6 練習

.bashrc に現れる環境変数を少しいじってみましょう．11 行目からの PS1 という変数を使って，遊んでみます．まずは，.bashrc の中身を確認しておきます;

```
11  # プロンプト。man bash 参照
12  if [ "$TERM" = "dumb" -o "$TERM" = "emacs" ]; then
13    PS1='\w\$ '
14  else
15    if [ "$UID" = "0" ]; then
16      PS1='\[\e[41m\]\w\$\[\e[m\] '
17    else
18      PS1='\[\e[7m\]\w\$\[\e[m\] '
19    fi
20  fi
```

PS1 はプロンプト (コマンドを打ち込む前に表示される印) を定義する変数です．どこのことを言っているのか分からない人もいると思いますので，とりあえずターミナル上で次のように入力してみましょう．‘=’ の前後に空白などは入れないでください．> の後ろには空白を 1 つくらい入れておいた方が，おそらく幸せです．

```
~$ PS1='> '
```

PS1 とは，これによって形が変わった場所のことです．良いですか？ これでは変で使いづらいとなったら

```
> source .bashrc
```

とすれば元に戻せます．注意して欲しいのは，「PS1」という変数は export されていないので，環境変数ではないということです．このことは

```
~$ env
```

としてみても，PS1 は現れないということから確認できます．

ここからは少し遊んでみましょう．以下の例を実行してみて下さい．また，Table 3 にいろいろな情報の参照の仕方をまとめました．いろいろと遊んでみましょう．

```
~$ PS1='\h '
```

```
~$ PS1='\h \u \n \d \t '
```

元祖ニコちゃんマーク．

```
~$ PS1=':-) '
```

冷や汗．

```
~$ PS1='^^; '
```

他にも色々バリエーションはあると思います．色なども設定できます．

表 3　PS1 で利用できる情報とその表記の例

表記	意味
\h	ホスト名
\u	ユーザ名
\d	日付
\t	時刻 (24 時間制)
\T	時刻 (12 時間制)
\w	カレントディレクトリ (フルパス)
\W	カレントディレクトリ名
\n	改行

8　設定ファイルのいろいろ

今までに見てきた設定ファイルは，主にシェルに関わる設定ファイルでした．そのほかにもさまざまな設定ファイルが存在し，環境やアプリケーションのカスタマイズが行えるようになっています．

8.1　.bashrc

先ほどから何度も出てきているファイルです．中身は先ほども見ましたので，ここでは省略します．

8.2　.aliases

先日の講義でエイリアスという機能を習いました．良く使うコマンドやオプションなどを便利に使うためのショートカットのようなものでしたね，.aliases は，旧 edu では先ほど見た.bashrc の最後の方で読み込まれていました.[*12]中をのぞいてみましょう．しつこいようですが，lv .aliasesです．

```
alias md='mkdir'
alias rd='rmdir'
if [ "$TERM" = "dumb" -o "$TERM" = "emacs" ]; then
  alias ls='ls -F'
else
  alias ls='ls -F --color=auto'
fi
alias lf='ls -F'
alias la='ls -a'
alias ll='ls -l'
```

[*12] bashrc 内に直接 alias が書き込まれている場合もあるようです．見当たらない場合は新規作成するか bashrc に直接書き込みましょう．

```
alias l.='ls -ld .*'
alias rm='rm -i'

alias   ..='cd ..'

alias   ggre='firefox http://www.google.co.jp'

alias grep='grep --color'
alias a2ps='a2psj'
alias wl='emacs -nw -f wl'
alias kterm='kterm -bg gray30 -fg white -cr yellow'

if [ -x /usr/bin/xdvi-ja ]; then
  alias xdvi='xdvi-ja'
fi

function xtitle() {
  /bin/echo -e "\033]0;$*\007\c"
}
```

今回はややこしい部分には触れませんので，エイリアスで定義されているコマンドをいろいろと使ってみましょう．`la` や `ll` などはすでに使っている人もいるかもしれません．よく使うオプションはエイリアスで定義しておくと便利です．また，`mkdir` のエイリアスとなっている `md` のように，少し長めのコマンドなどにもエイリアスを使うと便利です．一つ上のディレクトリに移動する`..` なども使い慣れると必須になります．先ほどの `line` コマンドもエイリアスの中で定義してやれば問題なく使えます．

エイリアスの意味や効用は分かりましたか？

8.3 .xsession

皆さんが今使っている Linux は，操作こそ CUI ですが，ログインの仕方やログイン後の画面などは Windows や Mac OS X などとそれほど違いを感じられなかったのではないかと思います．元祖の UNIX は完全にコマンドラインの操作のみで，画面全体にターミナルが表示されているような状態なのですが，皆さんが使っているのは，グラフィカルログインという形式で，違和感なくログインでき，マウスによる操作もできますし，Windows のようにウィンドウがぽんぽんと現れてくれます．これを実現しているのが X Window System と呼ばれるものです．単に ‘X (エックス)’ と呼ばれることもあります．

試しに，`Alt + Control + F3` を同時押ししてみましょう．画面が変わったのではないかと思います．それが X を使っていない状態です．元に戻すには，`Alt + Control + F7` とします．また，テキストログインした状態からでも `startx` とすることで X を起動することが可能です．

ここでは X について細かく説明することはしませんが，`.xsession` というのはこの X Window System に関する設定ファイルです．X が起動する際に実行されます．`.xsession` の他にも，x と付いているファイルは X Window System 用

のファイルであることが多いと思います．興味のある人は自分で調べてみましょう．

それでは，.xsession の中をのぞいてみましょう．cat -n .xsession | lvなどとして，指定する行の辺りをながめてみてください．コメントが付いていますので，参考にしてください．

いろいろとありますが，例えば 17 行目の applications という部分は，記述することで X が起動したときにアプリケーションを立ち上げることができます．今は全ての行にコメントアウトの記号 '#' が付いていますので，何も立ち上がらないはずです．その下の Console window や Terms and Editors も似たような感じです．42 行目の wallpaper は文字通り壁紙に関しての記述で，68 行目の Screensaver がスクリーンセーバーに関する記述です．

79 行目からの Window manager という部分は，X のユーザインターフェイスや見た目を選ぶことができる部分と考えれば良いでしょう．皆さんの使っているのは fluxbox と呼ばれるウィンドウマネージャです．世の中にはたくさんのウィンドウマネージャが存在し，それこそ Windows や Macintosh の画面と見間違えるような UI (ユーザインターフェイス) から，コマンドラインでの操作に特化したようなシンプルなデザインもあります[*13]．カスタマイズも GUI でできるものもあれば，エディタでしかできないものもあります．動作の重さもそれぞれです．

自分に合ったウィンドウマネージャを選べば良いのですが，そこに書いてあるすべてのウィンドウマネージャが 210 にインストールされているのかどうかはよく分かりません．ネットなどを調べていて，もし使ってみたいものが見つかったら，admin に相談してみることをお奨めします．

8.4 .emacs

そのまんまの名前ですが，Emacs の設定ファイルです．.bashrc がシェルの起動時に読み込まれるように，.emacs は Emacs の起動時に読み込まれるファイルです中をのぞいてみましょう．こちらも長いので，工夫して見てください．

ファイル内では言語環境などの設定がされていると思います．210 の環境では気づきませんが，23 行目，27 行目辺りで特に必要の無いメッセージは表示されないように設定されていたりもします．57 行目からは X で立ち上げたとき[*14]の設定が書かれており，メニューバーなどの表示・非表示もここで扱われています．88 行目からは色の設定がされています．120 行目からの記述は，TEX や Fortran を習うときに気にしておくとよいでしょう．

ちなみに，気づいたかと思いますが，先ほどまでのシェル関連のファイルと違い，.emacs ではコメントアウトの記号は ';' です．他のファイルと同列のように並べましたが，今まで見てきたファイルはシェルによって実行されるファイルだったのに対し，.emacs は Emacs Lisp という Emacs 用のプログラミング言語で書かれています．Emacs Lisp は Emacs Lisp で，語り出すと大変なことになりますので，興味のある人は調べてみて下さい．

9 設定ファイルをいじってみよう

中身が少し分かったところで，少しずついじってみましょう．ただし，特にシェルや X Window System 関連の設定ファイルは，下手にいじってしまうとログインできなくなる，という事態に陥りかねませんので，変更する場合は十分に気を付けましょう．また.emacs も Emacs 以外のエディタが使えない状況でおかしなことになってしまうと，修正することすらできなくなりますので，変更はある程度の知識や技術を身につけた後に行うのが無難かと思います．あとは自由にカスタマイズをして，便利に端末を使いましょう．

[*13] fluxbox はどちらかというとシンプルな部類に属します．

[*14] Emacs はターミナル上で立ち上げることも可能です．emacs -nwとしてみましょう．

9.1 .ssh

これまでに触れた以外にも SSH のオプションの数だけ設定ファイルに書き込める項目があります．たとえば，XForwarding などがあります．また，先の`~/.ssh/config` の例で説明のなく使っていた ProxyCommand も非常に便利です．「ssh 踏み台」等のキーワードで検索するよいでしょう．どのような項目が設定できるか調べてみて，便利に使えるようにカスタマイズしていきましょう．

9.2 .bashrc

自分で環境を構築していく場合以外は，このファイルを大きくいじることは無いと思います．そもそも，管理者が端末の環境に応じて設定している部分が多いので，中を見てみて，気になる部分を調べてみましょう．皆さんのホームディレクトリにある`.bashrc` は，コメントも付いていますので，参考にしてみてください．ちなみに，シェル起動中に`.bashrc` のような設定ファイルを読み込むには，ターミナル上で`source .bashrc`などとします．先ほどから何度か出てきましたね．

9.3 .aliases

エイリアスは，どんどん自分用にカスタマイズして，便利に使いましょう．書式は，

```
alias ll='ls -l'
```

といった具合です．“=” の前後に空白を入れないようにしてください．これは別に`.bashrc` や`.aliases` に書き込まなくてはいけないわけではなく，ターミナル上で設定できます．

```
~$ alias ls='ls -R'
```

```
~$ alias ls='sl'
```

とすれば，現在のシェル (ターミナル) が動いている間はこの設定が有効となります．しかし，これでは現在のシェルを終了させてしまうと設定が無効になり，次に起動させたシェルで再び入力をしなくてはなりません．`.bashrc` 内の設定も同様ですが，設定自体はターミナル上でコマンドを用いて行えるのですが，設定ファイルが読み込まれることで，このような手間を省いてくれているわけです．ちなみに，下側のやつは多分しない方がよい (`ls` が `sl` になってしまいます!!) です．alias をやり直したり，`.bashrc` を再読み込みするなどして，元に戻しておきましょう．

9.4 .emacs

Emacs は非常に拡張性に優れたエディタです[*15]．この Emacs の設定を行っているのが`.emacs` です．先ほど，中身をのぞきましたので，無難な部分として，初期メッセージの表示・非表示やメニューバーなどの表示，背景色などを変更してみましょう．初期メッセージに関しては ‘;’ でコメントアウトしてやればよいと思います．エディタで書き換えたら，Emacs を再起動してやってください．

ちなみに，`.emacs` の中に

```
(set-foreground-color "white")
```

[*15] メーラとしての機能やシェルモードなどの存在を考えると，ただのエディタとも言えませんが．弱点は若干重いことでしょうか．

```
(set-background-color "dark green")
(set-cursor-color "yellow")
(set-mouse-color "white")
```

と書き足すと，配色が黒板のようになります．

TEX やプログラミングをやり始めると，こういうのも便利かもしれません．

```
;; 対の括弧を明示する (for remark of the other paren)
(show-paren-mode t)
```

その他，いろいろなカスタマイズ例がネット上には転がっていますので，調べて，使ってみましょう．

10 まとめ

色々なことをしゃべりましたが，UNIX の仕組みについて重要な点をいくつかまとめておきます．

- コマンドはパスを通すことで，便利に使うことができている．
- パスは設定ファイルの中で定義されている．
- UNIX には様々な設定ファイルがあり，それらの構造を理解し，書き方を学ぶことでいろいろなカスタマイズができる．

UNIX と仲良くなれるような気になってきましたか？ コマンドラインの操作も起動などの仕組みも，実際には慣れによる部分が大きいのですが，ちゃんと順番を追っていけば，それほど意味不明なものではありません．

Windows などの普及している OS では，中で何が起こっているか，使っているだけではほとんど分かりません．逆に UNIX では，中で何が起こっているのかを理解しなくては，かなり意味不明なシステムになってしまいます．そのため，Windows などに慣れているユーザにとって，UNIX は難しい OS だと思われるのでしょう．しかし，中で起こっていることが分かっていれば，かゆい所に手が届くカスタマイズができるわけです．もちろん，それには相応の知識が必要になるわけですが，今日の講義でおぼろげな概観をつかんでもらえたなら，とても嬉しいです．

11 課題

今回の講義を踏まえて，いくつか課題を出します．締切は全て，1 週間後の 4 月 15 日 (木)0:00(JST) です．

提出方法

559 室の sakura または edu 上`/home2/mori2021/TA2021/`ディレクトリに，`s2126??/`という名前 (`s2126??`は自分のユーザ名) でディレクトリを作り，そこに各課題の解答ファイル (計 3 つ) を置いてください．全てについて，パーミッションは適切に設定してください[*16]．

[*16] 友達がカンニングできないように，また TA が採点できるように．他人がファイルを書き換えるのも防ぎましょう．

課題 0: ssh の設定

個人所有の PC から asano へワンタッチで ssh できるように./ssh/config を設定し，鍵認証を設定してください．設定したら公開鍵と config をコピーして提出してください．

課題 1: パイプ・リダイレクト

exercise ディレクトリの中に，earthquake.txt というファイルがあります．このファイルには 1949 年から 2016 年までの，震度別の地震の回数のデータが入っています[*17]．このデータの中から 1965 年から 2011 年までの期間，震度 1 および震度 7 のものを抜き出してください．結果は kadai1.txt に出力してください．

このファイルは半角スペースではなく、タブを使って列が区切られていることに注意しましょう。講義で扱っていないコマンドを使用しても構いません．

課題 2: PATH の設定

echo-sd という文字を装飾して表示するコマンドがあります[*18]．例えば，

```
~$ echo-sd '（ここに文字列を入力）'
```

とすると文字列が豪華になって返されると思います．といいましたが，パスが通っていないと実行できません．/home2/mori2021/TA2021/に echo-sd の実行ファイルがあるので，ここに PATH を設定して実行できるようにしてください．その後，適当な文字列を表示させて結果を kadai2.txt に記入してください．

課題 3: 端末のカスタマイズ

あなたが行ったカスタマイズについて教えてください．また，そのカスタマイズをして，嬉しかったことや便利になったことを教えてください．

実用的でない，ネタのようなものでも構いません．いいカスタマイズが思い浮かばなかったら，代わりに今回紹介しなかった便利なシェルコマンドを教えてください．（もちろん，いいカスタマイズをしたうえでシェルコマンドも教えてくれてもよいです）．

回答は kadai3.txt ファイルに記述してください．

[*17] http://www.data.jma.go.jp/svd/eqdb/data/shindo/index.php
[*18] http://github.com/fumiyas/home-commands/blob/master/echo-sd