ＵＮＩＸ
利用の手引き

明治大学　生田メディア支援事務室

目次

第１章　　UNIX の基本的な使い方

1.1　　UNIX を操作するにあたって

この章では、UNIX(Linux)を初めて体験する人を対象に、生田システムでの UNIX 操作の基本について説明します。

1.1.1　キー操作の表記について

UNIX の操作の中で特別な意味を持つ文字がいくつかあります。

その中には、キーボードのそれぞれのキーの上に印刷されている文字と、その文字を入力した時に画面上に表示される文字とが、一見すると同じものに見えないものがあります。

『¥』と
『\』

名前：エンマーク、バックスラッシュ

キーボード上の位置：『BackSpace』の左隣、右下『Shift』の左隣

この２つの記号はそれぞれ、日本語環境、英語環境で利用されるもので、表記は別々になっていますが、内部的には同じ文字コードが割り当てられた同じ文字となっています。

そのため、利用している環境や使用しているフォントによって、『¥』キーを押したにもかかわらず『\』と表示されたり、あるいはその逆であったりします。

どちらの表示でも UNIX の操作上は基本的に同じものなので、混乱しないように気を付けてください。

『~』

名前：チルダ

キーボード上の位置：『BackSpace』の２つ左

この記号は、画面上の表示位置が真ん中の場合と上寄りの場合とがあります。

入力する際には『Shift』キーを押しながら該当キーを押すことで入力できます。

『|』

名前：パイプ

キーボード上の位置：『BackSpace』の左隣（『¥』と同一キー）

この記号は、画面上では『¦』のように真ん中が少し切れて表示される場合もあります。

キーボード上でも同様に、繋がっている場合や、切れている場合があります。

入力する際には『Shift』キーを押しながら該当キーを押すことで入力をすることができます。

1.2　　ログインとログアウト

UNIX は、複数の利用者が同じコンピュータを共同で使う場合でも、利用者毎の資源や環境を個別に管理できるようになっています。

そのため UNIX は、現在、誰が利用しているのかを識別できるようなシステムを採用しています。

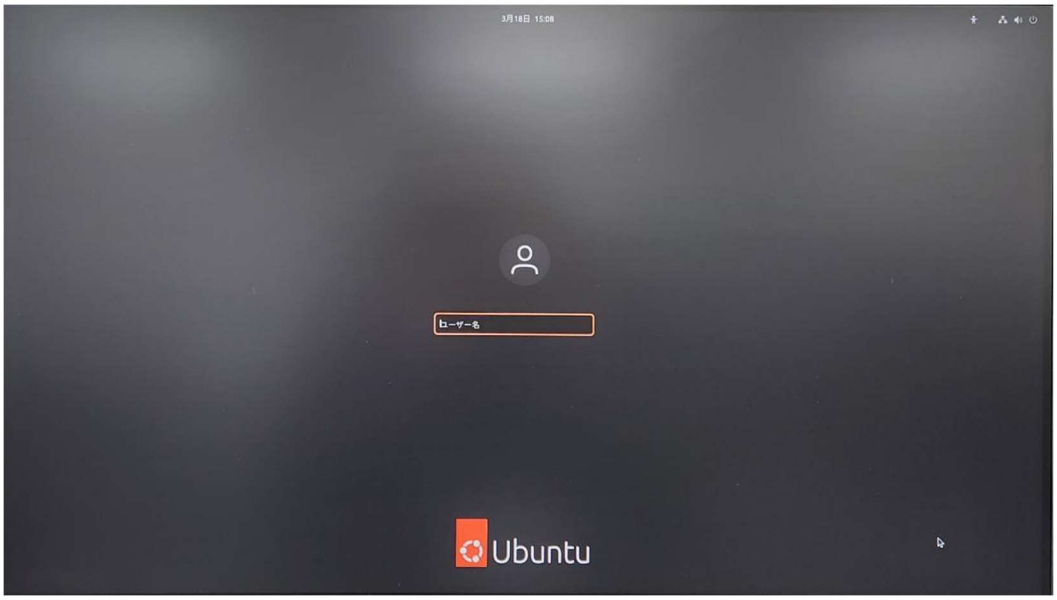
それに伴い、皆さんが UNIX を利用する際には**ログイン**、終了をする際には**ログアウト**という手続きが必要になります。

ログインは、個々の利用者に割り当てられた**ログイン名(ユーザーID)**と、利用者だけが知っている**パスワード**を入力することで行います。

それでは実際にログイン、ログアウトの操作を行ってみましょう。

- OS 選択画面
  コンピュータの電源をつけるとまず OS の選択画面になります。ここで、Linux(Ubuntu)を選択してください。

- ログイン画面（OS を選択してしばらく待つ）
  OS を選択した後、少し時間を置くと次の図のようなログイン画面になります。

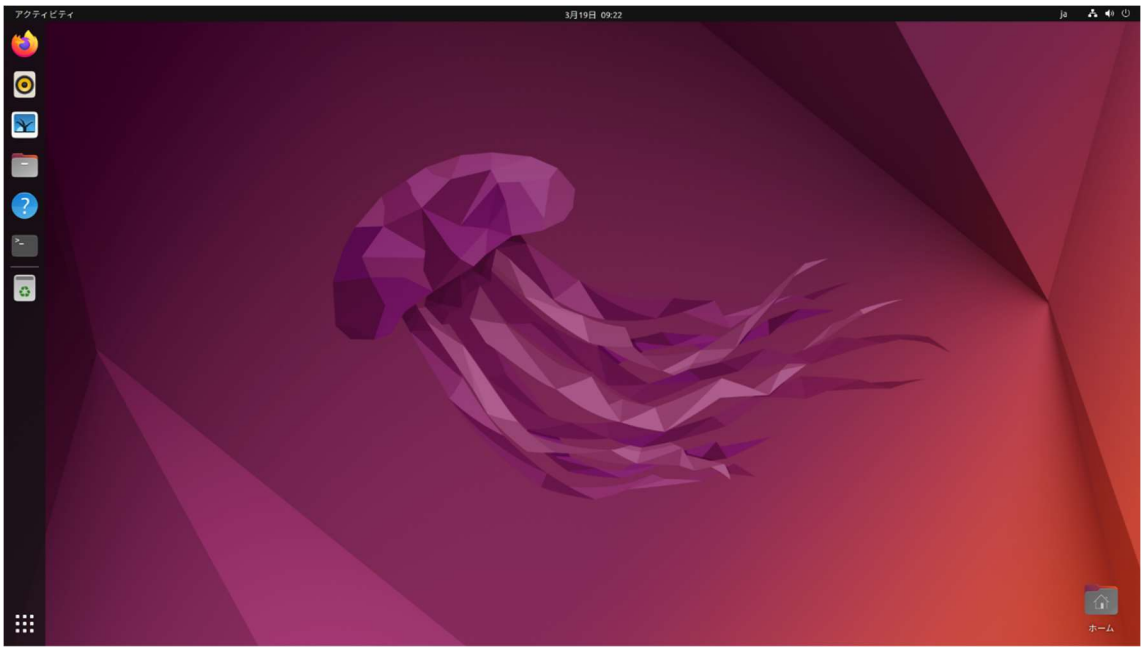


- 入力欄に自身のユーザー名を入力して Enter キーを押す

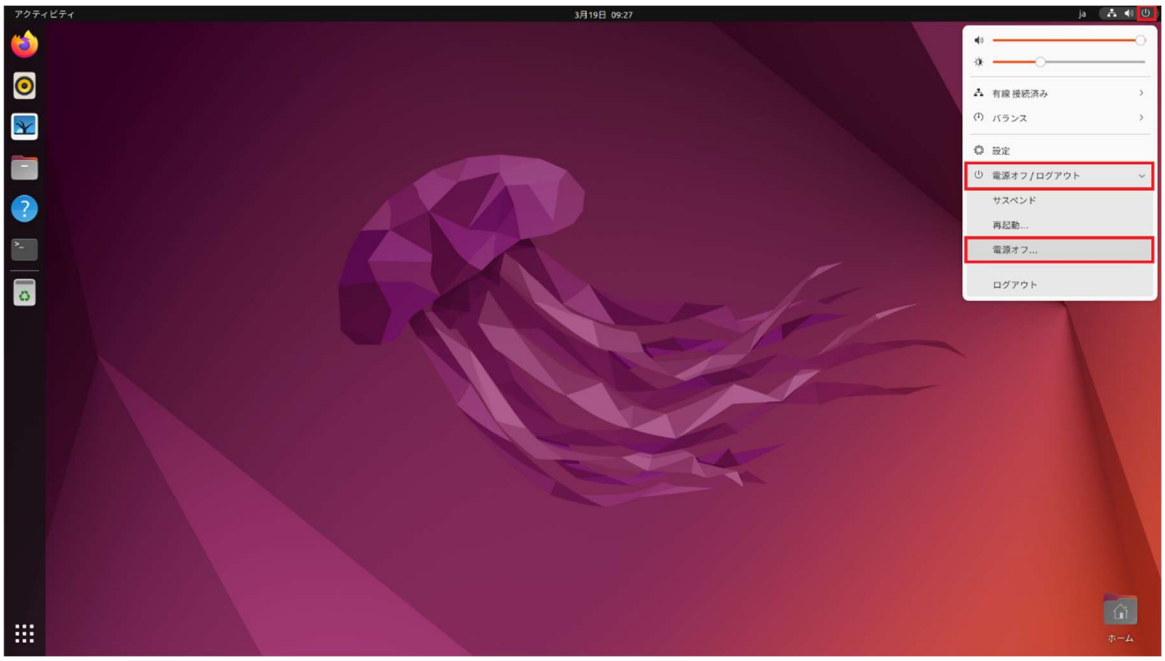
- パスワードを入力して Enter キーを押す

✓ 入力したパスワードは「センターポイント（〇）」で表示されます。

- 正常にログインできると、GNOME デスクトップが表示されます。これで UNIX の操作を行います。



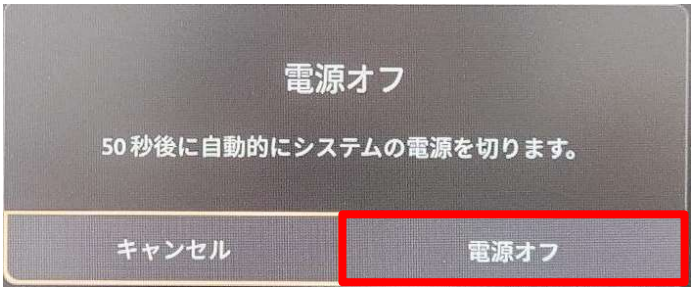
- Ubuntu を終了する場合には、⏻ を選択し 『電源オフ/ログアウト』をクリック、さらに表示されたメニューから『電源オフ』をクリックしてください。



- 『電源オフ 60 秒後に自動的にシステムの電源を切ります。』というダイアログが表示されますので、『電源オフ』ボタンをクリックしてください。なお、秒数の表示はカウントダウンで減っていき、0 になると自動的に電源オフになります。

- ログオンしていない状態から終了する場合は、ログオン画面最上段の右端の『⏻』をクリックしてください。

- 上図のようにメニュー画面が表示されるのでメニュー欄下段の『⏻電源オフ/ログアウト』をクリックし、さらに開いたメニューより、「電源オフ」を選択します。



- 『キャンセル』、『電源オフ』を選ぶメニューが表示されますので、『電源オフ』をクリックします。

ここまでの起動から停止までの流れは、何度も試してみて問題なく実行できるようにしておいてください。これがスムーズにできないと、その先の学習には進めません。

1.3　　プログラムの作成から実行まで(C 言語)

それでは練習のために、実際にプログラムを作成し実行してみましょう。

ここでは C 言語のプログラムを作成します。FORTRAN 言語で実習してみたい場合には 20 ページからの『プログラムの作成から実行まで(FORTRAN 言語)』を参照してください。

1.3.1　プログラムの作成について

C 言語のプログラムを作成するためには、テキストエディタというソフトを使います。

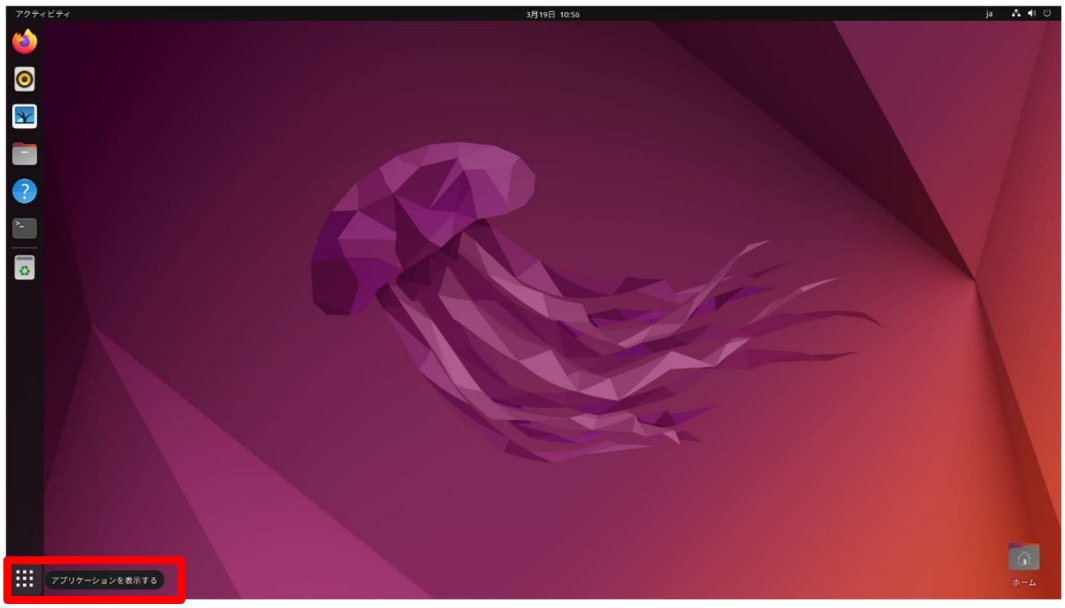
生田システムの UNIX 環境ではいくつかのテキストエディタを利用することができますが、ここでは GNOME デスクトップの標準エディタである『gedit』を使って説明を行います。

それではテキストエディタを使って C 言語のプログラムを作成してみましょう。プログラムのファイル名は『test.c』とします。

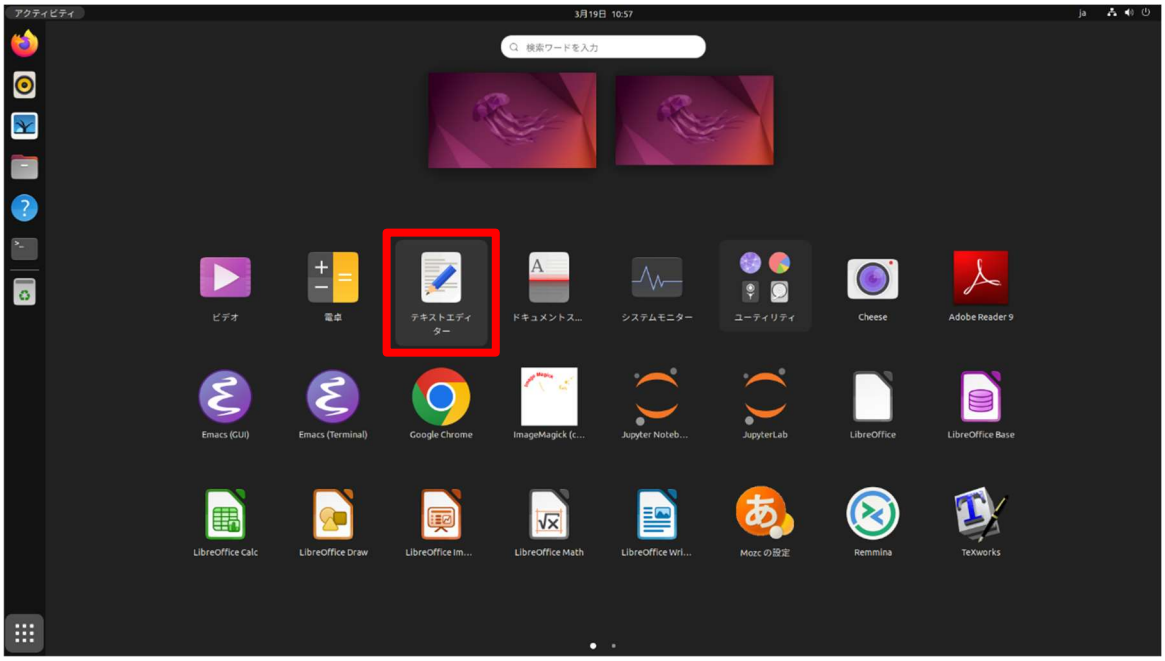
- ✓　ファイル名『test.c』の『.c』の部分を拡張子と言い、ファイルが C 言語のプログラムである事を示しています。
  プログラム自体は任意のファイル名で作成をすることができますが、UNIX システムに C 言語のプログラムである事を認識させるためには、必ずファイル名が『.c』で終わっている必要があります。
  例えば『test.c』、『sample.c』等であれば、C 言語のプログラムとしては正しいファイル名ですが、『test(拡張子『.c』が付いていない)』はもちろん、UNIX では英文字の大文字と小文字が別の文字として区別されるため『test.C(拡張子が大文字になっている)』も誤ったファイル名となります。

1.3.2　テキストエディタで C 言語プログラムを作成する

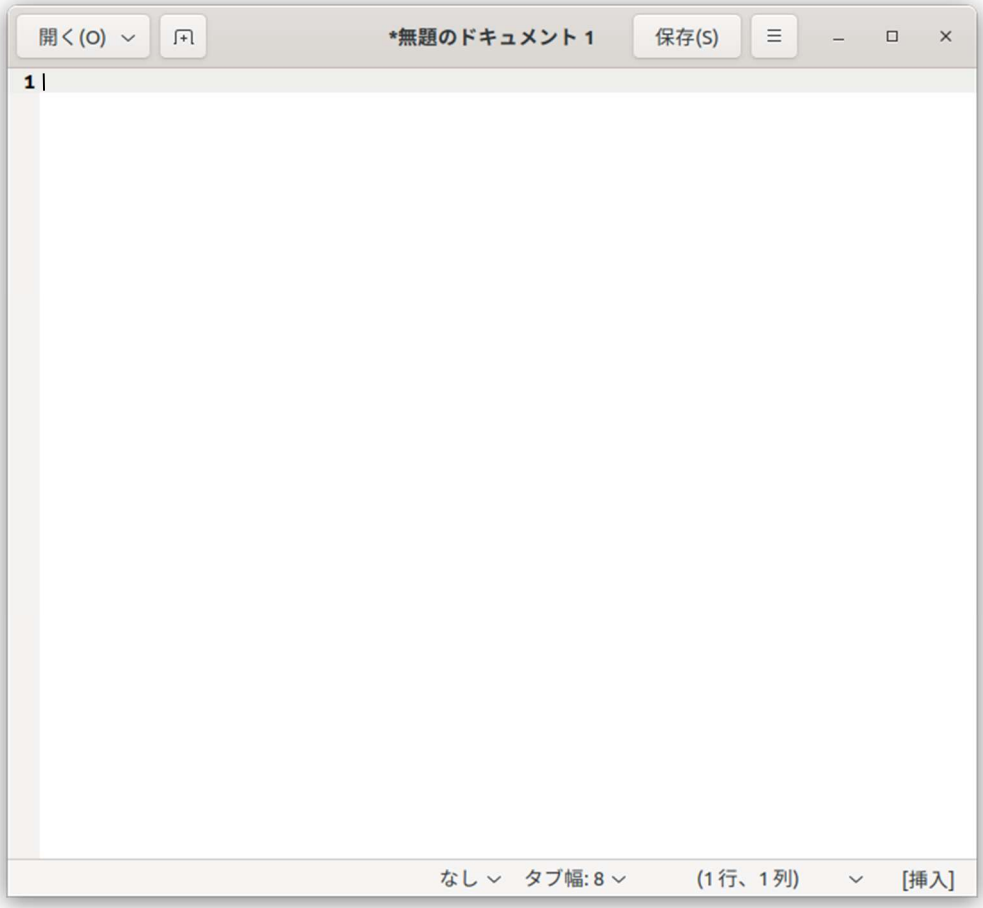
画面左下にある、  （9 点リーダー：マウスを上に移動させると『アプリケーションを表示する』と表示されます）をクリックします。



開いた画面にアプリケーション一覧が表示されます。この中から、『テキストエディター』をクリックします。



テキストエディタが起動し、ウィンドウ内の白い部分の一番左上に『|』が点滅して表示されています。この『|』をカーソルと呼び、この左側に文字が入力されていきます。



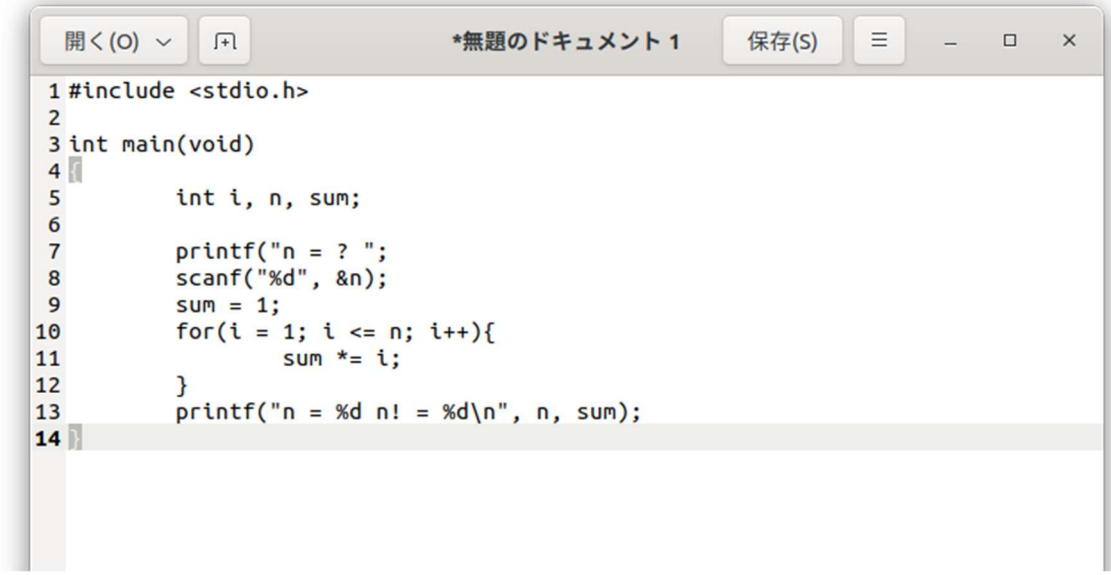
今回作成する C 言語のプログラムは、実行すると数値の入力を待っている状態となり、数値を入力するとその数値の階乗を計算して表示をするというものです。

まず 1 行目に、『#include <stdio.h>』と入力して『Enter』キーを押します。

入力が正しくできれば、カーソルは 2 行目の先頭に移動しているはずです。

もしもタイプミスをしてしまった場合は、矢印キー等を使って、間違えた文字のすぐ右へカーソルを移動し、『BackSpace』キーで間違えた文字を削除し正しい文字を打ち直してください。

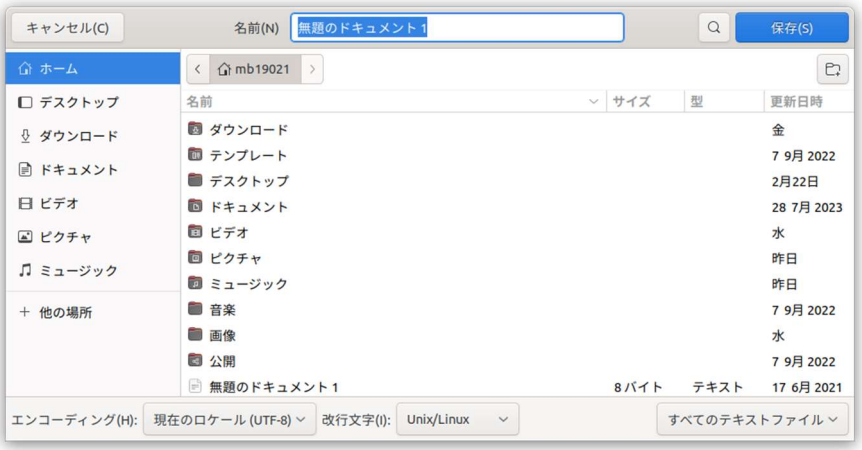
プログラムの残りの部分も続けて入力します。次の図のように文字を入力して完成させてください。



入力が完了したら、タイプミスが無いかよく確認しましょう。タイプミスが無いようならば、作成したプログラムを保存します。

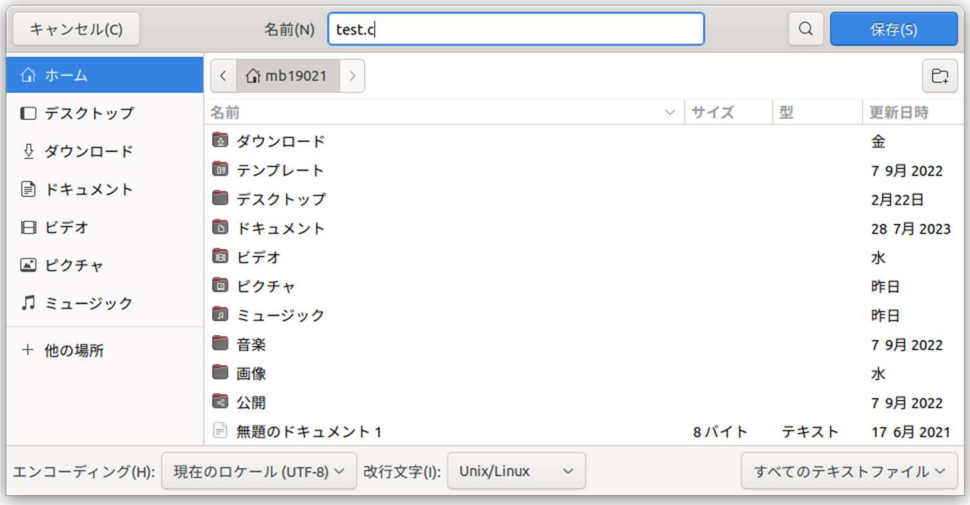
テキストエディタの上段にある『保存』と書かれているボタンの上にマウスカーソルを移動させ（カーソルがボタンの上に移動すると、『このファイルを保存します』というポップアップが表示されます)、クリックします。

クリックするとファイル保存用のダイアログが表示されます。

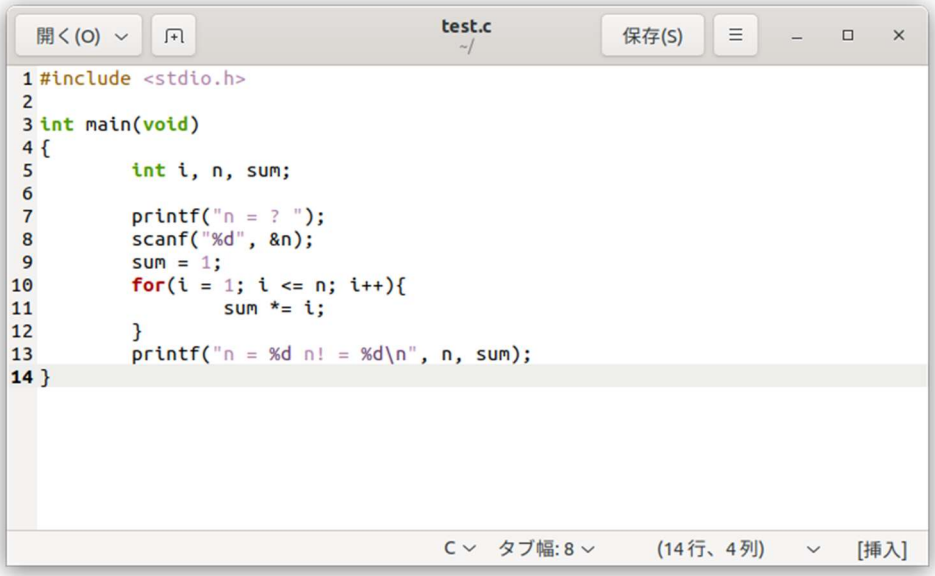


『名前(N)：』の『無題のドキュメント　1』を『test.c』に修正し、『エンコーディング(H)：』を『現在のロケール（UTF-8)』に変更し、任意の保存先を指定したうえで『保存(S)』ボタンをクリックします。今回はホームディレクトリに保存するので、『ホーム』をクリックした後『保存(S)』をクリックしてください。

これで『test.c』の作成が完了しました。



保存が終わるとテキスト編集画面に戻ります。先程の保存画面でファイル名を指定して保存をしたので、タイトルバーにファイル名が表示されています。



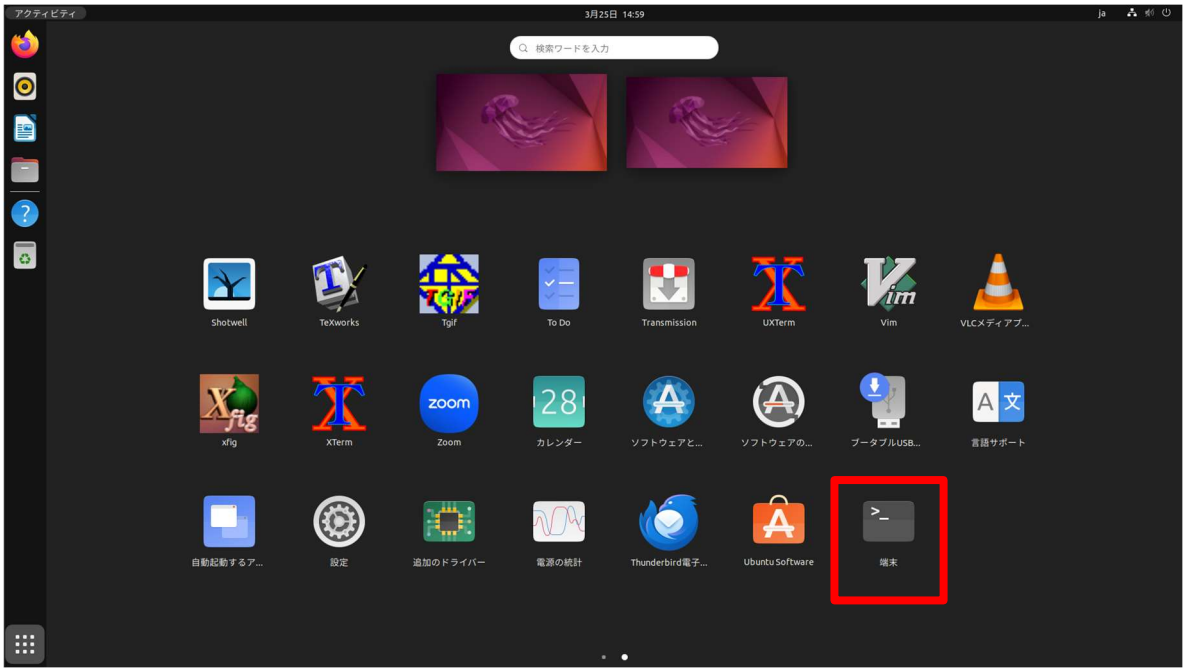
1.3.3　プログラムのコンパイル

こうして作成した C 言語のプログラムですが、実はそのままでは実行することができません。今作成したファイルは、正確にはソースプログラムと呼ばれるもので、人間による判読や作成の作業が容易な形式であり、コンピュータが直接解釈をして実行することができる形式ではないからです。

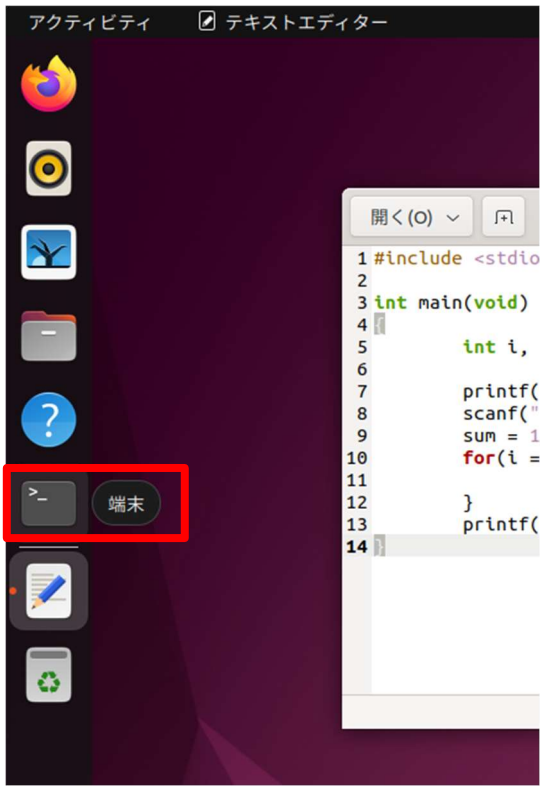
そのため、コンピュータが実行できる形式に変換する必要があります。この変換作業のことを『コンパイル』といい、変換に使うプログラムを『コンパイラ』といいます。C 言語のソースプログラムをコンピュータが実行できるように変換するプログラムは、一般に『C コンパイラ』と呼ばれています。

コンパイル作業はテキストエディタでは行えないので、作業のために別のプログラムを起動します。

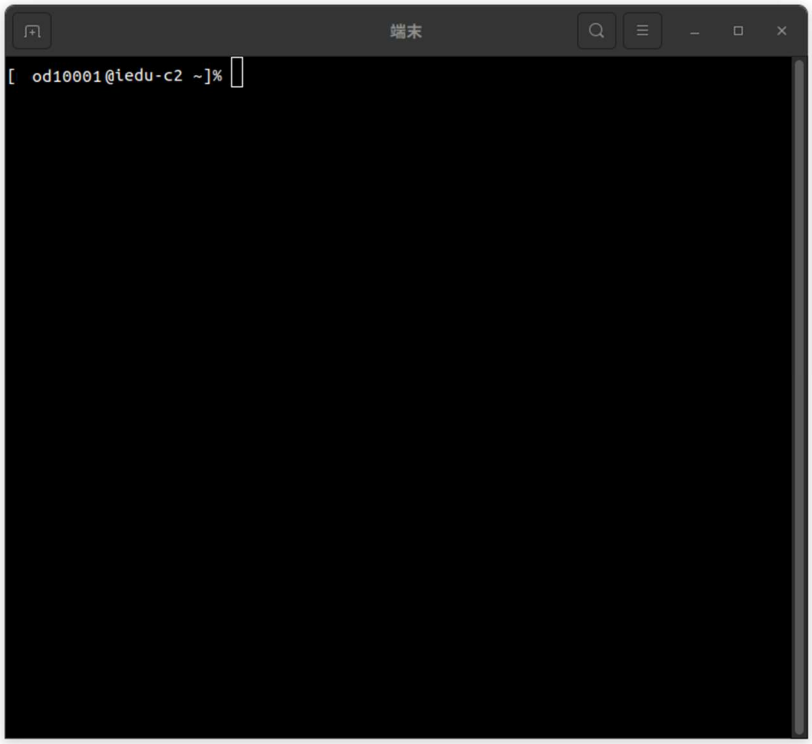
画面左下にある、  をクリックしアプリケーション一覧から『端末』をクリックします。



『端末』がアプリケーション一覧に見当たらない場合は、画面左側の Dock の中にあります。



『端末(ターミナル)』が新しいウィンドウで起動します。



ターミナルは、UNIX の各種コマンドを入力し、その実行結果を表示させるためのプログラムです。C コンパイラを使った C 言語のソースプログラムのコンパイル作業もこれで行います。

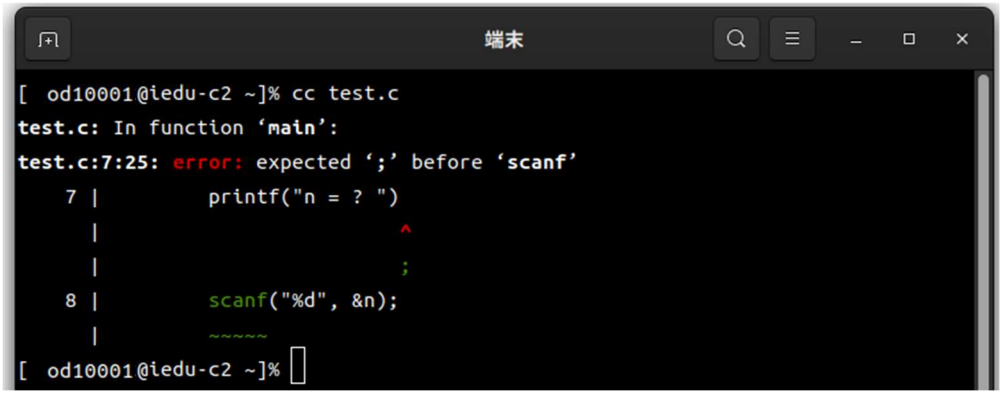
ターミナルの 1 行目に『[od10001@iedu-c2 ~]%』と表示されていますが、これはコマンドプロンプトと呼ばれるもので、ユーザーからコマンドが入力されるのを待っている状態です。

- ✓ 『od10001』は皆さんのユーザー名、『iedu-c2』は皆さんがその時に利用している(ログインしている)コンピュータの名前になります。『~』の部分はカレントディレクトリを表します。(『~』はホームディレクトリを意味するので、カレントディレクトリはホームディレクトリとなります)

それでは実際にコンパイルを行います。コマンドプロンプトに『cc test.c』と入力して『Enter』キーを押してください。

作成したソースプログラムにタイプミスなどの誤りが無ければ、特にメッセージ等が表示されずに再びコマンドプロンプトが表示されます。この場合、コンパイルは成功したことになります。

しかし、タイプミスなどがあると、次の図のように何らかのメッセージが出力されます。



この場合、7 行目辺りに間違いがあるようです。ただし、このエラーメッセージで表示されるエラーの該当箇所については、必ずしも正確ではありません。もっと前の時点でのミスが原因で、7 行目の処理時にエラーになることもあるからです。

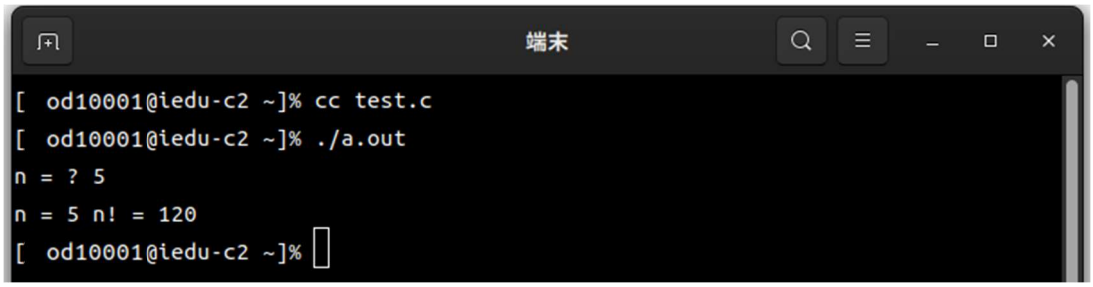
エラーメッセージが出力された場合は、テキストエディタに戻ってソースプログラムの修正を行ってください。修正後は先程と同じように、作成したソースプログラムを保存することを忘れないようにしてください。

1.3.4　プログラムの実行

コンパイルが成功したら、いよいよプログラムを実行してみましょう。

C コンパイラは、ユーザーが特に指定しない場合には『a.out』というファイル名で実行ファイルを作成します。この『a.out』を実行するためには、コマンドプロンプトから『./a.out』と入力して『Enter』キーを押します。

プログラムが階乗を計算する数値の入力を求めてくるので、適当な数値を入力します。ここでは例として『5』を入力し、『Enter』キーを押しました。すると次のような結果になります。



『5』の階乗である『120』が計算して出力されました。

✓　正しい答えが出ない場合はソースプログラムが間違っている可能性がありますので、ソースプログラムを見直してみてください。

ここで指定する数値は、10 位までにしておいてください。大きな数値を指定すると、桁溢れなどが起こり実行エラーとなる場合があります。また、数値以外を入力してもエラーとなってしまいます。『test.c』はあくまでも練習用のサンプルプログラムなので、実用的なプログラムには必要不可欠な様々なエラー処理を省いてあるからです。

✓　入力待ちの時などのプログラムの実行中に、強制的にプログラムを終了したい場合には、『Ctrl＋C』を入力してください。実行されているプログラムが強制的に終了され、コマンドプロンプトが再び表示されます。

プログラムのコンパイルをする際に、『**cc -o 出力ファイル名 test.c**』のようにすると、実行ファイル名を指定することができます。

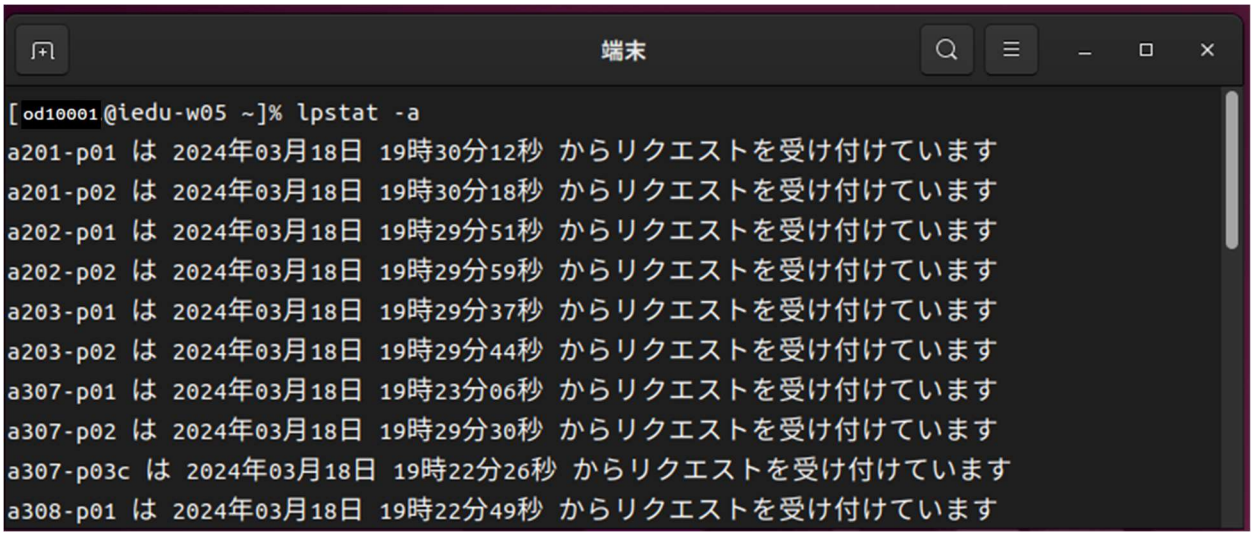
1.3.5　プリンタへの出力

作成したC言語のソースプログラム『test.c』の内容を、プリンタから印刷してみましょう。

印刷の実行には『lpr』というコマンドを使います。(コマンドの詳細については99ページを参照)

出力先のプリンタ名を指定する必要がありますが、印刷できるプリンタの名前は利用している部屋ごとに異なっているので、印刷を実行する前に必ず確認してください。

利用できるプリンタ名を確認するためには『lpstat -a』と入力して『Enter』キーを押してください。

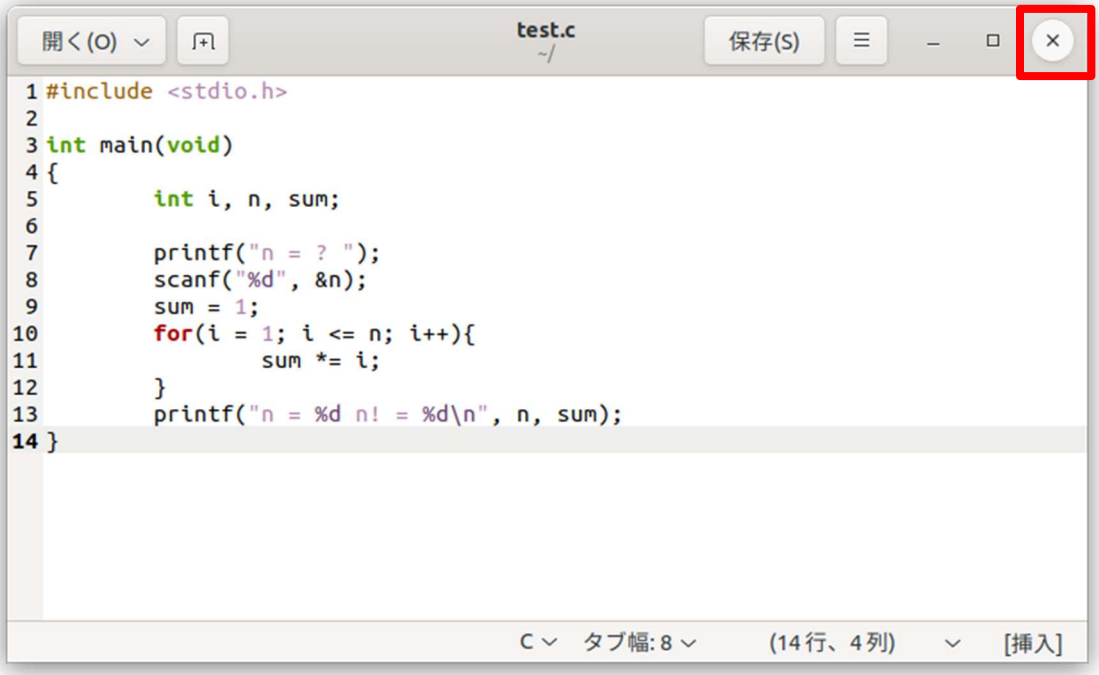


出力されたリストの各行の一番左側に印刷可能なプリンタ名が並んでいます。例えば、ファイル『test.c』をプリンタ『icr2-p01』から印刷するには、コマンドプロンプトに『lpr -Picr2-p01 test.c』と入力し『Enter』キーを押します。

これで、指定したプリンタから『test.c』の内容が印刷されます。

1.3.6　テキストエディタの終了

ここまでの実習が終了したらテキストエディタを終了しましょう。次の図のように、エディタの右上にある『×』をクリックします。



ファイルに修正がない場合はこのままウィンドウが閉じられます。修正があった場合には保存するかどうかの確認ダイアログが表示されますので、必要に応じてボタンをクリックしてください。

UNIX の実習を終了する場合には、ログアウトの手順（『1.2 ログインとログアウト』3 ページ）に従って終了処理を行ってください。

1.4　プログラムの作成から実行まで(FORTRAN 言語)

それでは練習のために実際にプログラムを作成し実行してみましょう。

ここでは FORTRAN 言語のプログラムを作成します。C 言語で実習してみたい場合には 8 ページからの『プログラムの作成から実行まで(C 言語)』を参照してください。

1.4.1　プログラムの作成について

FORTRAN 言語のプログラムを作成するためには、gedit というソフトを使います。

生田システムの UNIX 環境ではいくつかのテキストエディタを利用することができますが、ここでは GNOME デスクトップの標準エディタである『gedit』を使って説明を行います。

それでは gedit を使って FORTRAN 言語のプログラムを作成してみましょう。プログラムのファイル名は『test.f』とします。

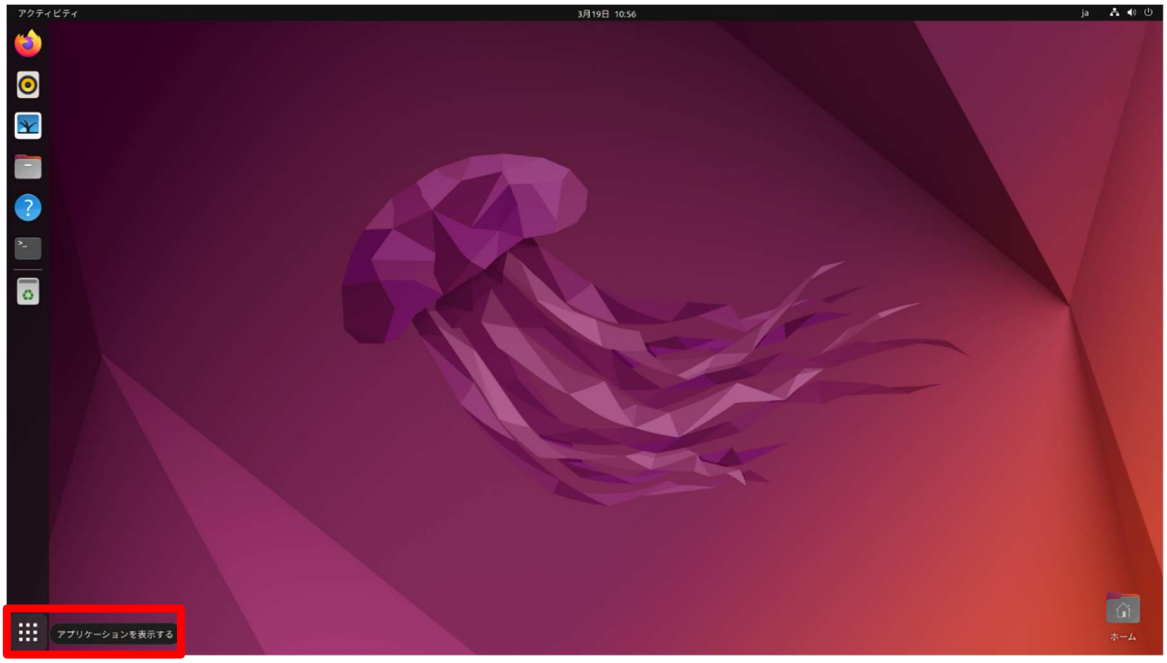
- ✓　ファイル名『test.f』の『.f』の部分を拡張子と言い、ファイルが FORTRAN 言語のプログラムである事を示しています。

  プログラム自体は任意のファイル名で作成をすることができますが、UNIX システムに FORTRAN 言語のプログラムである事を認識させるためには、必ずファイル名が『.f』で終わっている必要があります。

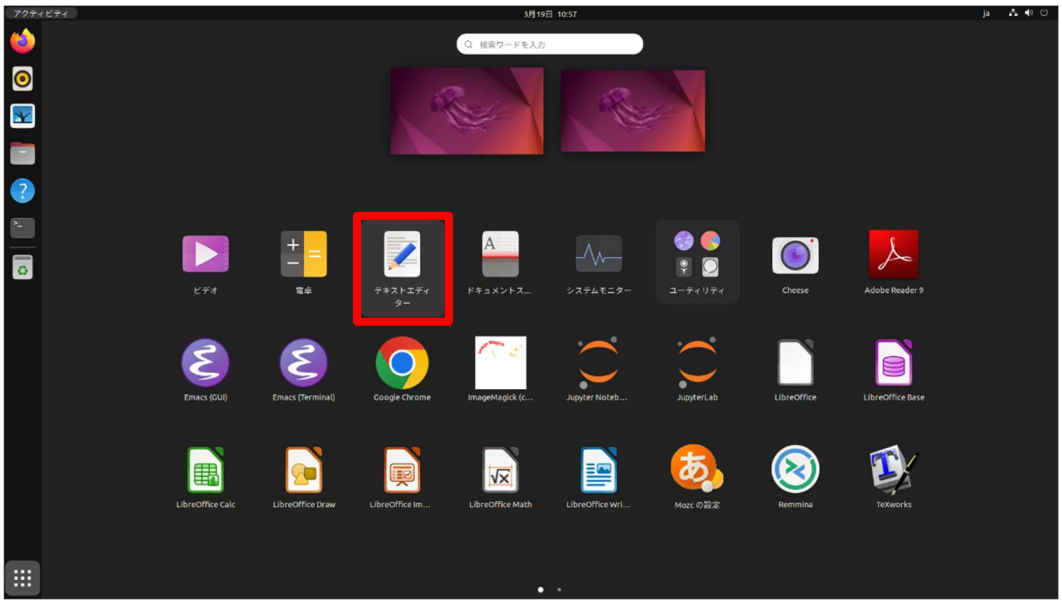
  例えば『test.f』、『sample.f』等であれば、FORTRAN 言語のプログラムとしては正しいファイル名ですが、『test(拡張子『.f』が付いていない)』はもちろん、UNIX では英文字の大文字と小文字が別の文字として区別されるため『test.F(拡張子が大文字になっている)』も誤ったファイル名となります。

1.4.2　テキストエディタで FORTRAN 言語プログラムを作成する

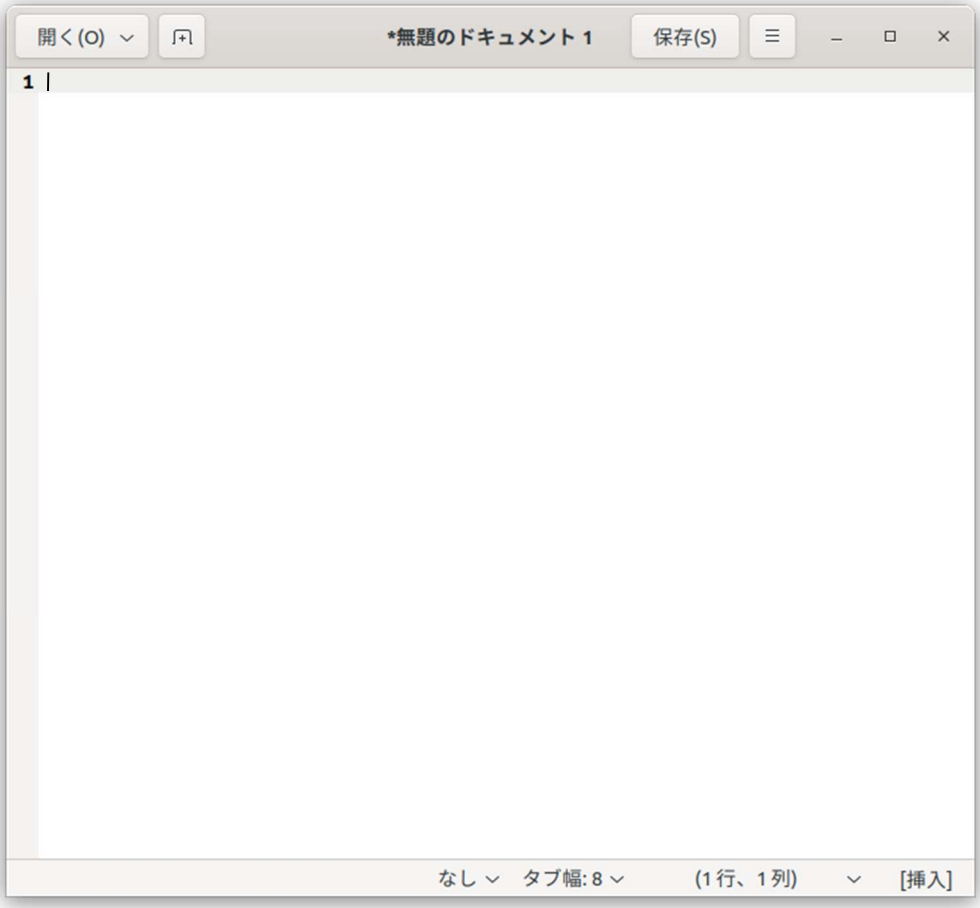
画面左下にある、  （9 点リーダー：マウスを上に移動させると『アプリケーションを表示する』と表示されます）をクリックします。



開いた画面にアプリケーション一覧が表示されます。この中から、『テキストエディター』をクリックします。



テキストエディタが起動し、ウィンドウ内の白い部分の一番左上に『|』が点滅して表示されています。この『|』をカーソルと呼び、この左側に文字が入力されていきます。

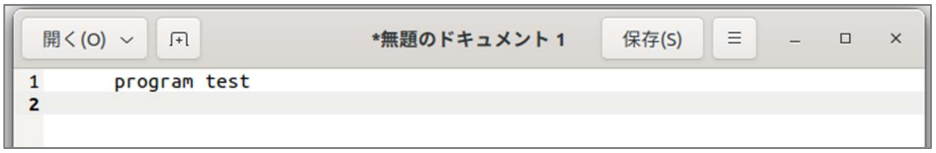


今回作成する FORTRAN 言語のプログラムは、実行すると数値の入力を待っている状態となり、数値を入力するとその数値の階乗を計算して表示をするというものです。

まず 1 行目の最初でキーボードの『Space』キーを 6 回押して半角の空白を 6 個入力してください。続けて『program test』と入力をして『Enter』キーを押します。

入力が正しくできれば、カーソルは 2 行目の先頭に移動しているはずです。

もしもタイプミスをしてしまった場合は、矢印キー等で、間違えた文字のすぐ右へカーソルを移動し、『BackSpace』キーで間違えた文字を削除して正しい文字を打ち直してください。

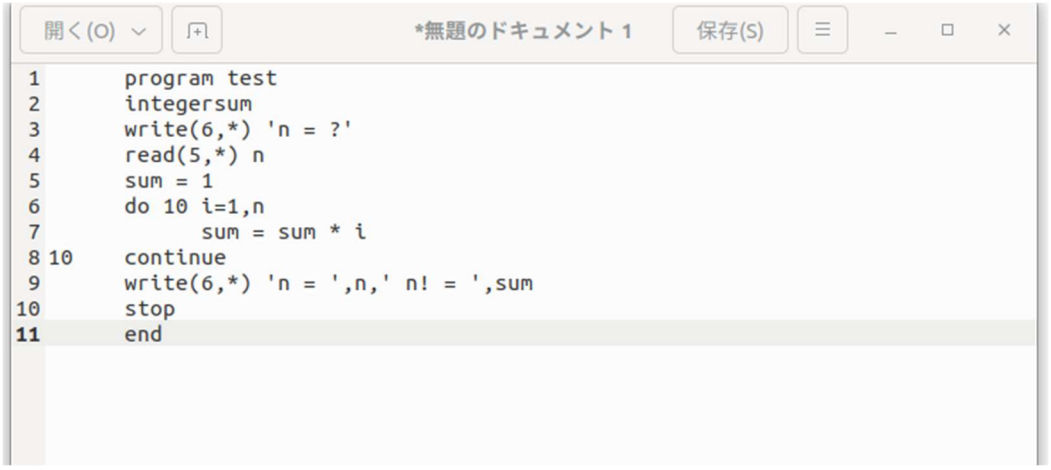


✓ FORTRAN 言語のプログラムは各行の 7 桁目～72 桁目の範囲に入力していきます。73 桁目以降は無視されるので、その部分で行毎に何をやっているかなどのコメント(注釈)を記載しておくことができます。

また 1 桁目に『***(アスタリスク)**』や『C』を入力すると、その行全体がコメント行(注釈行)になります。

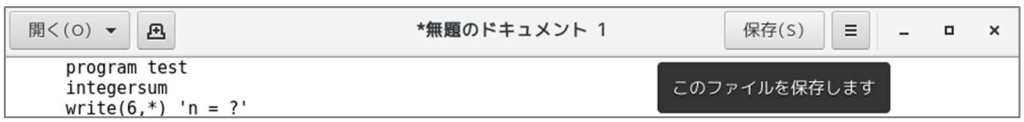
6 桁目に数字や文字を入力すると、直前の行の続き(継続行)と見なされます。

プログラムの残りの部分も続けて入力します。次の図のように文字を入力して完成させてください。



入力が完了したら、タイプミスが無いかよく確認しましょう。タイプミスが無いようならば、作成したプログラムを保存します。

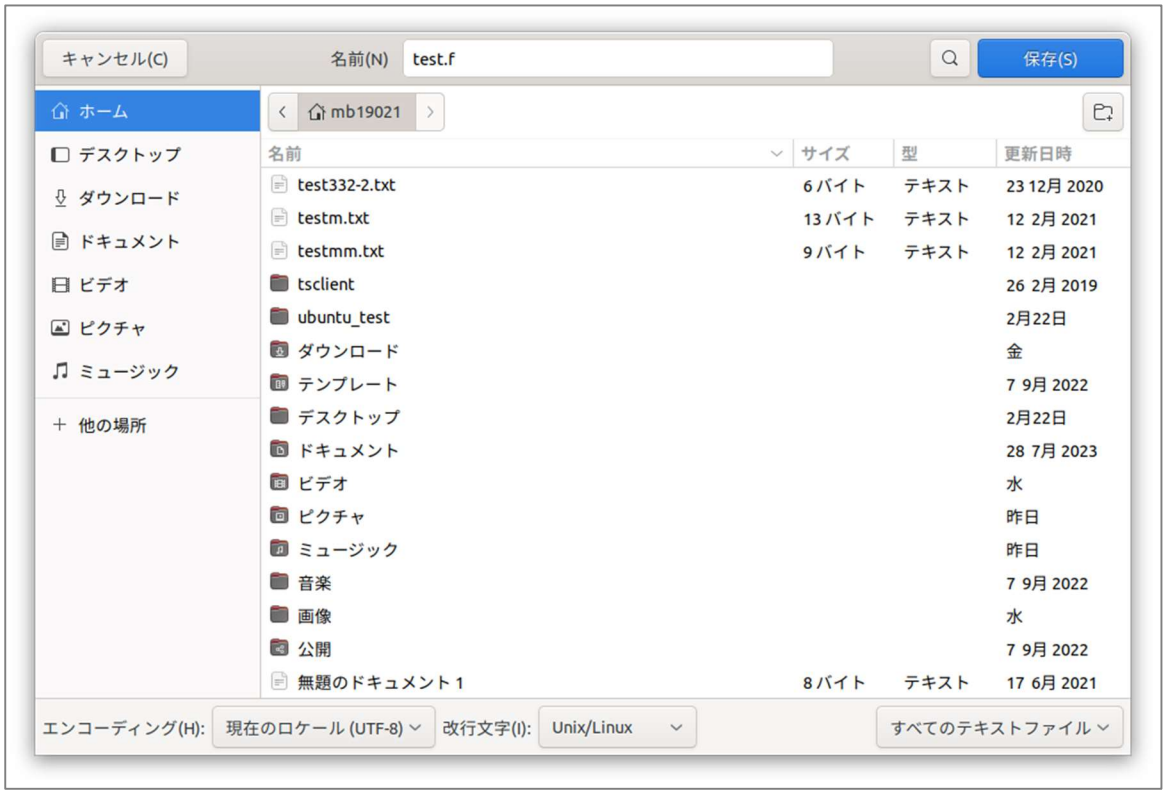
gedit の上段にある『保存』と書かれているボタンの上にマウスカーソルを移動させ（カーソルがボタンの上に移動すると、「このファイルを保存します」と表示されます)、クリックします。



クリックするとファイル保存用のダイアログが表示されます。

『名前(N)』の『無題のドキュメント　1』を『test.f』に修正し、『文字エンコーディング(H):』を『現在のロケール(UTF-8)』に変更し、任意の保存先を指定したうえで『保存(S)』ボタンをクリックします。今回はホームディレクトリに保存するので、『ホーム』をクリックした後『保存(S)』をクリックしてください。

これで『test.f』の作成が完了しました。



保存が終わるとテキスト編集画面に戻ります。先程の保存画面でファイル名を指定して保存をしたので、タイトルバーとタブにファイル名が表示されています。

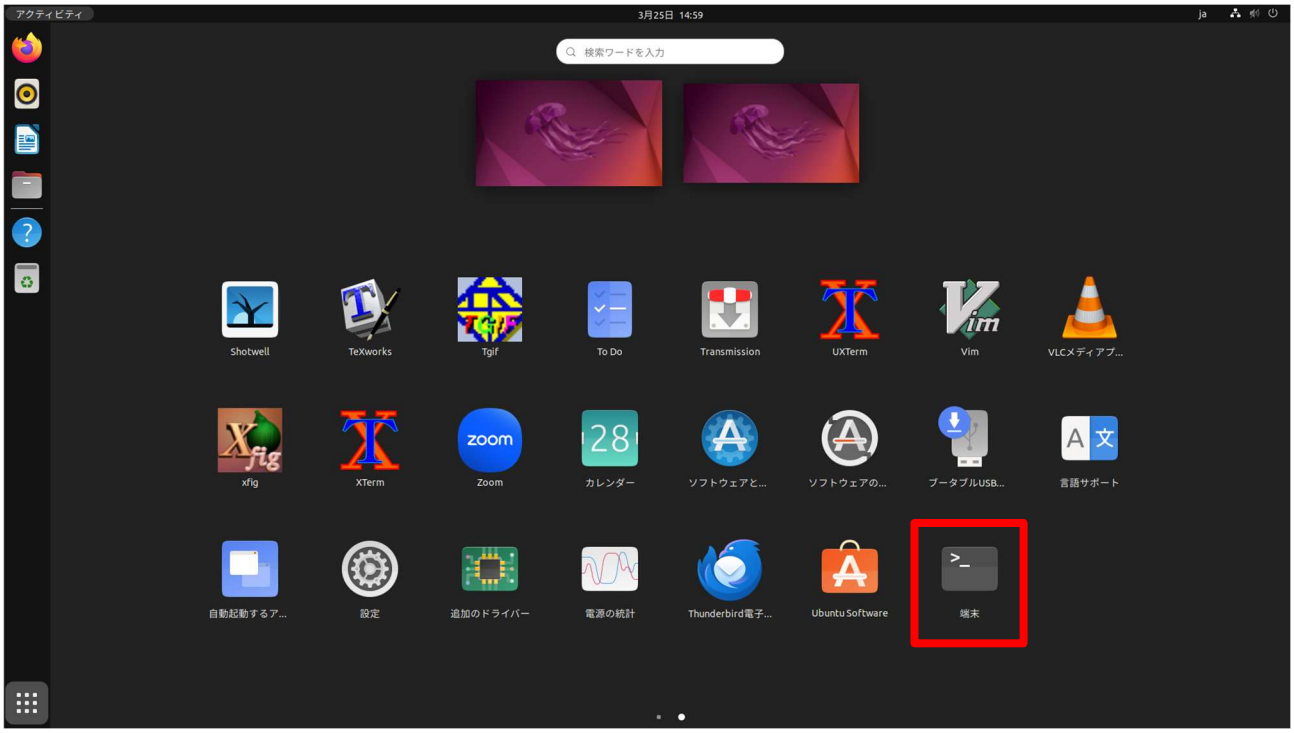
1.4.3　プログラムのコンパイル

こうして作成した FORTRAN 言語のプログラムですが、実はそのままでは実行をすることができません。今作成したファイルは、正確にはソースプログラムと呼ばれるもので、人間による判読や作成の作業が容易な形式であり、コンピュータが直接解釈をして実行することができる形式ではないからです。

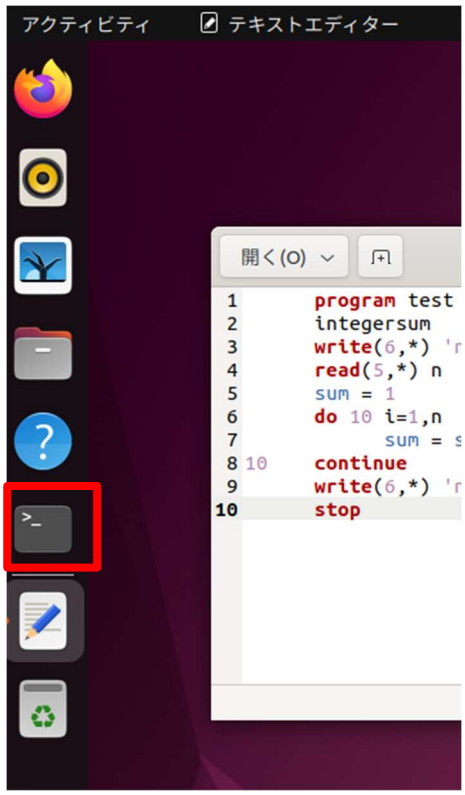
そのため、コンピュータが実行できる形式に変換する必要があります。この変換作業のことを『コンパイル』といい、変換に使うプログラムを『コンパイラ』といいます。FORTRAN 言語のソースプログラムをコンピュータが実行できるように変換するプログラムは、一般に『FORTRAN コンパイラ』と呼ばれています。

コンパイル作業は gedit では行えないので、作業のために別のプログラムを起動します。

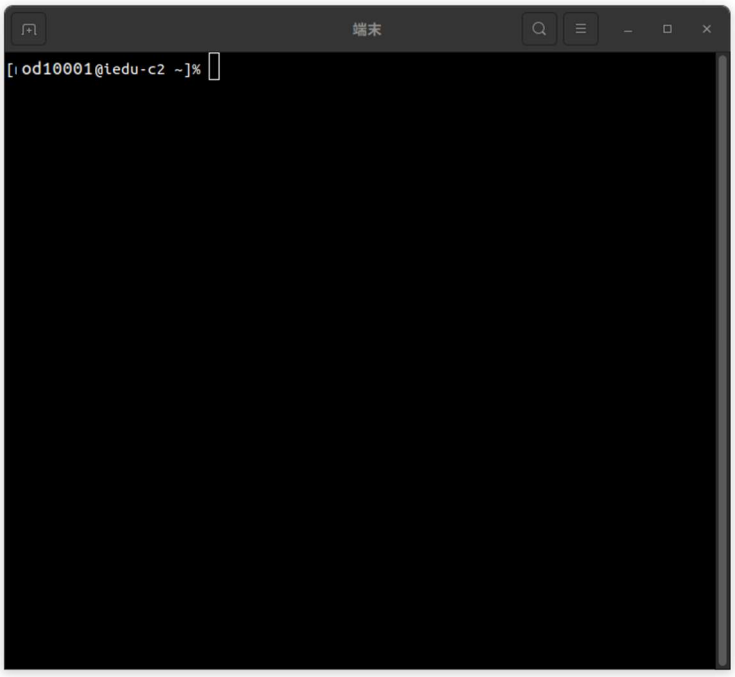
画面左下にある、  をクリックしアプリケーション一覧から『端末』をクリックします。



『端末』がアプリケーション一覧に見当たらない場合は、画面左側の Dock の中にあります。



『端末(ターミナル)』が新しいウィンドウで起動します。



ターミナルは、UNIX の各種コマンドを入力し、その実行結果を表示させるためのプログラムです。FORTRAN コンパイラを使った FORTARAN 言語のソースプログラムのコンパイル作業もこれで行います。

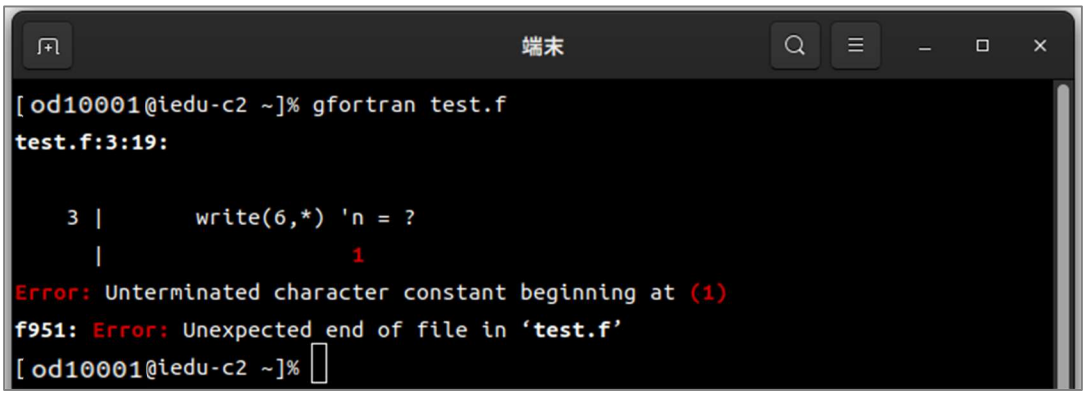
ターミナルの１行目に『[od10001@iedu-c2 ~]%』と表示されていますが、これはコマンドプロンプトと呼ばれるもので、ユーザーからコマンドが入力されるのを待っている状態です。

- ✓ 『od10001』は皆さんのユーザー名、『iedu-c2』は皆さんがその時に利用している(ログインしている)コンピュータの名前になります。『~』の部分はカレントディレクトリを表します。(『~』はホームディレクトリを意味するので、カレントディレクトリはホームディレクトリとなります)

それでは実際にコンパイルを行います。コマンドプロンプトに『gfortran test.f』と入力して『Enter』キーを押してください。

作成したソースプログラムにタイプミスなどの誤りが無ければ、特にメッセージ等が表示されずに再びコマンドプロンプトが表示されます。この場合、コンパイルは成功したことになります。

しかし、タイプミスなどがあると、次の図のように何らかのメッセージが出力されます。



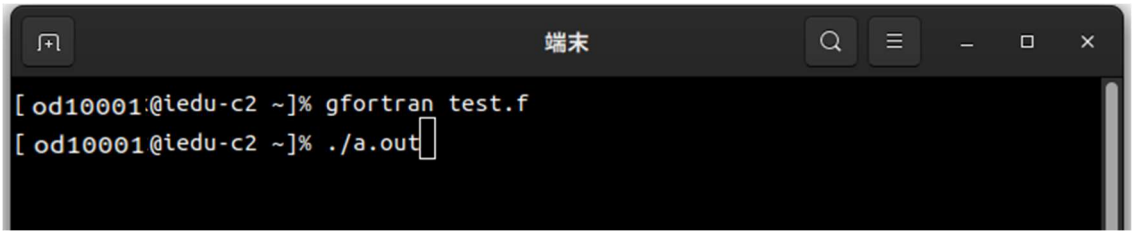
この場合、どうやら 3 行目辺りに間違いがあるようです。ただし、このエラーメッセージで表示されるエラーの該当箇所については、必ずしも正確ではありません。もっと前の時点でのミスが原因で、3 行目の処理時にエラーになることもあるからです。

エラーメッセージが出た場合は、テキストエディタに戻ってソースプログラムの修正を行ってください。修正後は先程と同じように、作成したソースプログラムを保存することを忘れないようにしてください。

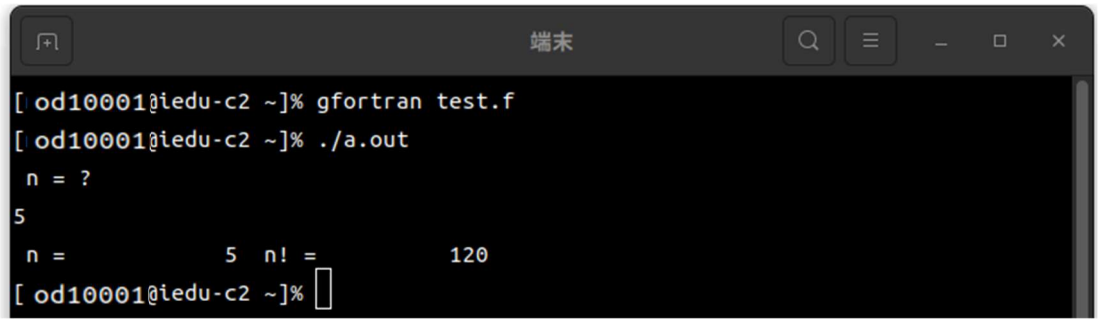
1.4.4　プログラムの実行

コンパイルが成功したら、いよいよプログラムを実行してみましょう。

FORTRAN コンパイラは、ユーザーが特に指定しない場合には『a.out』というファイル名で実行ファイルを作成します。この『a.out』を実行するためには、コマンドプロンプトから『./a.out』と入力して『Enter』キーを押します。



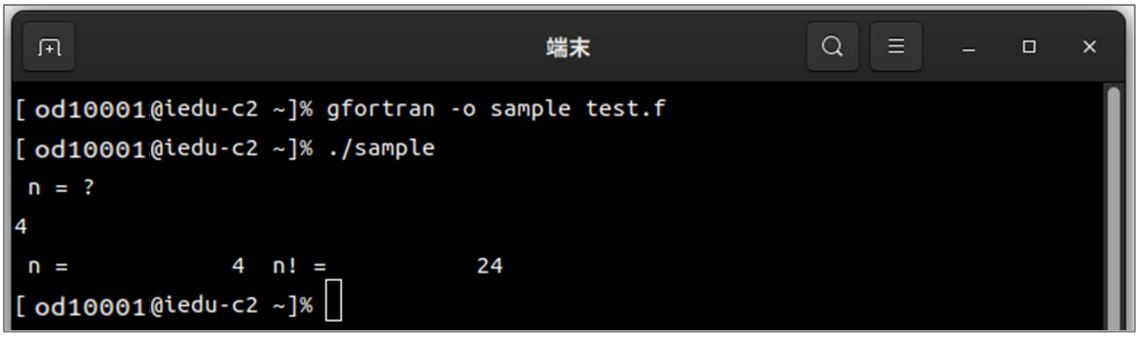
プログラムが階乗を計算する数値の入力を求めてくるので、適当な数値を入力します。ここでは例として『5』を入力し、『Enter』キーを押しました。すると次のような結果になります。



『5』の階乗である『120』が計算して出力されました。

- ✓　正しい答えが出ない場合はソースプログラムが間違っている可能性がありますので、ソースプログラムを見直してみてください。
  ここで指定する数値は、10 位までにしておいてください。大きな数値を指定すると、桁溢れなどが起こり実行エラーとなる場合があります。また、数値以外を入力してもエラーとなってしまいます。『test.f』はあくまでも練習用のサンプルプログラムなので、実用的なプログラムには必要不可欠な様々なエラー処理を省いてあるからです。
- ✓　入力待ちの時などのプログラムの実行中に、強制的にプログラムを終了したい場合には、『Ctrl＋C』を入力してください。実行されているプログラムが強制的に終了され、コマンドプロンプトが再び表示されます。

プログラムのコンパイルをする際に、『**gfortran -o 出力ファイル名 test.f**』のようにすると、実行ファイル名を指定することができます。



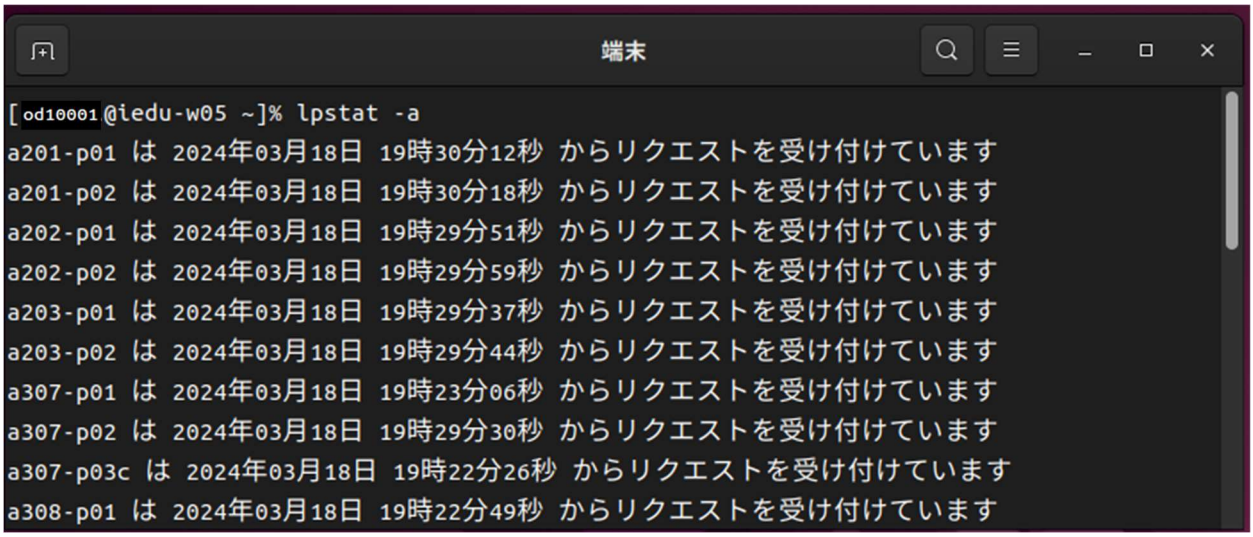
1.4.5　プリンタへの出力

作成した FORTRAN 言語のソースプログラム『test.f』の内容を、プリンタから印刷してみましょう。

印刷の実行には『lpr』というコマンドを使います。(コマンドの詳細については 99 ページを参照)

出力先のプリンタ名を指定する必要がありますが、印刷できるプリンタの名前は利用している部屋ごとに違っているので、印刷を実行する前に必ず確認をしてください。

利用できるプリンタ名を確認するためには『lpstat -a』と入力して『Enter』キーを押してください。



出力されたリストの各行の一番左側に印刷可能なプリンタ名が並んでいます。例えば、ファイル『test.f』をプリンタ『icr2-p01』から印刷するには、コマンドプロンプトに『lpr -Picr2-p01 test.f』と入力し『Enter』キーを押します。

これで、指定したプリンタから『test.f』の内容が印刷されます。

1.4.6　テキストエディタの終了

ここまでの実習が終了したらテキストエディタを終了しましょう。テキストエディタのウィンドウの右上の「×」をクリックして終了します。

ファイルに修正がない場合はウィンドウが閉じられます。修正があった場合には保存するかどうかの確認ダイアログが表示されますので、必要に応じてボタンをクリックしてください。

UNIX の実習を終了する場合には、ログアウトの手順（『1.2 ログインとログアウト』3 ページ）に従って終了処理を行ってください。

1.5　ウィンドウシステム（GNOME デスクトップ）の基本操作

ここでは Ubuntu の標準的なウィンドウシステムである GNOME の操作について説明します。

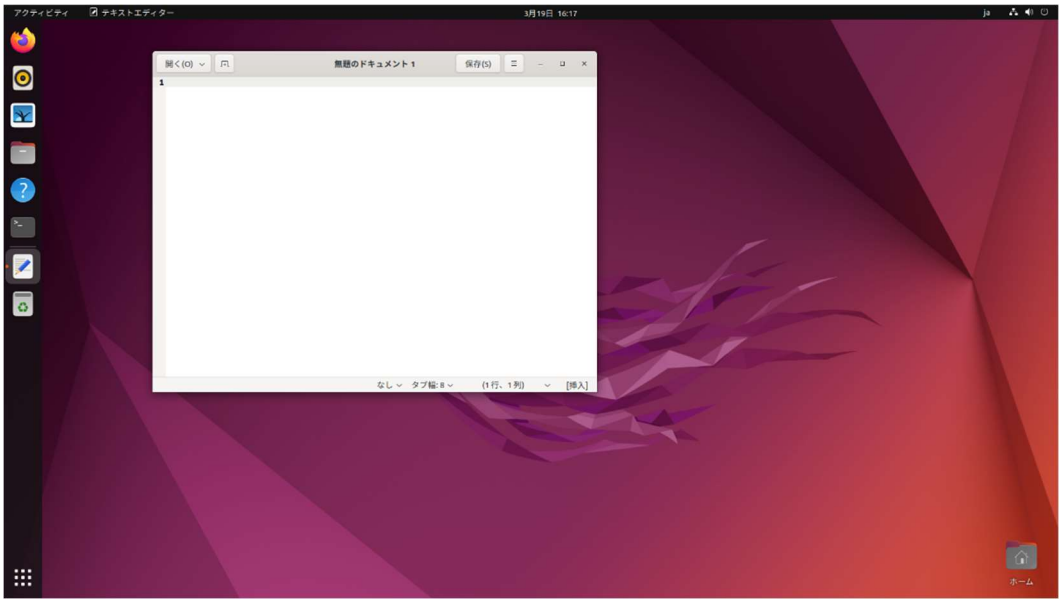
Ubuntu の起動と終了、ログインとログアウトの方法、Windows デスクトップと画面切り替え方法などについては、『第 1 章 UNIX の基本的な使い方』の 3 ページに記載されている『1.2 ログインとログアウト』などを参照してください。

1.5.1　ウィンドウの大きさを変更する

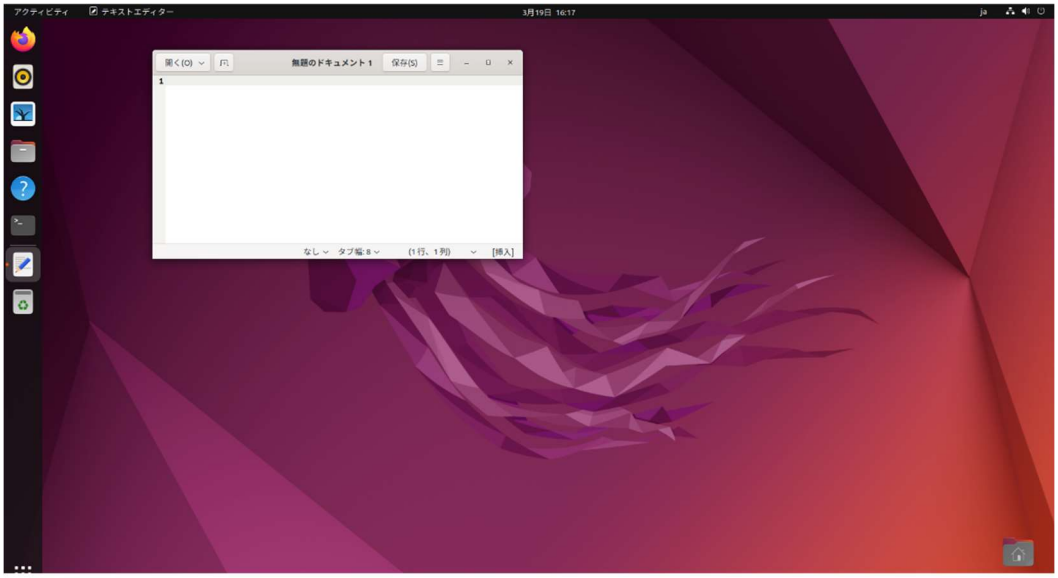
各ウィンドウには、右下にサイズ変更に使うリサイズコーナーがあります。ウィンドウの大きさを変更する場合には、まず、リサイズコーナーにマウスカーソルを移動させ、マウスカーソルの形状が変わったところでマウスをドラッグすると、カーソルの動きに合わせてウィンドウのサイズが変わります。

好みの大きさになったところでマウスのボタンから手を離すと、ウィンドウの大きさが変更されます。

- アプリケーションを立ち上げてそのままのウィンドウの状態



- ウィンドウのサイズを変更したところ



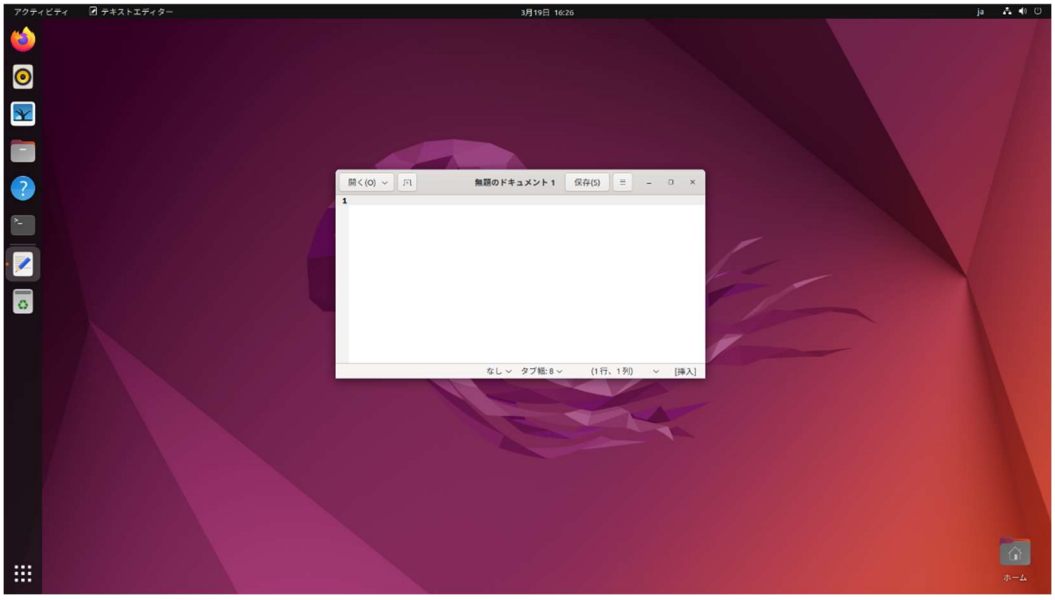
1.5.2　ウィンドウを移動する

ウィンドウは作業が行いやすいように好きな位置に移動することができます。

まず、移動したいウィンドウのタイトルバー(ウィンドウ最上部の黒い部分)にマウスカーソルを持っていきます。そして、マウスでウィンドウを移動したい位置までドラッグします。

好みの場所まで移動してマウスのボタンを離すと、その場にウィンドウが移動します。

- ウィンドウを移動したところ。

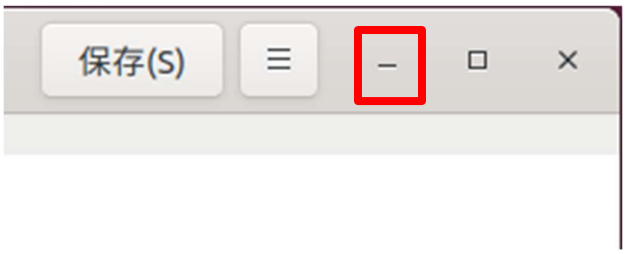


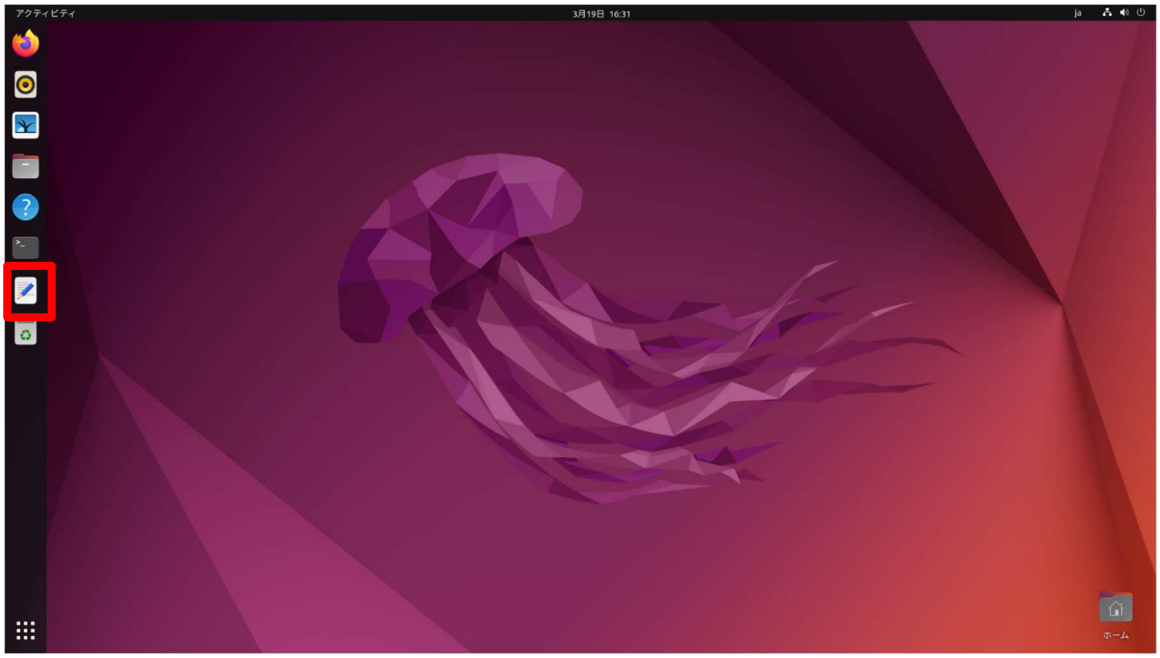
1.5.3　ウィンドウを最小化する

ウィンドウを開きすぎて邪魔になった場合には、最小化(アイコン化)をして片づけておくことができます。この機能は Windows とほぼ同じ操作です。

最小化(アイコン化)したいウィンドウのタイトルバーの右端にある３つのボタン『(＿□×)』のうち、『＿』をクリックするとウィンドウが画面左側の Dock に格納されます。

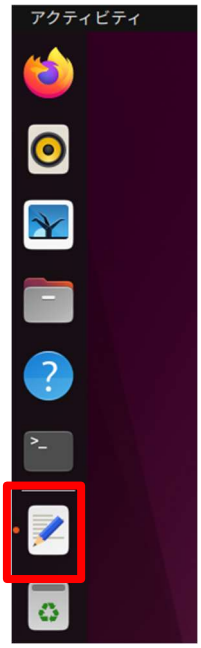
✓　この他、『□』は最大化ボタン、『×』は閉じるボタンです。必要に応じて利用してみてください。



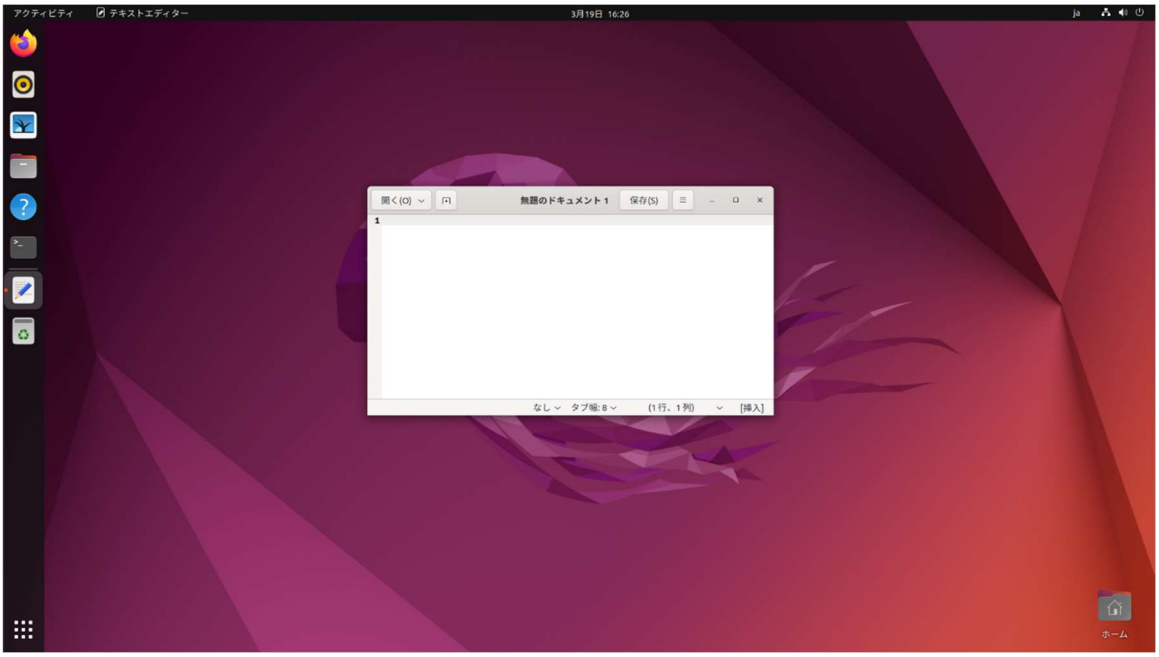


1.5.4　最小化したウィンドウを元に戻す

最小化しておいたウィンドウを元に戻すには、画面左部の Dock を使います。



Dock 内の目的のウィンドウのアイコンを左クリックすることで、最小化前の元の位置にウィンドウが表示されます。



第２章　　UNIX の基礎知識

2.1 UNIX とは

この章では、UNIX を利用する上で必要な基礎知識について解説します。

UNIX とは、ワークステーションと呼ばれるコンピュータで広く採用されている OS(Operating System)の総称です。近年パソコンの世界でも話題になっている Linux も、UNIX の仲間のひとつです。

生田システムのパソコン環境では、Linux のひとつである Ubuntu を UNIX 環境として採用しています。

また、パソコン環境とは別に、生田システムには PC クラスタおよび IA 計算サーバと呼ばれる環境があり、これらも UNIX を採用しています。

PC クラスタ、IA 計算サーバ共に Red Hat Enterprise Linux を採用しています。

※参考

環境	OS
学生用パーソナルコンピュータ	Windows11 Ubuntu 22.04 LTS
PC クラスタ(pcc-mgr)	Red Hat Enterprise Linux 7.5
IA 計算サーバ(isc-iasrv)	Red Hat Enterprise Linux 8.6

2.2　UNIX のコマンドについて

2.2.1　コマンドの入力

Windows や macOS などのパソコンの OS は、様々な操作をマウスで行うように設計されていますが、UNIX においてはキーボードから各種コマンドを入力して実行するという操作が基本になっています。そのため、パソコンの OS に比べて親しみづらいのは確かです。実際、Solaris や各種 Linux のパッケージも、多くの人に受け入れられることを目指して、マウス操作で各種操作が行えるように機能を追加してきています。

しかし、パソコンの OS と同じような使い方しかしないのであれば、UNIX を使う意味はあまりないでしょう。特に生田システムでは、Windows 上でも自分のデータ領域に保存したファイルの操作などを行えますし、各種アプリケーションソフトや言語ソフトも用意されています。せっかく UNIX を使うのならば、UNIX ならではの特徴と利点があるので、それをうまく活用できるように頑張ってください。

UNIX の操作を行うためのコマンドは、基本的に次のような形式になっています。

% コマンド名　［オプション］［引数］

最初の『%』の部分はプロンプトあるいはコマンドプロンプトといい、UNIX がユーザーからのコマンド入力を待っている状態を示しています。生田システムの UNIX 環境では、コマンドプロンプトは『[UserName@ServerName ~]%』のように、現在使用しているユーザー名、コンピュータの名前(サーバー、ホスト名)、カレントディレクトリが一緒に表示されるようになっています。

コマンド名の部分には実行したいコマンドの名前を入力します。コマンド名は省略することはできません。生田システムの UNIX 環境では、世の中の UNIX 環境のどこでも使える標準的なコマンドの他に、システム管理者がインストールを行った多数のフリーソフト、教育研究用に購入したソフトなど、たくさんのコマンドが利用できます。利用したいコマンドについては、その使い方をひとつひとつ覚えていく必要があります。

オプションは、コマンドに対して、細かい動作指示を与えるためのものです。利用できるオプションや指定方法は、コマンドによって違います。

引数は、コマンドに操作対象(ファイル名、ユーザー名など)を指定するためのものです。

- ✓　オプションと引数は[ ]で囲まれていますが、これは省略が可能であることを意味しています。ただし、コマンドによっては引数が必須のものもあります。

2.2.2 オンラインマニュアルについて

UNIX にはたくさんのコマンドが用意されていて、オプションの指定方法や使い方も様々です。その全てを憶えておくのはなかなか大変なので、常時参照できるオンラインマニュアルが用意されています。使い方を忘れてしまったときや、細かいオプションの指定方法などを確認したい場合には、このオンラインマニュアルを利用してください。

オンラインマニュアルの使い方については、101 ページの『man』コマンドの解説を参照してください。

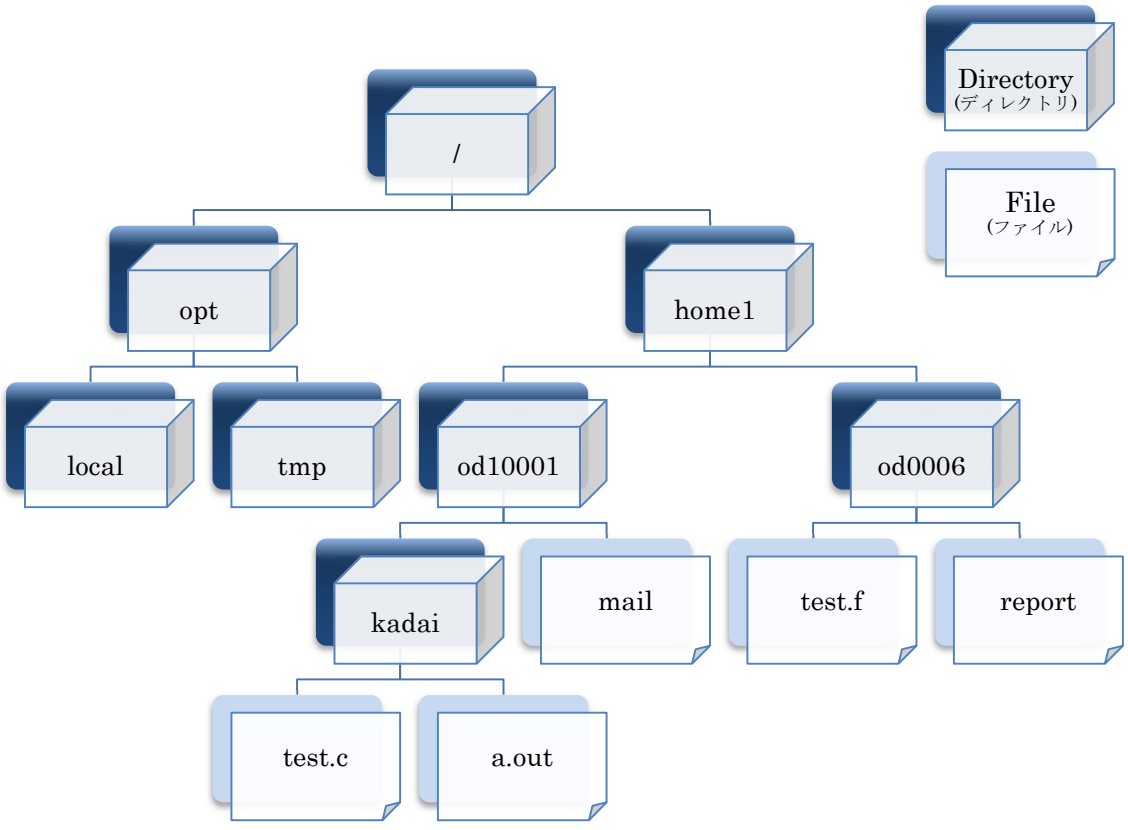
2.3　UNIX のファイルシステム

2.3.1　ファイルとディレクトリ

UNIX は、プログラムや文章、画像などを全てファイルという単位で管理しています。これに加えて、多数のファイルを効率良く管理するために、ディレクトリという仕組みを用意しています。

✓　ファイルとディレクトリという概念は、Windows や Mac などのパソコンの OS でのファイルとフォルダという概念と同じようなものです。

UNIX のディレクトリとファイルは、次の図のような構造になっています。



木を逆さまにしたような構造をしているので、ツリー構造とも呼ばれます。ツリー構造の根本(つまり最上部)のことを『ルートディレクトリ』と呼び、どのような UNIX システムでも、このルートディレクトリはひとつしか存在しません。そしてそのルートディレクトリの下に多数のディレクトリやファイルが階層状に存在しています。ディレクトリの中にはファイルだけではなく別のディレクトリを含むこともでき、そのディレクトリ

の中にも更にファイルやディレクトリを含むことができます。

2.3.2 ホームディレクトリ

UNIX 環境にログインした直後は、ある決まったディレクトリを参照している状態になります。この特別なディレクトリは『ホームディレクトリ(Home Directory)』と呼ばれ、ユーザー一人ひとりに専用のディスク領域が割り当てられています。自分のホームディレクトリの中であれば、ユーザーはファイルやディレクトリを自由に作成することができます。生田システムの環境では、ホームディレクトリの名前はそれぞれのユーザーのユーザーID(ログイン名)と同じになっています。

- ✓ ただし、ひとりのユーザーが使えるディスク領域の上限は決められているので、それを超えてしまうとそれ以上ファイルやディレクトリを作成することはできません。そうなってしまったら不要なファイルを削除してください。

2.3.3 絶対パスと相対パス

コマンドを実行したり各種の操作を行う場合に、目的のファイルやディレクトリを指定しなければならないことがよくあります。UNIX では、その指定の方法が２通りあります。

ひとつは絶対パスによる指定、もうひとつは相対パスによる指定です。

絶対パスとは、UNIX 環境に必ず存在し、しかもひとつしか無いルートディレクトリである『/』から辿ったときの経路を記述する方法です。

例として、先ほどの図で一番最下層にある『a.out』というファイルを絶対パスで指定すると次のようになります。

```
/home1/od10001/kadai/a.out
```

ディレクトリ名やファイル名は『/(スラッシュ)』で区切ります。

一方、相対パスとは、現在参照しているディレクトリを起点にして辿っていった時の経路を記述する方法です。

例として、先ほどの図で、上から３段目にある『od10001』というディレクトリで作業中であり先ほどと同じ『a.out』というファイルを相対パスで指定すると次のようになります。

```
./kadai/a.out
```

最初の『.(ドット)』は現在参照しているディレクトリ(カレントディレクトリ)を表す特殊な記号です。

もしも、カレントディレクトリのひとつ上の階層のディレクトリを参照しようと思った場合には『..(ドットドット)』を指定することでひとつ上の階層のディレクトリを表すことができます。

- ✓ カレントディレクトリを表す『.』を省略して『kadai/a.out』のような表記にすることも可能です。ただし、コマンドを実行するときのコマンド名の指定など、いくつかのケースでは、カレントディレクトリを表す『.』を省略してしまうと、別の場所のコマンドが実行されてしまったりするなどの思わぬ事態に遭遇することがあります。ある程度 UNIX の仕組みについて理解ができるまでは、面倒でも『./kadai/a.out』のようにカレントディレクトリをきちんと書くことをおすすめします。

コマンドを打ち込む際に、絶対パスと相対パスのどちらを使うべきか迷うかもしれませんが、通常のファイル操作では、その場面場面で使いやすい方を使ってください。

例として、カレントディレクトリが先ほどの図の『kadai』であった場合を考えてみてください

この時『a.out』の位置は次のようになります。

絶対パス：/home1/od10001/kadai/a.out
相対パス：./a.out

明らかに相対パスを使った方が簡単そうですね。

今度は図の左端にあるディレクトリ『opt』を指定する場合を考えてみましょう。

絶対パス：/opt
相対パス：../../../opt

絶対パスがすっきりと短く表せているのに対して、相対パスではずいぶんと長い記述になってしまっています。これはディレクトリ階層を何度も上にあがっていかなければならないためで、『..』を繰り返し記述することになるからです。

その場その場でどちらの指定方法を使った方が効率良いかは、UNIX を使い込んでいくに従って自然と身についてきますので、まずは色々と試してみることから始めてみてください。

2.3.4 ファイル操作命令

ファイルに対して行える基本的な操作には、次のようなものがあります。

- ファイルの内容を表示する

- ファイルを連結する
- ファイルをコピーする
- ファイル名を変更する
- ファイルを別の場所に移動する
- ファイルを削除する

具体的な操作については、コマンド解説のページを参照してください。ファイル操作命令については、それぞれ次のページに記載されています。

- 『cat』コマンド　ファイルの内容を表示する　74 ページ
- 『cat』コマンド　ファイルを連結する　74 ページ
- 『cp』コマンド　ファイルをコピーする　79 ページ
- 『mv』コマンド　ファイル名を変更する　106 ページ
- 『mv』コマンド　ファイルを別の場所に移動する　106 ページ
- 『rm』コマンド　ファイルを削除する　111 ページ

2.3.5 ディレクトリ操作命令

ディレクトリに対して行える基本的な操作としては、次のようなものがあります。
カレントディレクトリ(現在のディレクトリ)を表示する

- カレントディレクトリを変更する
- ディレクトリの内容を一覧表示する
- ディレクトリを作成する
- ディレクトリ名を変更する
- ディレクトリを別の場所に移動する
- ディレクトリを削除する

具体的な操作については、コマンド解説のページを参照してください。ディレクトリ操作命令については、それぞれ次のページに記載されています。

- 『pwd』コマンド　カレントディレクトリ(現在のディレクトリ)を表示する　110 ページ
- 『cd』コマンド　カレントディレクトリを変更する　75 ページ
- 『ls』コマンド　ディレクトリの内容を一覧表示する　100 ページ
- 『mkdir』コマンド　ディレクトリを作成する　103 ページ
- 『mv』コマンド　ディレクトリ名を変更する　106 ページ
- 『mv』コマンド　ディレクトリを別の場所に移動する　106 ページ
- 『rmdir』コマンド　ディレクトリを削除する　113 ページ

2.3.6　ファイルの保護モード

UNIX では、全てのファイル、ディレクトリについて、所有者はだれかという情報を持っています。また、ファイルの所有者はそのファイルに対して、誰が書き込みや読み込み、実行などの操作を行って良いのかなど、ぞれぞれのアクセス権(パーミッション)を設定することができます。

ファイルの保護は、次の３種類のカテゴリーで設定を行うことができます。

- そのファイルの所有者(user)に対してのもの
- ファイルの所有者と同じグループ(group)に所属している人に対してのもの
- それ以外の人(others)に対してのもの

✓　ただし、生田システムの現在の運用方針ではユーザーのグループ分けを行っていないため、group に対するパーミッションの設定はほとんど意味がありません。

また、それぞれのカテゴリー毎に設定できる許可内容としては次の３つがあります。

- 読み込みを許可するかどうか(r)

  ファイル　該当ファイルの中身を読み込んだり閲覧をしたりすることができるかどうかを設定できます。
  ディレクトリ　該当ディレクトリに含まれるファイルの一覧などを閲覧することができるかを設定できます。

- 書き込みを許可するかどうか(w)

  ファイル　該当ファイルへの書き込みができるかどうかを設定できます。

ディレクトリ　該当ディレクトリ以下に新たにディレクトリやファイルを作成したり、削除したり、編集したりできるかを設定できます。

- 実行を許可するかどうか(x)

ファイル　該当ファイルを実行できるかどうかを設定できます。
ディレクトリ　該当ディレクトリへ cd コマンドなどで移動できるかを設定できます。

ファイルやディレクトリに設定されているパーミッションは、『ls』コマンドなどによって確認することができます。

例として『ls』コマンドに『-l』というオプションを指定して実行をすると、次の図のような表示が得られます。

```
dbgsrv01% ls -l
合計 16
-rwxr-xr-x    1 od10001  assist        4688 Mar 20 14:07 a.out*
-rw-r--r--    1 od10001  assist          83 Mar 19 20:01 report.txt
-rw-r--r--    1 od10001  assist         219 Mar 20 14:07 sample.c
drwxr-xr-x    2 od10001  assist         512 Mar 19 20:08 test/
dbgsrv01%
```

各行には、スペースで区切られたフィールドが並んでいます。

一番右のフィールドがファイル名あるいはディレクトリ名です。

『report.txt』というファイルの詳細を見てみましょう。

左から３つ目のフィールドにある『od10001』という部分がこのファイルの所有者を表しています。

一番左のフィールドにある 10 バイトの文字列がパーミッションの状態を表しています。

－ｒｗ－ｒ－－ｒ－－　　１　ｏｄ０１０００１

パーミッションは、『読み出し許可、書き込み許可、実行許可』の３つで１セットになっており、これが『所有者、グループ、その他』について順番に表示されます。『r』は読み出し許可、『w』書き込み許可、『x』は実行許可です。『-』は許可が与えられていないこ

とを示します。(一番左のフィールドは別の意味を持っており、例えば『d』ならばディレクトリ、『l』ならばリンクファイルを意味しています)

『ls -l』コマンドの結果から、ファイル『report.txt』は、所有者に対しては『rw-』、グループに対しては『r--』、その他に対しても『r--』というパーミッションを持っていることが分かります。つまり『report.txt』のパーミッションは次のように設定されています。

- 所有者はこのファイルを読むことも書くこともできる。
- 所有者と同一のグループの人は、このファイルを読むことができる。
- その他の人は、このファイルを読むことができる。

ファイルを操作しようとして『アクセス権がありません』などのエラーが発生した場合には、ファイルやディレクトリのパーミッションを確認してみてください。

自分が所有者となっているファイルやディレクトリについては、パーミッションを好みの状態に変更することができます。ただし、適切でないパーミッションを設定してしまうと、ログインできなくなったり、動作がおかしくなったりすることもあるので、十分に注意した上で実行してください。パーミッションの変更方法についての詳細は76 ページの『ファイル、ディレクトリのパーミッションの変更』を参照してください。

2.4　コマンドシェル

2.4.1　コマンドシェル

コマンドシェル(Command Shell)は、利用者が入力したコマンド行を解釈し、その実行を UNIX に依頼するという役割を持っています。そのため、コマンドインプリタ(Command Interpreter)とも呼ばれます。生田システムの UNIX 環境で利用できるコマンドシェルには、Bourne シェル(sh)、C シェル(csh)、Korn シェル(ksh)、Bourne Again シェル(bash)などがありますが、生田システムが標準のシェルとして採用しているのは C シェルです。

C シェルは便利な機能をいくつも持っています。そして、UNIX を快適に操作するためにはこの C シェルの機能をうまく活用する必要があります。ここでは、その機能について順番に説明をしていきます。

2.4.2　標準入出力とは

UNIX では一般的に、プログラムに対する命令やデータの入力はキーボードから行い、プログラムの実行結果は画面(ディスプレイ)に表示されるように設定されています。しかし、入力はキーボードから、出力は画面に、というように固定されている訳ではありません。多くのプログラムは『標準入力』と呼ばれる仮想的な入力元から命令やデータを受け取るように、また、出力についても『標準出力』と呼ばれる仮想的な出力先に結果を書き出すようになっているだけですので、プログラム自身は、入力されたデータがキーボードから入力されたのかどうかについては判りませんし、そもそもそれを問題にしないように作られています。

しかし、通常は『標準入力はキーボード』、『標準出力は画面』というように UNIX が割り当てているため、特に何もしなければプログラムはキーボードから入力を受け、画面に出力をします。

このように、個々のプログラムが知らないところで、仮想的な標準入出力と、実際の入出力デバイス(機器)が結び付けられています。ですから、プログラムの外部でこの標準入出力と入出力デバイスの結びつきを変更してしまえば、プログラム自身には全く変更を加えずに実際の入出力を変更してしまうことができるのです。これをコマンドシェルから利用する機能が、リダイレクト機能とパイプ機能です。

2.4.3　リダイレクト機能

リダイレクト機能を使って、プログラムやコマンドの実行結果を画面ではなくファイルに書き出すには次のようにします。

```
% コマンド名 > 出力先ファイル名
```

例として、ディレクトリの中に含まれるファイルの一覧を表示するコマンド『ls』の出力を『out』というファイルに書き出す処理は次のようになります。

```
% ls
a.out*          report.txt    sample.c        test/
%
```

特に何も指定せずに『ls』コマンドを実行すると、結果は画面に表示されます。

```
% ls > out
%
```

リダイレクト機能である『>』を使ったので、結果は画面に表示されません。

```
% cat out
a.out*          report.txt    sample.c        test/
%
```

out ファイルの内容を表示

『cat』コマンドを使ってファイル『out』の内容を表示させてみました。『ls』コマンドの結果がファイルに書き出されていることが確認できます。

『>』で指定された出力先のファイルが存在していなかった場合には、新しくファイルが作成されてそこに書き込まれます。しかし、既にファイルが存在している場合には、内容が上書きされてしまい元の内容が失われてしまいます。

元の内容に上書きせずに追記したい場合には、次のように『>>』を使用してください。

% コマンド名 >> 出力先ファイル名

標準入力についても同じように変更することができます。リダイレクト機能を使って標準入力からファイルを読み込ませるようにするには、次のようにします。

% コマンド名 < 入力元ファイル名

例として、簡易な電卓プログラム『bc』コマンドへの計算命令をファイル『indata』から読み込ませる処理は以下のようになります。

```
% cat indata
3 + 5 * 6
%
```

『cat』コマンドでファイル『indata』の内容を表示。簡単な計算式が書かれている。

```
% bc < indata
33
%
```

電卓プログラム『bc』にファイル『indata』の内容をリダイレクト機能により入力。『bc』は入力された計算式を計算して『33』という答えを出力した。

標準入力と標準出力は、同時に変更することもできます。たとえば、『bc』コマンドにファイル『indata』から計算式を読み込ませ、結果をファイル『out』に書き出すには、次のような指定をします。

```
% cat indata
3 + 5 * 6
%
% bc < indata > out
%
```

標準出力も切り替えてしまっているので、画面には何も表示されない。

```
% cat out
33
%
```

計算結果がファイル『out』に書き出されていることを確認。

ここまで、標準入出力についての説明を行ってきましたが、出力については標準出力の他に、もうひとつ特別な出力があります。それが標準エラー出力です。プログラムの中には、通常のメッセージは標準出力に出力し、エラーなどに関する情報は標準エラー出力に出力するようになっているものがあります。デフォルトでは標準エラー出力も画面に表示されるようになっていますが、標準出力とは別の出力先に標準エラー出力を切り替えることもできます。そのためには次のようにします。

```
% コマンド名 >& 出力先ファイル名
```

例えば、『cat』コマンドを実行し、そのエラーメッセージをファイルに書き出す処理は次のようになります。

```
% cat uso.file
cat: uso.file : そのようなファイルやディレクトリはありません
%
```

存在していないファイルを指定したのでエラーメッセージが出力された。

```
% cat uso.file > error.msg
```

```
cat: uso.file : そのようなファイルやディレクトリはありません
```

『cat』コマンドのエラーメッセージは、標準エラー出力に出力されているので、通常のリダイレクト機能『>』では切り替えることができない。

```
% cat uso.file >& error.msg
&
& cat error.msg
cat: uso.file  : そのようなファイルやディレクトリはありません
%
```

今度は標準エラー出力の切り替えを行ったので、ファイルに結果が書き出されている。

2.4.4 パイプ機能

リダイレクト機能は、プログラムの入出力先とファイルを結びつけるものでしたが、パイプ機能はプログラム同士を結び付ける機能です。つまり、あるプログラムの標準出力を、そのまま別のプログラムの標準入力に結び付けてしまうのです。

パイプ機能を使って、コマンド１の出力をコマンド２の入力として渡したいときには次のようにします。

```
% コマンド１ | コマンド２
```

例として、カレントディレクトリのファイルの数を『ls』コマンドで出力し、その出力結果の行数を数える『wc』コマンドで数える処理は以下のようになります。

```
% ls -1
512.sh*
512.sh~
522.sh*
522.sh~
perl5/
test
test223-1
test314
test322
test323
test323~
test324
ダウンロード/
テンプレート/
デスクトップ/
```

```
ドキュメント/
ビデオ/
音楽/
画像/
公開/
% ls -1 | wc -l
20
%
```

『ls』コマンドが１行に１つずつファイル名を出力し、その行数を『wc』コマンドが数えた。

パイプは２つのコマンドをつなぐだけではなく、『% **コマンド１ | コマンド２ | コマンド３**』のように複数つなげることもできます。

2.4.5 メタキャラクタとは

C シェルはコマンド行でファイルやディレクトリを指定するときに、メタキャラクタという特殊な文字を使用することで、効率的な指定ができるようになっています。

メタキャラクタには次のようなものがあります。

- ?(クエスチョンマーク)

  『?(クエスチョンマーク) 』は任意の１文字と一致します。

  例を挙げると、test1.txt、test2.txt、test3.txt という３つのファイルがカレントディレクトリにあり、これをコマンドの引数としたい場合に、『test1.txt test2.txt test3.txt』のようにひとつひとつを指定することもできますが、これでは文字数が多くなってしまい入力ミスも発生しやすくなってしまいます。そこで『?』を利用することで先ほどの長い表記を『test?.txt』と短くすることができます。

  ただし、『?』は任意の１文字と一致してしまうので、先ほどの３つのファイル以外にも、『testA.txt』などのファイルが存在していた場合、それも指定してしまいます。

- *(アスタリスク)

  『*(アスタリスク)』は任意の長さの文字列と一致します。

  例を挙げると、カレントディレクトリに test.c、test.txt、test.memo という３つのファイルがあった場合、『test.*』と指定すれば３つのファイルを簡単に指定することができます。ただし、『*』は一致する文字の文字数を指定することができませんので、『３文字の任意の文字と一致するもの』というような指定をしたい場合には、『test.???』というような指定にします。

- [ ](ブラケット、角括弧)
  『[ ](ブラケット、角括弧)』は『[ ]』内に列挙した文字のどれかと一致します。例を挙げると、『[ph]op』と指定すると、『pop』または『hop』と一致します。この他、『[a-z](アルファベットの小文字のどれか１つ)』や『[0-9](数字の 0～9 のどれかひとつ)』というように文字の範囲の始まりと終わりを『-』でつないで指定することによって文字の範囲の指定をすることもできます。
  また、範囲指定と個別文字指定を組み合わせて『[a-z0-9.,](アルファベットの小文字、数字、.(ドット)、,(カンマ))』のように、いろいろな文字の中のどれか１つと一致させるような指定をすることもできます

メタキャラクタを使った効率的なファイルの取り扱いの例を次に挙げておきますので、それぞれの箇所でどのように指定をしているのかを考えてみてください。

```
% ls
14-2.png    15-3.png    16-3.png    25-1.png    26-1.png    test.c
15-1.png    16-1.png    24-1.png    25-2.png    sample.c    test.f
15-2.png    16-2.png    24-2.png    25-3.png    sample.f
% ls 25-?.png
25-2.png    25-2.png    25-3.png
% ls [a-z]*
sample.c    sample.f    test.c    test.f
% ls *c
sample.c    test.c
% rm *c
% ls *c
ls: No match.
% ls
14-2.png  15-2.png  16-1.png  16-3.png  24-2.png  25-2.png  26-1.png  test.f
15-1.png  15-3.png  16-2.png  24-1.png  25-1.png  25-3.png  sample.f
%
```

2.5　環境のカスタマイズ

2.5.1　コマンドサーチパス

例として、ユーザーがコマンドプロンプトに対して、次のコマンドを入力したとします。

```
% cd
```

このように入力して『Enter』キーを押すと、『cd』コマンドが実行されてホームディレクトリに移動します。実はこの『cd』というコマンドは、ひとつの実行ファイルになっていて、『/usr/bin』ディレクトリに存在しています。『cd』コマンド以外のほとんどのコマンドも同じようにそれぞれが実行ファイルになっていて、ディレクトリのツリー構造の中のどこかに置かれています。ですから、本来コマンドを実行する場合には次のように入力する必要があります。

```
% /usr/bin/cd
```

このように、目的のコマンドがどこにあるのかを正確に指定しなければなりません。しかし、標準的なコマンドについては、コマンド名を入力しただけで目的の実行ファイルを見つけて実行してくれます。これを補助しているのが、コマンドサーチパスという特別な変数です。

C シェルの場合には、『path』という名前になっています。現在『path』に設定されている値を確認するためには次のように入力します。

```
% echo $path
```

これを実行すると、次のように出力されます。

```
% echo $path
/usr/meiji/bin /usr/kerberos/bin /usr/local/bin /usr/bin /bin /usr/X11R6/bin
/home/USERNAME/bin
%
```

絶対パスの形式で、多数のディレクトリ名が設定されているのが分かります。C シェルはコマンドラインから、『**% コマンド名**』などと指定されると、指定されたファイル名の実行ファイルが存在するかどうか、この『path』変数に設定されているディレクトリの中から探します。探す順番は、『path』変数に設定されているディレクトリ順となりま

す。目的の実行ファイルが見つかればそれを実行し、もしも最後のディレクトリまで探しても見つからなかった場合には、エラーを出力します。

生田システムの UNIX 環境に存在する実行ファイルであっても、『path』変数にその存在場所のディレクトリが登録されていないと、C シェルはそのコマンドを探しだすことができません。

生田システムでは、標準的なコマンドや、よく使われるコマンドについては、あらかじめ『path』変数に設定してありますが、それ以外のものについては登録してありません。これは次のような理由があるからです。

- ありとあらゆるディレクトリを登録しておくと、シェルがコマンドを探し出す際に効率が悪くなる。
- UNIX では、同一ファイル名でありながら全く違うプログラムが存在する。この場合、その同一名称のプログラムの内のどれを実行したいのかは、利用者一人ひとりで異なっているため、生田システム側では一律に決められない。

そのため、自分がよく使うコマンドのディレクトリが、生田システムの標準設定では設定されていない場合には、各自で設定を追加してください。

設定を追加するためには、次のコマンドを実行します。

```
% set path=( 追加設定したいディレクトリ名 $path )
```

設定ができたら、『echo $path』と入力して、設定が間違いなく反映されているかを確認してみてください。

> ✓ ここで、単に『% set path=(追加設定したいディレクトリ名)』などとしてしまうと、標準設定のコマンドサーチパスが全て上書きされてしまいますので、必ず『$path』を付けるようにしてください。

こうして行った設定ですが、その設定したウィンドウを閉じてしまったり、ログアウトしてしまったりすると元に戻ってしまいます。これを防ぐために、ログイン時に毎回自動的に『path』への追加が行われるようにするための方法があります。その方法については、56 ページの『2.5.3 環境の自動設定』を参照してください。

2.5.2 コマンドのエイリアス

コマンドを指定する時には、そのコマンドのファイル名を正確に指定しなければなりません。しかし、ファイル名が長いコマンド、例えば『euctojis』などの場合、毎回入力をするのが面倒になってしまいます。また、特定のコマンドを実行するときには、いつも同じオプションを指定して実行したいということもあるでしょう。このような場合に、それらを短いコマンドで実行できたらとても便利です。

C シェルには、これを実現する『alias(エイリアス)』という機能があります。『alias』を設定するには、次のように入力します。

```
% alias エイリアス名 実行したいコマンド名とそのオプション
```

また、設定をしたエイリアス名を削除したい場合には、次のように入力します。

```
% unalias エイリアス名
```

『path』と同様に、コマンドの『alias』もログアウトなどをしてしまうと削除されてしまいます。『alias』を自動的に設定するための方法については、56 ページの『2.5.3 環境の自動設定』を参照してください。

2.5.3 環境の自動設定

『path』変数へのコマンドサーチパスの追加や、コマンドの『alias』の設定などは、UNIX からログアウトしてしまうと全部初期化されてしまい、次にログインをした時にはまた最初から設定をやりなおさなければなりません。しかし、これでは効率的に使うために用意された『path』や『alias』の機能が生かせません。そこで、毎回使う設定などについては、ログイン時などに自動的に読み込まれるようにすることができます。

具体的には、各ユーザーのホームディレクトリにあらかじめ決められた名前のファイル名でファイルを置いておくと、ログイン時やコマンドツールなどの端末エミュレータ起動時に、シェルが自動的にそのファイルを実行してくれるようになっています。

C シェルは、この目的のために２つのファイルを使います。

- 『.login』ファイル
  ログイン時に１度だけ C シェルによって読み込まれ、実行されるファイル。
- 『.cshrc』
  C シェルが起動される度に自動的に読み込まれ、実行されるファイル。『path』や『alias』の設定は、このファイルに記述をして置くといいでしょう。

みなさんのホームディレクトリにあらかじめ存在している『.login』や『.cshrc』ファイルには、生田システムの環境を利用するために必要な設定が書き込まれています。

この設定が有効になっていないと、利用に支障をきたす可能性がありますので、くれぐれも、あらかじめ設定されている部分を削除したり変更したりしないでください。個人のカスタマイズ内容については『.login』や『.cshrc』の最後に追加してください。

✓　みなさんのホームディレクトリには、これ以外にもファイル名が『.（ドット）』で始まるファイルやディレクトリが多数存在します。これはほとんどの場合、システムや各種プログラムが利用する設定ファイルや、データの保存場所です。

ですから不用意に削除あるいは変更してしまうと、それを利用するプログラムが動かなくなってしまうこともありますので、それぞれのファイルやディレクトリが何に利用されているものなのか理解できるまでは、削除や変更をしないようにしてください。

第３章　　テキストエディタについて

3.1　テキストエディタの種類

生田システムの UNIX 環境では、いくつかのテキストエディタが利用できるようになっています。

特に凝った作業を行うのでなければ、『第 1 章 UNIX の基本的な使い方』で紹介をした GNOME デスクトップ標準のテキストエディタで十分です。また Windows のテキストエディタでファイルを作成し、UNIX へファイルを転送することもできます。

ですが、もっと高度なことを行いたいということもあるかと思います。ここでは、そのような場合に有用なテキストエディタの利用方法について説明していきます。

3.1.1　Emacs について

テキストファイルを作成するに当たって、高度な編集作業を UNIX 環境上で行いたいのであれば、『Emacs』というテキストエディタが便利です。ただし、『Emacs』は単なるテキストエディタではなく、メールの送受信、プログラムのコンパイル作業、画像ファイルの表示など、非常にたくさんの機能を持っています。1 個の独立した統合作業環境と言っても過言ではありません。そのため、『Emacs』の機能を使いこなせるようになるためには、それなりの練習が必要になります。

また、操作方法にも独特な癖がありますので、利用を始めたばかりのうちはかなり戸惑う事と思いますが、慣れてくると非常に快適に各種操作ができるようになりますので、UNIX 環境を使いこなしてみたいという方は、是非ともチャレンジしてみてください。

3.1.2　キー操作の表記について

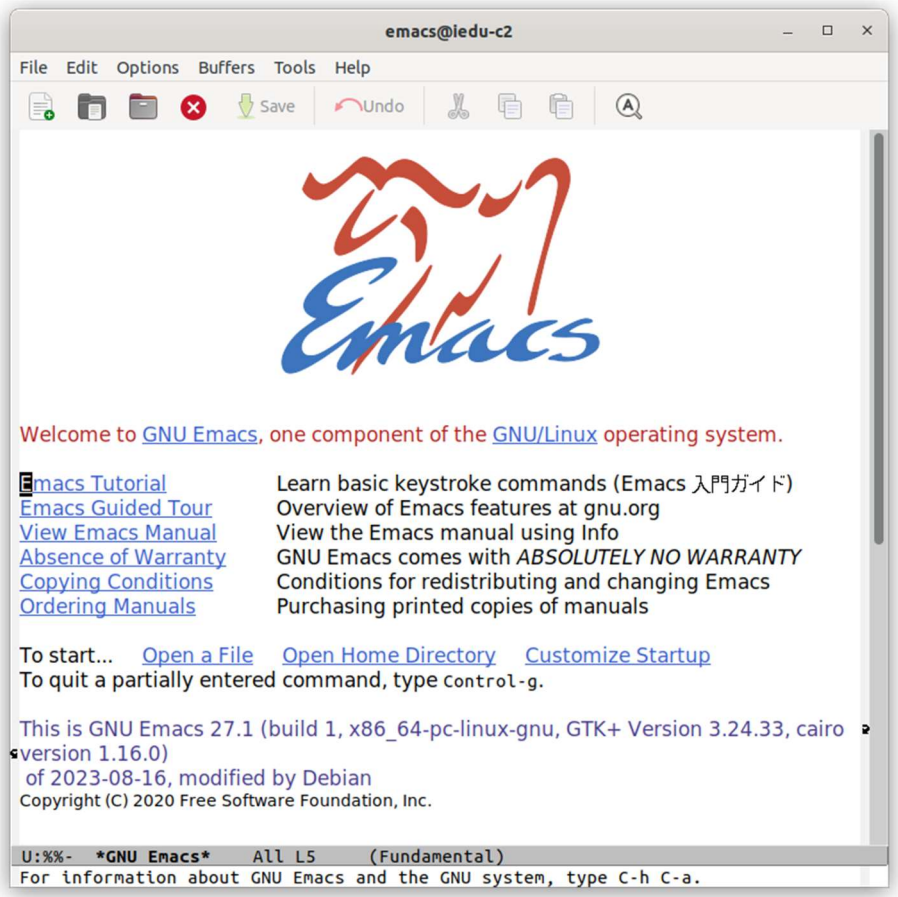
『Emacs』でよく利用するキー操作について、以降は以下のような表記をします。(この表記は、『Emacs』内のマニュアルの表記に準じています)

表記	操作の内容
C - y	『Ctrl』キーを押しながら『y』キーを押す
C - x , u	『Ctrl』キーを押しながら『x』キーを押し、いったん両方のキーから指を離し、続けて『u』キーを押す
C - x , C - s	『Ctrl』キーを押しながら、『x』キー、『s』キーの順に押す
M - y	『Esc』キーを押して指を離し、次に『y』キーを押す

3.2　Emacs の起動と終了

『Emacs』は、コマンドプロンプトに『emacs』と入力し、『Enter』キーを押すか、もしくは、メニューバーの『アプリケーション』→『アクセサリ』→『Emacs』で起動することができます。

『Emacs』を起動すると、次のようなウィンドウが表示されます。



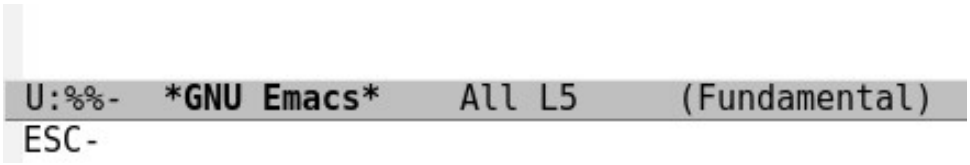
時代の流れもあり、『Emacs』もパソコン用のワープロやテキストエディタのように、メニューバーやツールバーから各種の操作ができるようになってきています。しかし、各種機能を呼びたすのに、いちいちマウスを操作してメニューバーからコマンドを選んだりしているのでは、『Emacs』を利用する意味があまりありません。ほとんどの操作をキーボードの上に両手を置いたままでできるところに『Emacs』の意味があるので、『Emacs』を利用するのであれば、是非そのレベルまで操作に習熟するように努力してみてください。

起動している『Emacs』を終了するには、『Emacs』のウィンドウがアクティブになっている状態で、『C－x , C－c』を入力します。

3.2.1　Emacs のチュートリアルファイルを使って操作を練習する

『Emacs』は、初心者が自分で操作方法を勉強するためのチュートリアルファイルを内蔵しています。『Emacs』をはじめて利用する方は、まずこのファイルを使って練習することから始めると良いでしょう。

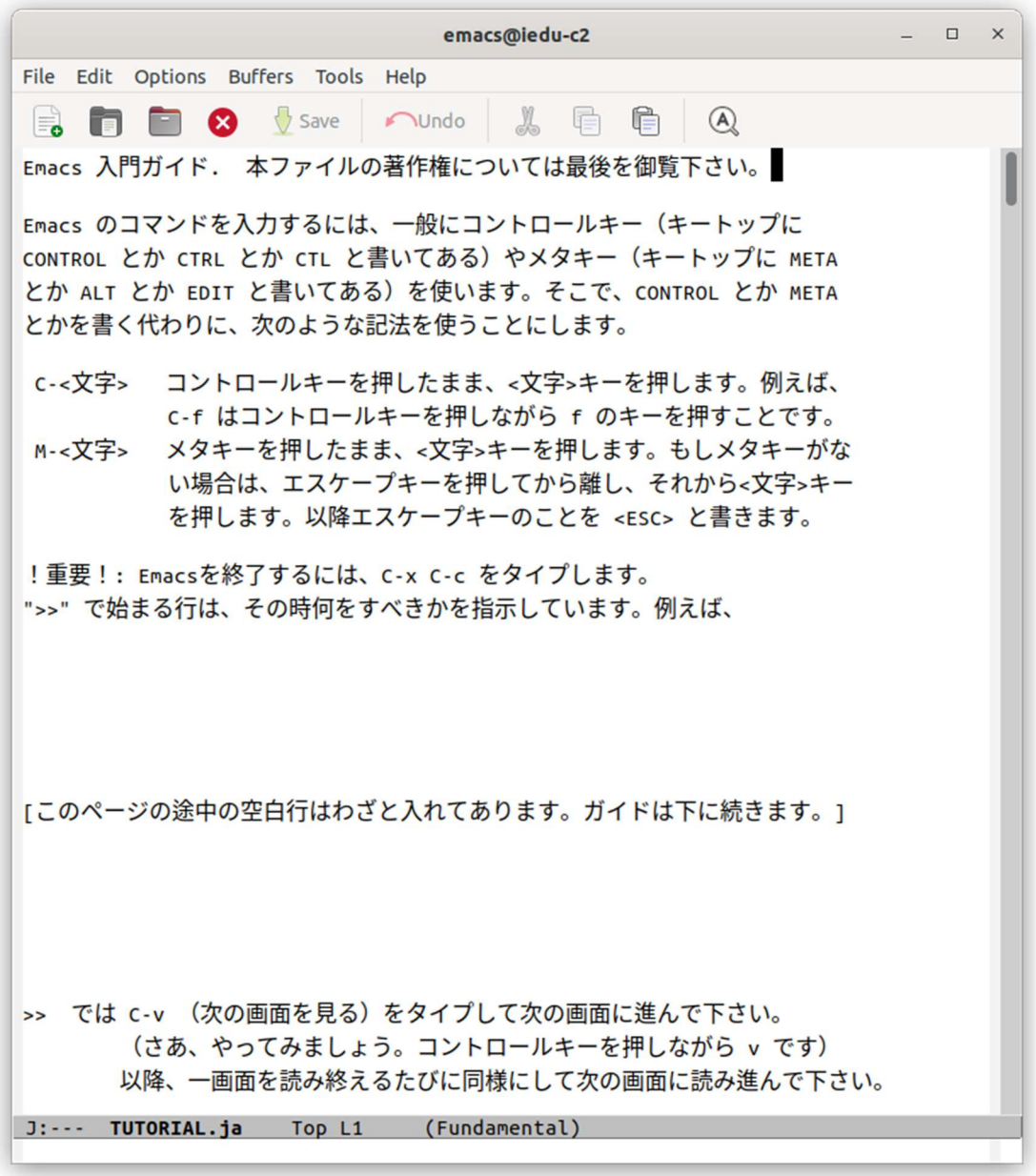
チュートリアルファイルを表示するには、まず『Esc』キーを１度押します。すると、『Emacs』のウィンドウの最下段に『ESC-』と表示されます。



続けて『x』キーを押すと、表示が『M - x』に変化します。

次に『help-with-tutorial』と入力し『Enter』キーを押します。

正しく入力できていれば、ウィンドウにチュートリアルファイルが表示されます。

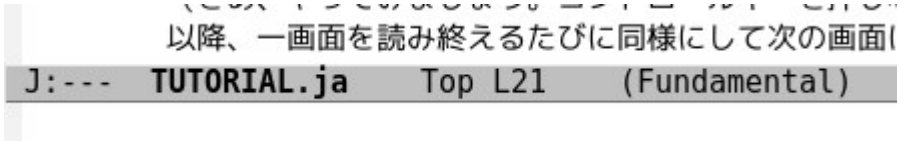


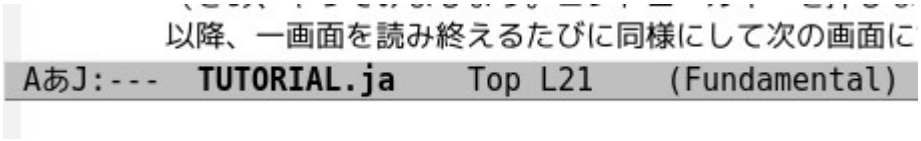
✓ 『Emacs』がとにかく初めてでキーボード操作もよく分からない、という方は『Emacs』ウィンドウの最上段(ツールバー)右端にある『Help』メニューの中から『Emacs Tutorial』を選択して、チュートリアルファイルを表示させてください。

以降は、チュートリアルファイルの指示に従い、『Emacs』の操作練習を行ってみてください。

3.3　Emacs 上での日本語入力

『Emacs』上で日本語入力を行う場合には、『C－¥』を入力します。すると、『Emacs』の下部に表示されているステータスバーの表示が次のように変化します。





これで日本語入力モードに切り替わったことが分かります。もう１度『C－¥』を押せば、日本語入力モードから直接入力モードに切り替わります。

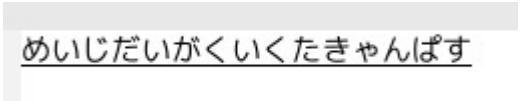
基本的な日本語変換操作は次の通りです。

キー操作	動作
Space(スペース)キー	次の候補に変換
C－n	次の候補に変換
C－p	ひとつ前の候補に変換
C－o	変換対象となっている文節を１文字伸ばす
C－i	変換対象となっている文節を１文字縮める
C－f	変換対象文節をひとつ右の文節に移す
C－b	変換対象文節をひとつ左の文節に移す
M－h	変換対象文節をひらがなに変換する
Shift－k	変換対象文節をカタカナに変換する
M－s	変換候補の一覧を表示する

実際の操作を次に示していきます。

日本語入力モードに切り替えてから、「めいじだいがくいくたきゃんぱす」と入力します。

入力した文字列に下線が引かれていて、それ以外の文字と区別されています。



ここで、『Space』キーを１度押して変換してみると、次のように漢字に変換されますが、文節も漢字も正しく変換できていない状態になります。

明事大がくいくたきゃんぱす

最初の文節は『めいじ』としたいので、『C - i』を押して文節を２文字縮めます。
すると、文節が縮まったと同時に意図したとおりに自動変換されます。

明治だいがくいくたきゃんぱす

次の文節に移るために『C - f』を押します。これでもし変換が意図したものでない場合には、意図した候補になるまで『Space』キーを押してください。

明治大学いくたきゃんぱす

意図したとおりの変換ができ次第、『C - f』を押し次の文節に移り、意図した候補になるまで『Space』キーを押します。

明治大学生田侠ぱす

『C - o』を押して『きゃんぱす』が文節になるようにします。次に『Space』キーではカタカナ変換ができないため、『Shift - k』を押し、カタカナ変換を行います。

明治大学生田キャンパス

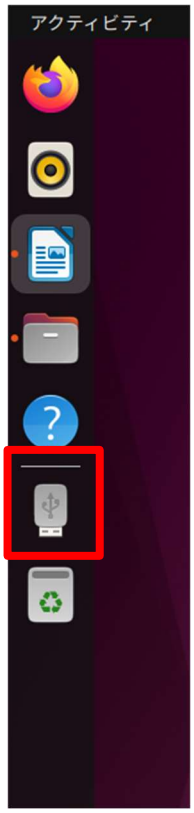
明治大学生田キャンパス

変換に問題がないことを確認し、『Enter』キーを押せば変換完了です。

第４章　　便利な使い方

4.1　USB メモリの利用

Ubuntu でも USB メモリを利用することができます。
USB メモリをセットすると自動的に画像のようにマウントされます。

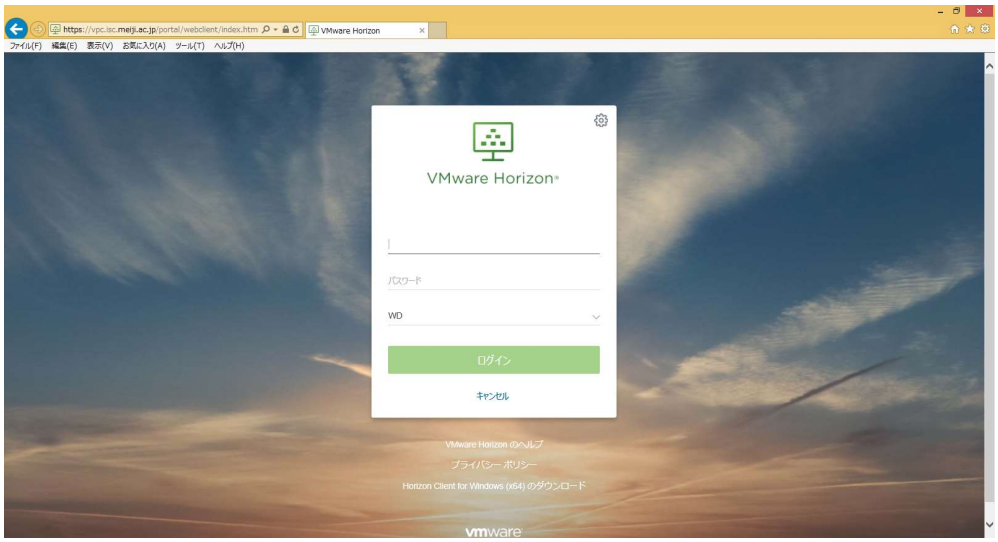


USB メモリのアイコンをクリックするとフォルダが開きます。
取り外す際にはデータの通信中でないことを確認し、アイコンを右クリックし、メニューの中から「取り出す」を選択するとアンマウントされるので安全に取り外しができます。

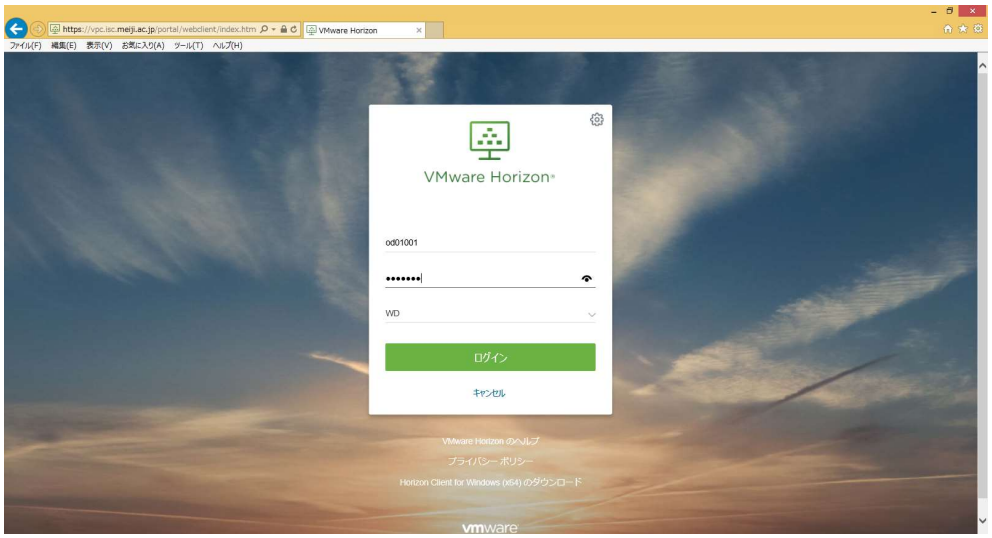
4.2　生田仮想デスクトップPCの利用

生田仮想デスクトップPCとは、学内に設置されているサーバーシステム上に構築された仮想PCを、自分のPCやタブレット端末等を用いて、遠隔操作するサービスのことをいいます。生田仮想デスクトップPCを用いることで自宅でもUbuntuの環境を利用することが可能です。
なお、Ubuntuは同時接続数の制限（50台まで）があるので、利用者が多い際、接続できない場合もあります。

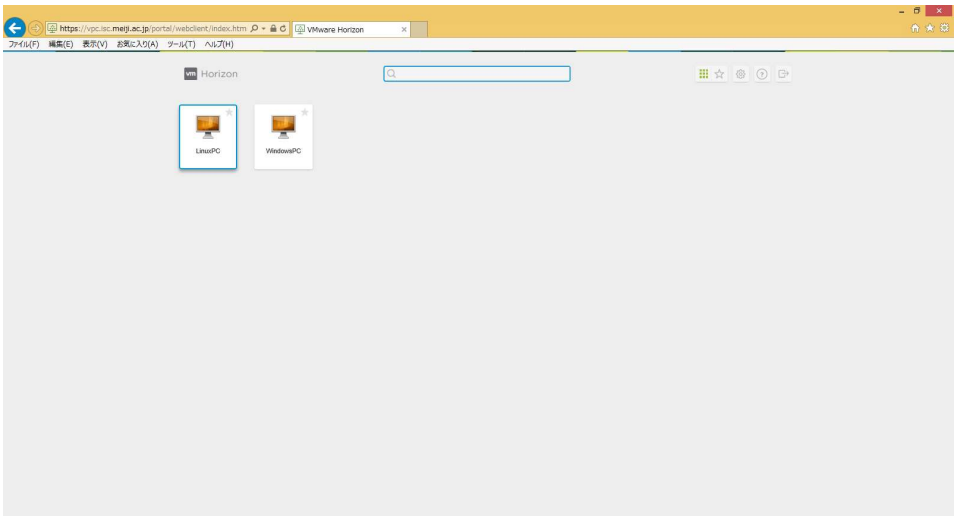
- ブラウザやView Clientより、https://vpc.isc.meiji.ac.jp/　にアクセスしてください。
（ここではブラウザでの利用について紹介します）

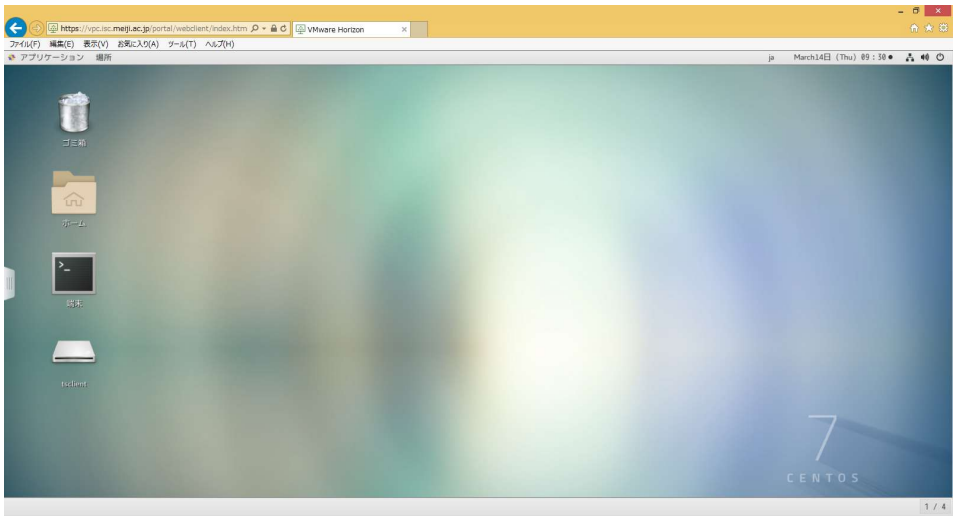


- ログイン名（ユーザID）とパスワードを入力してください。



- 「LinuxPC」を選択することで Ubuntu を利用することができます。





✓ クライアントからの利用など生田仮想デスクトップの詳細は下記の HP をご覧ください。
https://www.meiji.ac.jp/isys/vdesktop/index.html

第５章　　主要コマンド解説

alias

コマンド名のエイリアス(別名)を設定する

書式

alias　**エイリアス名　エイリアスに設定するコマンド名とオプション**

使い方

- ディレクトリの作成を『md』というコマンド名で実行できるようにする

```
% ls
a.out*      test.c
%
```

カレントディレクトリは現在このような状態になっている。

```
% md test
md: コマンドが見つかりません。
%
```

『test』ディレクトリを作成しようとしたが、『md』というコマンドは無いのでエラーになる。

```
% alias md mkdir
%
```

『mkdir』コマンドに、『md』というエイリアスを設定する。

```
% md test
%
```

今度はエラーにならずに実行される。

```
% ls
a.out*      test/      test.c
%
```

『test』ディレクトリが作成されている。

関連項目：『unalias』119ページ

bg

フォアグラウンドで中断しているジョブをバックグラウンド実行状態に持っていく

書式

bg [%ジョブ ID]

使い方

- フォアグラウンドで実行しているジョブをバックグラウンドに持っていく

% kterm

『kterm』をフォアグラウンドで実行中。

^Z

停止(利用者要求による)

%

『Ctrl+z』を入力して、中断状態にする。

% bg

[1] + kterm &

%

中断状態だった『kterm』をバックグラウンドで実行する。

関連項目:『fg』86 ページ、『jobs』95 ページ

cat

ファイルを連結する

ファイルの内容を表示する

書式

cat ［ファイル名 …］

使い方

● ファイルの内容を表示する

% cat study.memo

ファイル『study.memo』の内容が画面に表示される。

%

● ファイルを連結し、その内容を画面に表示する

% cat question.txt answer.txt

ファイル『question.txt』と『answer.txt』の内容を連結したものが画面に表示される。

%

● ファイルを連結し、その内容を別のファイルに出力する

% cat file1.memo file2.memo > file.new

『file1.memo』と『file2.memo』を連結した内容のものが画面に表示される。

%

✓ 『cat』コマンド自体にはコピーするという機能は無いので、シェルが持つリダイレクト機能を併用してコピーを実現します。

『cat』というコマンド名は、『concatenate』(連結する、鎖状につなぐ)という言葉から来ています。これで分かるように『cat』コマンドの本来の目的は、『ファイルを連結する』というものですが、結果を標準出力に出力するようになっているため、ファイルの内容を画面に表示するためにも使われています。

関連項目：『リダイレクト機能』48 ページ

cd

カレントディレクトリを変更する

指定したディレクトリに移動する

ホームディレクトリに移動する

書式

cd ［ディレクトリ名］

使い方

- カレントディレクトリを変更する

% cd work

カレントディレクトリを『work』に変更する。

- 指定したディレクトリに移動する

% cd test

ディレクトリ『test』に移動する。

✓ 指定したディレクトリに移動すると、そこがカレントディレクトリ(現在のディレクトリ)になるので、カレントディレクトリを変更することと指定したディレクトリに移動することは、同じ動作になります。

- ホームディレクトリに移動する

% pwd
/home/od10001/Mail

カレントディレクトリは、『/home/od10001/Mail』ディレクトリ。

% echo $HOME
/home/od10001

このユーザーのホームディレクトリは、『/home/od10001』。

% cd
% pwd
/home/od10001
%

移動先のディレクトリ名を指定せずに『cd』を実行すると、ホームディレクトリに移動する。

✓ ユーザーのホームディレクトリは環境変数『$HOME』に設定されていますので、『cd $HOME』を実行してもホームディレクトリに移動できます。
環境変数『$HOME』に設定されている内容は、『echo $HOME』などとすれば確認できます。

関連項目：『pwd』110 ページ、『UNIX のファイルシステム』41 ページ

chmod

ファイル、ディレクトリのアクセス権の変更

ファイル、ディレクトリのパーミッションの変更

書式

chmod [-R] **アクセス権指定　ファイルまたはディレクトリ名**

オプション

-R　　ディレクトリに対して『chmod』コマンドを実行した場合に、指定したディレクトリだけではなく、そのディレクトリに含まれているファイルやディレクトリに対しても、指定したアクセス権に変更する。

使い方

『chmod』コマンドは、ファイルやディレクトリのアクセス権の変更に使用します。このコマンドを使うと、ファイルを読み取り専用(書き込みができない)状態にしたり、実行可能状態にできたりします。また、アクセス権は、そのファイルやディレクトリの所有者、同じグループに所属するユーザー、その他のユーザーの３つのカテゴリーのユーザー毎に設定することができます。

- ファイルを読み込み専用にする

```
% ls -l test.txt
-rw-r--r-- 1 od10001 assist 21 Mar 18 15:21 test.txt
```

ファイルの所有者『od10001』に対して、書き込み権『w』が付いている。

```
% chmod u-w test.txt
```

ユーザー(u)の書き込み権を取り除く『-w』。

```
% ls -l test.txt
-r--r--r-- 1 od10001 assist 21 Mar 18 15:21 test.txt
%
```

ユーザーの書き込み権がなくなり、ファイルは読み取り専用になった。

- ファイルを実行可能にする

```
% cat lsl
#!/bin/sh
ls -l
```

ファイル『lsl』の中身を表示。

```
% ls -l lsl
-rw-r--r-- 1 od10001 assist 15 Mar 18 15:48 lsl
```

『lsl』は現時点では実行可能な状態ではない。(『x』が付いていない)

```
% ./lsl
lsl: 許可がありません。
```

実行可能な状態ではないので、実行しようとしてもエラーになる。

```
% chmod u+x lsl
```

ファイル『lsl』に、実行権『+x』を付ける。

```
% ls -l lsl
-rwxr--r-- 1 od10001 assist 15 Mar 18 15:48 lsl*
```

実行可能な状態になった。

```
% ./lsl
合計 4
-rwxr--r-- 1 od10001 assist 15 Mar 18 16:10 lsl*
-r--r--r-- 1 od10001 assist 21 Mar 18 16:10 test.txt
%
```

ファイル『lsl』が実行ファイルとして認識されたので、その中に記述されているコマンド『ls -l』が実行された。

- ✓ ただし、このように実行可能な状態にするファイルは、その中身も正しく実行できるものである必要があります。普通の文書や、C 言語のソースファイル、画像ファイルなど、どんなファイルにも『chmod』コマンドで実行権を与えることはできますが、中身が正しく実行できるように書かれたファイルで無ければ、実行時にエラーとなってしまいます。

関連項目：『ls』100 ページ、『ファイルの保護モード』45 ページ

clear

端末の画面をクリアする

書式

clear

使い方

『clear』コマンドは、現在使っているターミナルの画面をクリアします。端末エミュレータや『Terminal』の中から『clear』コマンドを実行すると、それまで表示されていた内容がクリアされて、最上段にコマンドプロンプトが表示されます。

ただし、端末エミュレータなどで、スクロールバーが付いていて、画面外に消えてしまった内容がスクロールさせると見えるようになっている場合は、その部分についてはクリアされません。あくまでも、現在画面で表示されている１画面分がクリアされるだけです。

- 画面をクリアする

% clear

端末画面がクリアされ、最上段にコマンドプロンプトが表示される。

cp

ファイルをコピーする

書式

cp [-fir] **コピー元ファイル名　コピー先ファイル名**

オプション

-f	コピー先のファイルが読み取り専用であっても、強制的に上書きコピーする。
-i	コピー先ファイルが既に存在していた場合に、上書きコピーをしていいかどうかの確認を求められる。
-r	コピー元ファイルにディレクトリを指定した場合、そのディレクトリに含まれる全てのファイルとディレクトリもそのままコピーする。

使い方

- ファイル『test.txt』を、新規ファイル『test.copy』にコピーする

```
% ls
test.txt
```

『test.txt』しか存在していない。

```
% cp test.txt test.copy
```

ファイル『test.txt』が『test.copy』にコピーされる。

```
% ls
test.copy test.txt
%
```

- ファイル『test.txt』を、すでに存在している『test.copy』に上書きコピーする

```
% ls
test.copy test.txt
% cp test.txt test.copy
```

『test.txt』の内容で『test.copy』が上書きされる。

- ファイル『test.txt』を、既に存在している読み取り専用ファイル『test.copy』に上書きコピーする

```
% ls
test.copy test.txt
% ls -l test.copy
-r--r--r--  1 od10001 assist  21 Mar 18 16:44 test.copy
```

『test.copy』は読み取り専用ファイル。

```
% cp test.txt test.copy
cp: cannot create regular file 'test.copy': 許可がありません
```

読み取り専用ファイルのため、そのままではコピー(上書き)できない。

```
% cp -f test.txt test.copy
```

%

『-f』オプションを指定することにより、読み取り専用ファイルに対しても強制的に上書きできる。

● ディレクトリ『test』の中身をまるごとディレクトリ『test2』にコピーする

```
% ls -F
test/
```

ディレクトリ『test』がある。

```
% ls test
test.copy test.txt
% cp -F test test2
% ls -F
test/ test2/
```

ディレクトリ『test2』が作成されている。

```
% ls test2
test.copy test.txt
```

ディレクトリ『test』の中身もコピーされている。

✓ 『-r』オプションを使用する場合、コピー先に指定したディレクトリが存在しているかどうかで、動作が変わることに注意してください。コピー先ディレクトリが存在しない場合には、先ほどの実行例のように『指定されたディレクトリを作成した上で、その中にコピー元ディレクトリの中身をコピーする』というような動作になります。この場合、コピー元とコピー先のディレクトリの内容は全く同じものになります。

しかし、コピー先ディレクトリが既に存在していた場合には、『コピー先ディレクトリの中に、コピー元ディレクトリごとコピーを行う』という動作になります。先ほどの例で言えば、ディレクトリ『test2』の中にディレクトリ『test』が作成されその中にディレクトリ『test』が作成されその中にディレクトリ『test』の中身がコピーされることになります。

関連項目：『UNIX のファイルシステム』41 ページ

date

日付と時刻を表示する

書式

```
date
```

使い方

- 現在の日付と時間を表示する

```
% date
Wed Apr 20 14:26:19 JST 2011
```

現在の日時は、2011 年 4 月 20 日(水曜日)の 14 時 26 分 19 秒である。

```
%
```

diff

２つのテキストファイルの相違点を表示する

書式

diff　ファイル名１　ファイル名２

使い方

- ファイル『test1.txt』と『test2.txt』を比較し、相違点を表示する

```
% cat test1.txt
IBM
Microsoft
SONY
```

ファイル『test1.txt』の中身。

```
% cat test2.txt
IBM
Apple
SONY
```

ファイル『test2.txt』の中身。

```
% diff test1.txt test2.txt
2c2
< Microsoft
---
> Apple
%
```

二つのファイルを比較した結果、それぞれのファイルの２行目の『Microsoft』と『Apple』に違いがあることが表示される。

✓　比較をする２つのファイルの内容が全く同じである場合に『diff』コマンドで比較を行うと、何も表示されずにコマンドプロンプトに戻ってしまいます。これは、歴史的な事情などにより、UNIX では正常に実行された場合には余計なメッセージを出さないコマンドが多く、それがUNIX のひとつの文化となっているからです。

du

ディスクの使用状況を表示する

ディスクをどのくらい使用しているか表示する

書式

du [-ks] [ディレクトリ名 …]

オプション

-k　　ファイルのサイズを 1024 バイト(1K バイト)単位で出力する。このオプションを指定しないと、512 バイト単位での出力となる。

-s　　指定したディレクトリが使用しているディスク使用量の合計サイズだけ表示する。

使い方

● カレントディレクトリの使用状況を 1K バイト単位で表示する

```
% pwd
/home/od10001/test
% ls -F
test.copy      test.txt      test2/
% du -k
5         ./test2
8         ./
```

ディレクトリ『test2』のディスク容量は 5K バイトで、『test2』を含むカレントディレクトリ(./)のディスク使用量は 8K バイト。

● カレントディレクトリの使用状況の合計だけを 1K バイト単位で表示する

```
% pwd
/home/od10001/test
% ls -F
test.copy      test.txt      test2/
% du -ks
8         ./
```

カレントディレクトリ(./)のディスク使用量の合計は 8 バイト。

✓ 管理用途などには、デフォルトの 512 バイト単位での出力のほうが便利なことが多いのですが、一般ユーザーが自分が使用しているディスク容量を知る場合には、-k オプションを指定して 1K バイト単位の出力にしたほうが分かりやすいでしょう。

echo

指定した文字列(メッセージ)を表示する

書式

echo ［文字列］

使い方

- 指定した文字列を表示する

```
% echo Message
Message
%
```

指定した文字列『Message』が表示される。

- メタキャラクタなどシェルが使う特殊な文字を含む文字列を表示する

```
% echo How are you?
echo: 照合パターンに合いません
```

指定した文字列の中に、シェルが扱う特別な文字であるメタキャラクタ『?』が含まれている。この場合、『echo』コマンドに渡される前に、シェルによって解釈されてしまう。今回の場合は、シェルはファイル名が『you2』のような『you＋任意の一文字』のファイルを指定されたと解釈し、条件に見合うファイルを探そうとして見つからなかったためにエラーの出力を返してきた。

```
% echo 'How are you?'
How are you?
```

文字列を『'(シングルクォーテーション)』で囲むことで、先ほどのようなエラーを防ぐことができる。

✓ 実際には『echo』コマンドを手動で実行することはほとんどありません。主な用途は、シェルスクリプトファイルの中で動作チェックをしたり、現在実行している内容を使用者にアナウンスしたりするためのメッセージを画面に表示する等です。

env

一時的に環境変数を指定してコマンドを実行する

現在設定されている環境変数をすべて表示する

書式

env ［環境変数設定］［コマンド名］

使い方

- 一時的に環境変数『LANG』を変更してコマンドを実行する

```
% echo $LANG
ja_JP.eucJP
```

環境変数『LANG』は、言語環境を指定するもので、『ja_JP.eucJP』は日本語を意味している。環境変数『LANG』に対応しているコマンドの場合、セットされている値(言語環境)に合わせて、メッセージの言語を変えたりできる。

```
% ls -l
合計  4
-rw-r--r--  1 od10001 assist   17 Mar 18 18:01 test1.txt
-rw-r--r--  1 od10001 assist   13 Mar 18 18:02 test1.txt
```

環境変数『LANG』が『ja(日本語)』となっているのでメッセージが日本語(『合計』の部分)で表示されている。

```
% env LANG=C ls -l
total  4
-rw-r--r--  1 od10001 assist   17 Mar 18 18:01 test1.txt
-rw-r--r--  1 od10001 assist   13 Mar 18 18:02 test1.txt
```

『env』コマンドで、一時的に環境変数『LANG』の値を『C(英語)で表示』に変更して『ls』コマンドを実行したので、メッセージが英語(『total』の部分)で表示された。

- 現在設定されている環境変数を全て表示する

```
% env
```

設定されている環境変数が全て画面に表示される。

fg

バックグラウンドで実行しているジョブをフォアグラウンドに持ってくる

書式

fg [%ジョブ ID]

使い方

- バックグラウンドで実行しているジョブを、フォアグラウンドに持ってくる

```
% kterm &
[1] 326
%
```

『&』を付けて『kterm』を実行することで、バックグラウンドで動作する。『kterm』が現在『ジョブ id 1』と『プロセス id 326』で実行されていることが分かる。バックグラウンドで『kterm』を実行したので、コマンドプロンプト『%』が表示され、この端末から続いて別のコマンドを実行することができる。

```
% jobs
[1] + 実行中です              kterm
```

『jobs』コマンドで確認すると、『ジョブ id 1』で『kterm』がバックグラウンド実行されていることが確認できる。

```
% fg %1
kterm
```

『kterm』の『ジョブ id 1』を引数にして『fg』コマンドを実行し、『kterm』をフォアグラウンドに持ってくる。『kterm』がフォアグラウンドジョブになったので、端末への入出力は全て『kterm』が横取りすることになるため、コマンドプロンプトは表示されない。『kterm』を終了するか、『bg』コマンドで再びバックグラウンドジョブに持っていかない限り、この端末での操作はできない。

この状態で『kterm』を終了するとコマンドプロンプトへ戻る。

```
%
```

関連項目：『bg』73 ページ、『jobs』95 ページ

file

ファイルの種類を判別する

書式

file ファイル名 …

使い方

『file』コマンドは、指定されたファイルなどがどんなタイプのファイルなのかを判別し、結果を表示します。ただし、どのような種類のファイルでも判別できるという訳ではなくて、UNIX(OS)が知っている種類のファイルしか判別できません。

例えば、C 言語のソースファイルや UNIX の実行形式ファイル、JPEG 形式の画像ファイルなどは判別できますが、QuickTime 形式のムービーファイルなどは判別できません。

- 様々なファイルのファイルタイプを判別させてみる

```
% ls -F
Sample.mov       Welcome.html   sample1.jpg   test.c
Screenshot.png   a.out          test          test.f
% file *
Sample.mov:          data
Screenshot.png:      PNG image data, 1024 x 768, 8-bit/color RGB, non-interlaced
Welcom.html:         HTML document text
a.out:               ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for GUN/Linuux
2.2.5, Dynamically linked (users  shared libs)
sample.jpg:          JPEG image data, JFIF standard 1.02, resolution (DPI), 72 x 72
test:                directory
test.c:              ASCII C program text
test.f:              ASCII

%
```

殆どのファイルは正しく判別されているが、判別できないファイル形式を持つ『sample.mov(QuickTime 形式のムービーファイル)』の場合は、単純に『data』とだけ表示される。

find

ファイルを検索する

書式

find　パス　検索条件

使い方

- 現在のディレクトリ以下のディレクトリ階層を表示する

```
% find . -print
.
./01.jpg
./test2
./test2/test2.txt
./test2/test1.txt
./test.c
./a.out
%
```

カレントディレクトリとその下に含まれるすべてのファイル、ディレクトリのパスが表示される。

```
% find /home/od10001/test
/home/od10001/test
/home/od10001/test/01.jpg
/home/od10001/test/test2
/home/od10001/test/test2/test2.txt
/home/od10001/test/test2/test1.txt
/home/od10001/test/test.c
/home/od10001/test/a.out
%
```

このように、パス指定を絶対パスで指定すれば、表示されるパスも絶対パス表記になる。

✓　他に特に条件やコマンドを指定しなければ、この例のようにパスを表示するオプション『-print』は省略することができます。

- 指定したディレクトリ以下に含まれるファイルの中から、ファイル名が『.txt』で終わるファイルを探し、その内容の１行目を表示する

```
% find . -name '*.txt' -print -exec head -1 {} \;
./test/test.txt
This is a test file.
.test/test2/test2.txt
IBM
.test.txt
22765 3264
```

%

条件に該当するファイルのパスが表示され、それぞれのファイルの先頭の１行が表示されている。

- ✓ 『-name』オプションは、その後に検索したいファイル名の条件を付けます。この例ではメタキャラクタである『*』を使って、『test.txt』、『test2.txt』など、ファイル名の末尾が『.txt』で終わるファイルを指定しています。

  『'(シングルクォーテーション)』で囲んでいるのは、こうしないと、『find』コマンドに渡される前にメタキャラクタがシェルによって解釈されてしまい、思い通りの動作にならないためです。

  その後に続く『-print』オプションで、見つかったファイルのパスを表示することを指示します。

  『-exec』オプションは、その後ろに続く UNIX コマンドを実行するために使います。ここではファイルの先頭から指定した行数の内容を表示する『head』コマンドを引数 1(先頭１行を指定)で実行することを指定しています。

  『{}(中括弧)』は『-exec』で指定したコマンドを実行する際に、対象となっているファイルなどのパス名で自動的に置き換えられます。『-exec』オプションを使う場合には、その後に続く文字列のどこまでがコマンドおよびそのコマンドの引数のなかを示すために、『;(セミコロン)』を付けなければなりません。メタキャラクタと同様に、そのままではシェルに解釈されて取り除かれてしまうので、エスケープキャラクタである『¥(エンマーク)』を前につけて『¥;』として正しく『find』コマンドに渡されるようにする必要があります。

- ✓ 『find』コマンドは非常に多機能なコマンドで、様々なオプションを組み合わせると、かなり色々なことができます。『man』コマンドによるオンラインマニュアルや、市販の参考書などを参考にしてください。

関連項目：『UNIX のファイルシステム』41 ページ

ftp

ファイルを別のコンピュータとの間でやり取りする

書式

ftp [ホスト名]

使い方

『ftp』コマンドは、ネットワーク経由で他のコンピュータとファイルをやり取り（転送）するために使用します。次におおまかな『ftp』の作業の流れを記述します。

- リモートコンピュータ『samba』に『ftp』コマンドを使って接続する

```
% cd test
```

『ftp』起動後にもカレントディレクトリを変更することはできるが、まず転送するファイルがある、あるいはファイルを転送して保存をしたいディレクトリに移動しておくのが良い。

```
% ls
01.jpg
```

『01.jpg』という画像ファイルがひとつある。これを『samba00』の『/tmp』ディレクトリに転送する。

```
% ftp samba00
Connected to samba00 (133.26.150.18).
220 Welcome to blah FTP service.
Name (samba00:od01308):
```

ユーザーID を問い合わせてくるので、自分のユーザーID を入力する

```
Name (samba00:od01308): od01308
331 Please specify the password.
Password:
```

パスワードを問い合わせてくるので、自分のパスワードを正しく入力する。
入力したパスワードは表示されないので、入力ミスに注意する。

```
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

この『ftp』という表示が『ftp』コマンドのプロンプトで、『ftp』がコマンドの入力を待っている状態であることを示している。

- 『samba00』の『/tmp』ディレクトリに移動する

```
ftp> cd tmp
250 Directory successfully changed.
```

```
ftp> ls
227 Entering Passive Mode (133,26,150,18,58,246)
150 Here comes the directory listing.
226 Directory send OK.
```

- 『01.jpg』ファイルを『samba00』に転送する

```
ftp> put 01.jpg
local: 01.jpg remote: 01.jpg
227 Entering Passive Mode (133,26,150,18,243,120)
150 Ok to send data.
226 File receive OK.
394392 bytes sent in 0.00602 secs (65502.74 Kbytes/sec)
ftp> ls
227 Entering Passive Mode (133,26,150,18,66,21)
150 Here comes the directory listing.
-rw-r--r--    1 2429     2429       394392 Mar 06 13:04 01.jpg
226   rectory send OK.
.....
```

『01.jpg』が転送されたことが確認できた。

- 『samba00』の『/tmp』ディレクトリにある『sample.c』ファイルを今使っているコンピュータのカレントディレクトリに転送する

```
ftp> get sample.c
local: sample.c remote: sample.c
227 Entering Passive Mode (133,26,150,18,223,135)
150 Opening BINARY mode data connection for sample.c (70141 bytes).
226 File send OK.
70141 bytes received in 0.00898 secs (7813.41 Kbytes/sec)
```

現在使っているコンピュータのカレントディレクトリに、『samba00』からファイル『sample.c』が転送された。

※ 『ftp』は暗号化されない通信のため、コンピュータ間のファイルのやり取りは『scp』を推奨します

関連項目：『scp』114 ページ

grep

ファイルの中から目的の文字パターンを持つ行を抜き出す

書式

grep ［-n］ **文字パターン** ［ファイル名 …］

オプション

-n　　検索して見つかった行を出力する際に、行頭に行番号を付けて出力する。

使い方

- ファイル『test.c』の中で、『printf』という文字パターンを持っている行を抜き出して表示する

```
% grep printf test.c
      printf( “Howdy, world !!¥n*” );
%
```

『test.c』の中で『printf』という文字パターンを持っている行が表示される。

- 複数のファイルの中で、『printf』という文字パターンを持っている行を抜き出して表示する

```
% grep printf test.c test2.c
test.c:       printf("Howdy, world ""¥n");
test2.c:       printf("Success¥n");
%
```

複数のファイルから検索した場合、行頭に抜き出し元のファイル名が付く。

- 複数のファイルの中で、『printf』という文字パターンを持っている行を抜き出し、それぞれの行番号と共に表示する

```
% grep -n printf test.c test2.c
test.c:6:       printf("Howdy, world ""¥n");
test2.c:8:       printf("Success¥n");
%
```

『test.c』ファイルには６行目に、『test2.c』ファイルには８行目に『printf』という文字パターンを持つ行があることが分かる。

✓　文字パターンに空白を含む文字列を指定したい場合には、文字パターンを『'(シングルクォーテーション)』で囲む必要があります。また、文字パターンには単なる文字列だけでなく、正規表現を含むこともできます。正規表現を使うと、より複雑な検索条件を指定できます。

head

ファイルの先頭から指定行数分を表示する

書式

head ［-行数］［ファイル名 ...］

使い方

- ファイルの先頭から 10 行を表示する

```
% head number.txt
1
2
3
4
5
6
7
8
9
10
%
```

行数を指定しない場合には、先頭から 10 行を表示する。

- ファイルの先頭から 7 行を表示する

```
% head -7 number.txt
1
2
3
4
5
6
7
%
```

引数で 7 を指定したので、ファイル『number.txt』の先頭から 7 行が表示された。

関連項目：『tail』116 ページ

hostname

現在使用しているコンピュータのホスト名を表示する

書式

```
hostname
```

使い方

- 現在使用しているコンピュータのホスト名を表示する

```
% hostname
icr1-0001l.wd.isc.meiji.ac.jp
%
```

現在使用しているコンピュータは『icr1-0001l』であることが分かる。

✓ 通常、『hostname』コマンドを単独で使用することはほとんどありません。せいぜい、X サーバーから複数のコンピュータを同時に使用しているような場合に、現在操作している端末がどのコンピュータに繋がっているのかを確認するくらいです。

シェルスクリプトや、ログイン時に実行される環境設定のためのスクリプト(『.cshrc』、『.profile』など)の中で、実行するコンピュータ毎に別々の設定をするために、現在実行しているコンピュータを判別するときなどに使います。

jobs

使っている端末から実行しているジョブの一覧を表示する

書式

```
jobs
```

使い方

- 使っている端末から実行しているジョブの一覧を表示する

```
% jobs
[1]  + 実行中です          emacs
[2]  - 実行中です          kterm
%
```

使っている端末から、『emacs』と『kterm』の２つのプログラムをバックグラウンドで実行していて、それぞれのジョブ id は、１と２であることが分かる。

✓ ジョブ id が分かると、『fg』コマンドを使って特定のバックグラウンドジョブをフォアグラウンドに持ってきたり、『kill』コマンドを使ってプログラムを終了させることができます。

関連項目：『bg』73 ページ、『fg』86 ページ、『kill』96 ページ

kill

指定したプロセスを終了させる

指定したプロセスにシグナルを送る

書式

```
kill [-シグナル] プロセス id
kill [-シグナル] ジョブ id
```

使い方

- 『ps』コマンドで、自分がバックグラウンドで実行している『emacs』のプロセス id を調べて、『kill』コマンドで『emacs』を終了させる

```
% ps
  PID TTY                TIME CMD
 5460 pts/2          00:00:00 csh
 5479 pts/2          00:00:00 emacs
13602 pts/2          00:00:00 ps
%
```

『emacs』のプロセス id は 5479。

```
% kill 5479
%
[1]    終了                emacs
%
```

『emacs』が『kill』コマンドによって終了された。

✓ 『kill』コマンドの本来の目的は、プロセスを終了させることではなく、指定したプロセスに特定の行動を促すシグナルを送ることです。UNIX では、再起動や終了、強制終了をはじめ様々なシグナルが定義されています。このシグナルが『kill』コマンドのプロセスに送られると、プロセスはそのシグナルに従った行動をとります。ただ、一般のユーザーにとってはほとんどのシグナルは使う必要がなく、プログラムを外部から終了させる場合に多く利用されます。引数でシグナルを指定しなかった場合、終了シグナルが送られます。

✓ プログラムを強制終了させたい場合には、『kill -KILL』で『KILL』シグナルを送ります。

- 『jobs』コマンドで、自分がバックグラウンドで実行している『kterm』のジョブ id を調べて、『kill』コマンドで『kterm』を終了させる

```
% jobs
[1]  + 実行中です          emacs
[2]  - 実行中です          kterm
%
```

『kterm』のジョブ id は 2。

```
% kill %2
%
[2]    15 で終了しました kterm
%
```
『kterm』が『kill』コマンドによって終了された。

- ✓ このように『kill』コマンドは、プロセス id またはジョブ id を引数に取ります。これらは同じコマンドを実行しているように見えますが、実際は別々のコマンドです。プロセス id を引数に取る『kill』コマンドは、単独の実行プログラムファイルとして用意されています。
  一方、ジョブ id を引数に取る『kill』コマンドは『csh』、『sh』などのコマンドシェルに組み込まれているシェルの内部コマンドとなっています。ただ、この違いを特に意識する必要はないでしょう。

関連項目：『ps』109 ページ

lp

印刷を実行する

書式

lp [-d プリンタ名] [-hRS] [ファイル名 ...]

オプション

-d　　続けて指定したプリンタ名のプリンタに出力する。
端末の設定によっては省略ができない。

使い方

- プリンタ『p21』からファイル『test.c』を印刷する

```
% lp -dp21 test.c
%
```

指定したプリンタ『p21』から、ファイル『test.c』が印刷される。

✓ プリンタ名を確認したい場合には『lpstat -t』コマンドを実行してください。

関連項目：『lpr』99 ページ

lpr

印刷を実行する

書式

lpr [-d プリンタ名] [-hRS] [ファイル名 ...]

オプション

-d　　続けて指定したプリンタ名のプリンタに出力する。
　　端末の設定によっては省略ができない。

使い方

- プリンタ『p13』からファイル『sample.f』を印刷する

```
% lpr -dp13 sample.f
%
```

　　指定したプリンタ『p13』から、ファイル『sample.f』が印刷される。

プリンタ名を確認したい場合には『lpstat -t』コマンドを実行してください。

関連項目：『lp』98 ページ

ls

ディレクトリの内容を一覧表示する

書式

ls [-adFIR1] [ファイル名またはディレクトリ名]

オプション

-a	隠しファイル(ファイル名の先頭に『.』が付いているファイル)も表示する。
-d	引数にディレクトリを指定した時、ディレクトリの中身ではなくディレクトリ名だけを表示する。
-F	ファイル名とディレクトリ名の末尾に、種類を表す記号を付けて表示する。実行可能ファイルの場合は『*』を、ディレクトリの場合には『/』が末尾に付く。
-l	各ファイル、ディレクトリの詳細な情報(アクセス権、ファイルサイズ、修正時刻、など)を表示する。
-R	指定したディレクトリの中にサブディレクトリが含まれている場合には、そのサブディレクトリの内容も全て表示する。
-1	単独で指定すると、1行に1個ずつファイル名またはディレクトリ名が表示される。

使い方

● カレントディレクトリにあるファイル、ディレクトリの一覧を表示する

```
% ls
  01.jpg   a.out*   test.c   test.txt   test2/
%
```

カレントディレクトリにあるファイルの一覧が表示される。

✓ このように一覧を表示するファイル名またはディレクトリ名を省略すると、カレントディレクトリを指定したことになります。また、先ほどのような結果になるためには、本来なら『-F』オプションを指定する必要があります。これは、生田システムでは『ls -F』を『ls』コマンドとして定義しているからです。

● カレントディレクトリにあるサブディレクトリ『test』の詳細な情報を表示する

```
% ls -dl test
drwxr-sr-x   3 od10001  assist    512  Mar 19 14:10 test/
%
```

サブディレクトリ『test』の詳細な情報が表示される。

関連情報：『UNIX のファイルシステム』41 ページ

man

マニュアルを表示する

書式

man [-s セクション] コマンド名

オプション

-s　　指定したセクションの中から、コマンドのマニュアルを検索する。

使い方

『man』コマンドは、システム側で用意しているオンラインマニュアルページを表示します。コマンドの使い方が分からなかったり、どのようなオプションがあるのか調べたい場合に使います。

- ls コマンドのマニュアルページを表示する

```
% man ls
LS(1)                                                    LS(1)

名前
     ls, dir, vdir - ディレクトリの中身をリスト表示する

書式
     ls [オプション] [file ...]

     POSIX オプション [-CFRacdilqrtul].........

     『ls』コマンドのマニュアルページが表示される。

.........
― 続ける ―
```

1 画面で表示しきれない場合には、1 画面分を表示したところで『-- 続ける --』を表示してストップする。次のページを読みたい場合には、スペースキーを押す。一行ずつスクロールしたい場合には、『Enter』キーを押す。途中で終了したい場合には、『q』キーを押す。

✓　表示されたメニュアルページをよく見ると、『LS(1)』のように指定したコマンド名の後ろに括弧で囲まれた数字が付いています。これは、そのコマンドが所属しているマニュアルセクションを表すセクション番号です。セクション１は、ユーザーコマンドが含まれるセクションです。

通常はセクション番号を指定せずに、単に『man コマンド名』、とすればいいのですが、稀に同じ名前のものが複数のセクションに入っていることがあります。例えば、『exit』という名前に対するマニュアルページは、セクション１の『exit』コマンドに対してのものの他に、セクション 3C(C 言語などの関数についてのマニュアル)セクションにも、『exit』関数についての

ものがあります。『man』コマンドは、順番に検索していって最初に見つかったマニュアルページを表示してしまうので『exit』関数について調べたくても、単に『man exit』としただけでは、『exit』コマンドを先に見つけてしまいます。そこで、このような時には、『man -S 3c exit』のように明示的にセクション番号を指定することで、求める『exit』関数について調べることができます。

mkdir

ディレクトリを作成する

書式

mkdir ［-p］ ディレクトリ名 ...

オプション

-p　　指定したディレクトリの親ディレクトリが存在しなかった場合、その親ディレクトリも一緒に作成する。

使い方

● カレントディレクトリに『kadai』ディレクトリを作成する

```
% ls
test1.txt      test2.txt
```

現時点では『kadai』ディレクトリは存在していない。

```
% mkdir kadai
```

『kadai』ディレクトリを作成。

```
% ls
kadai/     test1.txt     test2.txt
%
```

『kadai』ディレクトリが作成できた。

✓ ディレクトリ名の指定には、このような相対パスによる指定だけではなく、絶対パスによる指定もできます。

● カレントディレクトリの下の『report』ディレクトリに『part1』ディレクトリを作成する

```
% ls
test1.txt      test2.txt
```

現時点では『part1』ディレクトリだけでなく、『report』ディレクトリも存在していない。

```
% mkdir report/part1
mkdir: ディレクトリ’report/part1’を作成できません: そのようなファイルやディレクトリはありません。
%
```

このように、存在しないディレクトリ『report』の下にいきなりディレクトリを作成しようとしてもエラーになってしまう。このような場合には『-p』オプションを使うことで、一度に複数階層のディレクトリの作成を行うことができる。

```
% mkdir -p report/part1
% ls
report/     test1.txt     test2.txt
```

存在していなかった『report』ディレクトリが作成される。

```
% cd report
```

『report』ディレクトリに移動。

```
% ls
part1/
```

『report』ディレクトリの下に『part1』ディレクトリが作成されている。

- ✓ 『mkdir』コマンドでディレクトリを作成できるのは、基本的に自分のホームディレクトリの中だけです。システムのディレクトリや、他のユーザーのホームディレクトリなどに対しては書き込み権限を持っていないので、ディレクトリを作成することはできません。

『rmdir』113 ページ、『UNIX のファイルシステム』41 ページ

more

テキストファイルの内容を 1 画面分ずつ表示する

書式

more ［ファイル名 ...］

使い方

● ファイル『sample.c』の内容を 1 画面分ずつ表示する

```
% more sample.c
#include <stdio.h>

int main(void)
{
        int    a;
        double b;
..........
```

『sample.c』の内容が表示される。1 画面で表示しきれない場合には、1 画面分表示したところで、画面の最下段に次のようなメッセージを表示してストップする。

```
― 継続 ―(32%)
```

次の画面を表示させるためには、スペースキーを押す。一行ずつスクロールをしたい場合には、『Enter』キーを押す。途中で終了したい場合には『q』キーまたは『Ctrl+c』キーを押す。

● 別のコマンドの実行結果を入力元とし、1 画面分ずつ表示する

```
% ls -alR | more
.:
合計 10
drwxr-sr-x 3 od10001 assist  512 Mar 19 19:39 ./
drwxr-sr-x 9 od10001 assist 1024 Mar 19 17:17 ../
drwxr-sr-x 3 od10001 assist  512 Mar 19 19:39 report/
```

『ls -alR』コマンドの実行結果が 1 画面分ずつ表示される。

mv

ファイルを別の場所に移動する

ディレクトリを別の場所に移動する

ファイル名を変更する

ディレクトリ名を変更する

書式

```
mv [-i]  移動させるファイルまたはディレクトリ  移動先
mv [-i]  ファイルまたはディレクトリ  新しいファイル名またはディレクトリ名
```

オプション

-i　　移動先あるいは新しくつけようとした名前と同じ名前のファイルやディレクトリが既にある場合、確認のメッセージを表示する。

使い方

- カレントディレクトリにあるファイル『sample.c』を、ディレクトリ『test』の下に移動する

```
% ls
sample.c     test/
% mv sample.c test/
```

『sample.c』ファイルをサブディレクトリ『test』の下に移動。

```
% ls
test/
```

『sample.c』ファイルが無くなっている。

```
% ls test/
sample.c
%
```

サブディレクトリ『test』の下に『sample.c』が移動している。

✓　この例では、ディレクトリを表すのに『test/』のように末尾に『/(スラッシュ)』を付けていますが、これは付けなくても構いません。

- ファイル『sample.c』のファイル名を『report.c』に変更する

```
% ls
sample.c
```

ファイル『sample.c』がある。

```
% mv sample.c report.c
% ls
report.c
```

%

『sample.c』のファイル名が『report.c』に変更されている。

✓ 『mv』コマンドの本来の目的はファイルやディレクトリを移動することですが、このように使うことでファイル名やディレクトリ名の変更をすることができます。

関連項目：『UNIX のファイルシステム』41 ページ

nkf

ファイルの漢字コードを変換する

書式

nkf [-ejsmMv] [入力ファイル名 ...]

オプション

-e	EUC 漢字コードに変換する。
-j	JIS 漢字コードに変換する。(デフォルト)
-s	シフト JIS 漢字コードに変換する。
-w	UTF8 コードに変換する。
-m	MIME エンコードされた文字列をデコードする。
-M	文字列を MIME エンコードする。
-v	nkf のバージョンと、ヘルプを表示する。

使い方

- シフト JIS 漢字コードで書かれているファイル『test.sj』を EUC 漢字コードに変換してファイル『test.euc』に書き出す

```
% nkf -e test.sj > test.euc
%
```

ファイル『test.sj』の内容が EUC 漢字コードに変換され、シェルのリダイレクト機能により『test.euc』ファイルに書き出された。

- どの漢字コードで書かれているのか分からないファイル『test.nazo』をシフト JIS 漢字コードに変換し、ファイル『test.sj』に書き出す

```
% nkf -s test.nazo > test.sj
%
```

✓ 『nkf』コマンドはほとんどの場合、自動的に入力ファイルに使われている漢字コードを判別してくれます。そのため、先ほどの２つの例のように、特に入力ファイルの漢字コードを指定する必要がありません。

ps

プロセスの実行状況の一覧を表示する

書式

```
ps [-efl]
```

オプション

-e	自分がその端末から実行したプロセスだけでなく、そのコンピュータで実行されている全てのプロセスの一覧を表示する。
-f	絶対パス指定で実行されたプロセスのパス名など完全な形式の一覧を表示する。
-l	より詳細な形式で一覧を表示する。

使い方

- 自分がその端末から実行しているプロセスの一覧を表示する

```
% ps
   PID TTY          TIME CMD
  8624 pts/3    00:00:00 csh
 20836 pts/3    00:00:00 xclock
  9208 pts/3    00:00:11 emacs
%
```

現在、『csh』、『xclock』、『emacs』の３つのプロセスを実行中であることが分かる。

✓ 『-f』オプションと、『-l』オプションでは、表示される情報に重複している部分もありますが、どちらか片方のオプションでしか表示されない情報もあります。『-fl』のように、ふたつのオプションを同時に指定すれば、より詳細な情報を表示することができます。

pwd

カレントディレクトリ(現在のディレクトリ)を表示する

書式

```
pwd
```

使い方

- 現在のディレクトリ(カレントディレクトリ)を表示する

```
% pwd
/home/od10001
```

カレントディレクトリは『/home/od10001』

```
% cd test/
```

サブディレクトリ『test』に移動。

```
% pwd
/home/od10001/test
```

カレントディレクトリが『/home/od10001/test』に変わっている。

```
%
```

✓ 『pwd』コマンドによるディレクトリ表示は絶対パスの形式になります。また、カレントディレクトリは、作業ディレクトリなどとも呼ばれます。

関連項目：『cd』75 ページ、『UNIX のファイルシステム』41 ページ

rm

ファイルを削除する

指定したディレクトリ以下のファイルとディレクトリを削除する

書式

rm [-ifr] ファイルまたはディレクトリ名 ...

オプション

-i	削除前に確認を求めるプロンプトを出す。
-f	書き込み保護のあるファイルも確認プロンプトを出さずに強制的に削除する。
-r	ディレクトリおよびサブディレクトリを削除する。

使い方

- ファイルを削除する

```
% ls
a.out*     test/     test.c
%
```

カレントディレクトリの内容を表示。

```
% rm test.c
%
```

『test.c』を削除する。

```
% ls
a.out*     test/
%
```

『test.c』が削除されているのが確認できる。

- 削除前に確認付きでファイルを削除する

```
% ls
a.out*     test/
%
```

カレントディレクトリの内容を表示。

```
% rm -i a.out
rm: remove 通常ファイル 'a.out'?
```

確認を求めてくるので、良ければ『y』と答える。

```
rm: remove 通常ファイル 'a.out'? y
%
% ls
test/
%
```

『a.out』が削除されているのが確認できる。

- 指定したディレクトリとその中身を削除する

```
% ls
a.out*     test/     test.c
%
```

カレントディレクトリの内容を表示。

```
% ls test/
kanji.txt     sample.txt
%
```

『test』ディレクトリに2つのファイルが存在している。

```
% rm -r test
```

『-r』オプション付で『rm』コマンドを実行。

```
% ls
a.out*     test.c
%
```

『test』ディレクトリと、その中の2つのファイルが一度に削除されている。

関連項目:『rmdir』113ページ、『UNIXのファイルシステム』41ページ

rmdir

ディレクトリを削除する

書式

rmdir　ディレクトリ名

使い方

● 中身が空のディレクトリを削除する

```
% ls
a.out*     test/     test.c
```

カレントディレクトリの内容を表示。

```
% ls test/
```

『test』ディレクトリの中身は空。

```
% rmdir test
```

『test』ディレクトリを削除する。

```
% ls
a.out*     test.c
%
```

『test』ディレクトリが削除されていることが確認できる。

● 中身が空でないディレクトリの削除を試みる

```
% ls
a.out*     sample/
```

カレントディレクトリの内容を表示。

```
% ls sample/
test.c
```

『sample』ディレクトリの中にはファイルが入っていて空ではない。

```
% rmdir sample
rmdir: 'sample': ディレクトリは空ではありません。
```

ディレクトリが空ではないためエラーになってしまった。

```
% ls
a.out*     sample/
%
```

『sample』ディレクトリの削除に失敗した。

関連項目：『rm』111 ページ、『UNIX のファイルシステム』41 ページ

scp

セキュアにコンピュータ間でファイルのやり取りを行う

書式

scp［オプション］［ログイン名@ホスト名］:転送元ファイル名［ログイン名@ホスト名］:転送先パス

オプション

-C	通信を圧縮する
-P	ポート番号を指定する
-p	コピー元の更新時間とモードを維持する
-r	ディレクトリ内を再帰的にコピーする
-1	SSH のプロトコルバージョン 1 を使用する
-2	SSH のプロトコルバージョン 2 を使用する
-4	IPv4 を使用する
-6	IPv6 を使用する

使い方

● リモートコンピュータ『samba00』にローカルコンピュータの『test.txt』を転送する

```
% scp test.txt od10001@samba00:/home/od10001/test
test.txt                                      100%    0      0.0KB/s   00:00
```

● リモートコンピュータ『samba00』と『isc--iasrv』間で『test.txt』を転送する

```
% scp od10001@samba00:/tmp/test.txt od10001@samba00:/home/od10001/test
od10001@isc-iasrv's password:
```

パスワードを求められる場合はパスワードを入力します

```
test.txt                                      100%    0      0.0KB/s   00:00
```

関連項目：『ftp』90 ページ

ssh

セキュアにほかのコンピュータにログインする

書式

ssh ホスト名

オプション

-p	ポート番号を指定する
-l	ユーザー名を指定する
-i	公開鍵ファイルを指定する
-C	通信を圧縮する
-1	SSH のプロトコルバージョン 1 を使用します。
-2	SSH のプロトコルバージョン 2 を使用します。
-4	IPv4 を使用します。
-6	IPv6 を使用します。

使い方

- リモートコンピュータ『samba00』にログインする

```
% ssh samba00
od10001@isc-iasrv's password:
```

パスワードを求められる場合はパスワードを入力します

```
Last login: Fri Mar 22 09:44:44 2019 from vpc-0071.isc.meiji.ac.jp
samba00%
```

これでリモートコンピュータ『samba00』へのログインが完了。コマンドプロンプトが表示されているので、以降は通常通りの操作で使用することができる。

関連項目：『telnet』117 ページ

tail

ファイルの末尾から指定行数分を表示する

書式

tail [-行数] [ファイル名 ...]

使い方

- ファイルの末尾から10行を表示する

```
% tail number.txt
11
12
13
14
15
16
17
18
19
20 ここが最終行です。
%
```

行数を指定しない場合には、末尾の10行を表示する。

- ファイルの末尾から4行を表示する

```
% tail -4 number.txt
17
18
19
20 ここが最終行です。
%
```

引数で4を指定したので、ファイル『number.txt』の末尾の4行が表示された。

関連項目：『head』93ページ

telnet

ほかのコンピュータにログインする

※ 『telnet』は暗号化されない通信のため、通常、リモートログインは『ssh』を使用します。
以下の samba00 に接続する例も現在は利用できません。参考情報としてください。

書式

telnet　**ホスト名**

使い方

● 現在使っている端末から、リモートコンピュータ『samba00』にログインする

```
% telnet samba00
Trying 133.26.150.23...
Connected to samba00.wd.isc.meiji.ac.jp (133.26.150.23).
Escape character is '^]'.
This system is a restricted access system. All activity on this system is subject to monitoring.
If information collected reveals possible criminal activity or activity that exceeds
privileges, evidence of such activity may be provided to the relevant authorities for further
action. By continuing past this point, you expressly consent to this monitoring.

login:
```

ユーザーid(ログイン名)を聞いてくるので入力する。

```
login: od10001
Password:
Last login: Wed Apr 27 10:50:36 from iedu-c3.isc.mei
Sun Microsystems Inc.   SunOS 5.10      Generic January 2005

samba00%
```

これでリモートコンピュータ『samba00』へのログインが完了。コマンドプロンプトが表示されているので、以降は通常通りの操作で使用することができる。

関連項目：『scp』114 ページ

これでリモートコンピュータ『samba00』へのログインが完了。コマンドプロンプトが表示されているので、以降は通常通りの操作で使用することができる。

time

指定したコマンドを実行し、実行にかかった時間を表示する

書式

time　**コマンド名　[指定したコマンドの引数 ...]**

使い方

『time』コマンドは、引数で指定したコマンドを実行し、そのコマンドの実行にかかった時間を表示します。

- ファイル『a.out』を実行し、実行にかかった時間を計測する

```
% time ./a.out
Program start
:
:
:
:
Program end
0.660u 5.960s 0:10.92 60.6%      0+0k 0+0io 19978pf+0w
%
```

ここで『Program start』から『Program end』の行はプログラム『a.out』自身が出力したメッセージで、『time』コマンドの出力はその次の行になる。先頭にある『u』がついている部分が、ユーザープロセスで実行にかかった時間を表していて、ここでは0.660秒のCPU時間を消費したことが分かる。

unalias

設定してあったコマンド名のエイリアス(別名)を削除する

書式

unalias　**エイリアス名**

使い方

- 定義されているエイリアス『lsl』を削除する

```
% alias
lsl       (ls -l)
df        (df -k)
%
```

『lsl』を実行すると、実際には『-l』オプション付きで『ls』コマンドが実行されるようにエイリアスが定義されている。

```
% lsl
合計 18
-rwxr-xr-x  1 od10001 assist   4888 Mar 20 14:07 a.out
-rw-r--r--  1 od10001 assist     83 Mar 16 09:42 sample.c
drwxr-sr-x  2 od10001 assist    512 Mar 19 20:08 report
%
```

実際に、『lsl』を実行すると、『ls -l』が実行されている。

```
% unalias lsl
```

エイリアス『lsl』を削除する。(定義を消す)

```
% lsl
lsl: コマンドが見つかりません。
%
```

エイリアス『lsl』が削除されてしまったので、実行できずにエラーになった。

```
% alias
df        (df -k)
%
```

『alias』コマンドで確認をしてみると、『lsl』は消えている。

- ✓ 『unalias』コマンドを、引数を何もつけずに実行してしまうと、全てのエイリアスが削除されてしまうので注意をしてください。

関連項目：『alias』72 ページ

uniq

ファイルの中の重複行の削除

ファイルの中の重複行の表示

書式

uniq ［-d］ 入力ファイル名 ［出力ファイル名］

オプション

-d　　重複している行だけ出力する。それぞれの重複部分については 1 行に削って表示される。

使い方

- ファイル『sample.txt』の中から重複している部分を１行に削って表示する

```
% cat sample.txt
竹やぶに竹たてかけた。
隣の客はよく柿食う客だ。
隣の客はよく柿食う客だ。
青まきがみ赤まきがみ黄まきがみ。
%
```

ファイル『sample.txt』の内容はこのようになっている。

```
% uniq sample.txt
竹やぶに竹たてかけた。
隣の客はよく柿食う客だ。
青まきがみ赤まきがみ黄まきがみ。
%
```

重複していた『隣の客はよく柿食う客だ。』の部分が１行に削られて表示された。

- ファイル『sample.txt』の中から重複している部分を抜き出して表示する

```
% cat sample.txt
竹やぶに竹たてかけた。
隣の客はよく柿食う客だ。
隣の客はよく柿食う客だ。
青まきがみ赤まきがみ黄まきがみ。
%
```

ファイル『sample.txt』の内容はこのようになっている。

```
% uniq -d sample.txt
隣の客はよく柿食う客だ。
%
```

重複していた『隣の客はよく柿食う客だ。』の部分だけが表示された。

wc

ファイル行数、単語数、文字数を表示する

書式

wc [-cmClw] [ファイル名 ...]

オプション

-c	バイト数を数えて表示する
-m	文字数を数えて表示する
-C	『-m』と同じ
-l	行数を数えて表示する
-w	単語数を数えて表示する

使い方

- ファイル『sample.c』の行数、単語数、文字数を表示する

```
% wc sample.c
       13       50      348 sample.c
%
```

『wc』コマンドの表示は、左から『行数』、『単語数』、『文字数』となっている。

✓ オプションを何も指定しないで『wc』コマンドを実行すると、『-clw』オプション(バイト数、行数、単語数を数えて表示)を指定した時と同じ動作になります。

- ファイル『sample.c』と『report.txt』の行数を数えて表示する

```
% wc sample.c report.txt
       13       50      348 sample.c
        4        5       70 report.txt
       17       55      418 total
%
```

複数のファイルを指定すると、それぞれのファイルについての情報のほかに、それらを合計した値も最後に出力されます。

- ファイル『kanji.txt』の行数、単語数、文字数を表示する

```
% cat kanji.txt
明治大学の生田キャンパス
中央校舎の５階
%
```

ファイル『kanji.txt』の内容はこのようになっている。

```
% wc -mlw kanji.txt
        2        2       40 kanji.txt
%
```

ファイル『kanji.txt』は、行数が 2、単語数も 2、文字数は 34 であると表示された。文字数については、日本語として見ると『明治大学の生田キャンパス』が 12 文字、『中央校舎の５階』が 7 文字、そして画面には表示されていないが、各行の終わりに改行コードが 1 文字(1 バイト)分付いているので 2 文字、合計 21 文字となる筈である。しかし、デフォルトでも使われるオプション『-c』はバイト数によるカウントのため、表示のような結果となる（日本語文字 1 文字は 2 バイト）。

```
% wc -mlw kanji.txt
       2       2      21 kanji.txt
%
```

このように、『-m』オプションを指定すれば、日本語の場合にも正しく文字数を表示することができる。

✓ 残念ながら『wc』コマンドで数えることのできる『単語』とは、英語のように単語間がスペースで区切られているものだけなので、そのような形態になっていない日本語には対応できません。そのため『明治大学の生田キャンパス』のような言葉も、ひとつの単語として数えられてしまっています。

which

そのコマンド名で実行されるコマンドファイルの場所を表示する

書式

which [コマンド名 ...]

使い方

- どのディレクトリの『pwd』コマンドが実行されるのかを表示する

```
% pwd
/home/od10001
```

『pwd』コマンドはカレントディレクトリを表示するコマンド。

```
% which pwd
/usr/bin/pwd
```

『pwd』とだけ入力して実行した時には、『/usr/bin』ディレクトリにある『pwd』を実行していることが分かる。

```
% /usr/bin/pwd
/home/od10001
%
```

このように絶対パスで『pwd』コマンドを実行しても、同じ結果を得ることができる。

- 『ls』と入力した際に何が実行されているのかを調べる

```
% ls
01.jpg   a.out*   test/   test2.c
```

『ls』コマンドは、ディレクトリ、ファイルの一覧を表示するコマンド。

```
% which ls
ls:          aliased to ls -CF
```

『ls』コマンドは、実際はエイリアスが設定されていて、『ls -CF』が実行されていたことが分かる。

```
% /usr/bin/ls
01.jpg   a.out*   test    test2.c
%
```

『ls』コマンドの本体は『/usr/bin/ls』に存在しており、これを絶対パスで直接実行すると『-CF』オプションが指定されないので、表示結果がエイリアスを実行した時と異なっている。

- ✓ エイリアスについては C シェルの初期設定ファイル『.cshrc』の中で定義されたものしか『which』コマンドでは表示できません。プロンプトから『alias』コマンドで定義したエイリアスなどについては『which』コマンドの表示結果には反映されないので注意が必要です。また K シェルや bash など他のコマンドシェルはそれぞれ別の初期設定ファイルを使うので、『which』コマンドでのエイリアス検索は利用できません。

関連項目：『alias』72 ページ、『unalias』119 ページ

who

現在そのコンピュータにログインしているユーザーを表示する

書式

```
who
```

使い方

- 現在そのコンピュータにログインしているユーザーを表示する

```
% who
od01002  pts/5    Mar 20 18:14    (samba01)
od01003  dtremote Mar 20 12:36    (samba04)
od10001  pts/7    Mar 20 20:13    (dilemma)
od10001  pts/3    Mar 20 19:10    (dilemma)
%
```

現在ログインしているユーザーは、『od01002』、『od10003』、『od10001』の３人であることが分かる。

✓ よく見ると、右端に『samba01』、『samba04』など、括弧『()』で囲まれた文字列は、それぞれのユーザーがどのコンピュータからログインしてきたかを示しています。また、『od10001』が２人ログインしているように見えるのは、一人のユーザーが２つの端末からログインしている場合に、それぞれ別のユーザーとして表示されるためです。

Memo

Memo

索引

い

か

け

こ

し

せ

そ

た

つ

て

は

ひ

ふ

ほ

ま

UNIX 利用の手引き

発行日　　2024 年 4 月 1 日

発行　　生田メディア支援事務室

〒214-8571　神奈川県川崎市多摩区東三田 1-1-1

Tel.) 044-934-7710　　Fax.) 044-934-7904