

初心者のためのシェル(bash)入門

GUIの世界から端末エミュレータやコンソールのCUIの世界へようこそ。よくいらっしやいました。

このページではGNU/Linuxシステムにほぼ標準採用されているシェル [Bash \(Bourne Again Shell\)](#) の操作についてとりとめなく書いています。

X Window System を使っている場合は [端末エミュレータ](#) のウィンドウを開くことでシェルを利用することができます。

- [初心者のためのシェル\(bash\)入門](#)
 - [シェルって何？](#)
 - [PATH って何ですか？](#)
 - [一時的に環境変数を設定してコマンドを実行したい](#)
 - [何か操作が違う。ひょっとしてbashじゃないのかもしれない・・・？](#)
 - [/.bashrc, ~/.bash_profile, /etc/profile, /etc/bashrc ってどう違うのよ？](#)
 - [コマンド履歴の数を変更したい。](#)
- [コマンド補完機能を使おう。](#)
 - [コマンド補完って何？](#)
 - [使用例 \(cd /usr/src/linux\)](#)
 - [関連リンク](#)
- [ジョブ制御](#)
 - [コマンドを実行する](#)
 - [フォアグラウンドで実行](#)
 - [バックグラウンドで実行 \(&\)](#)
 - [実行中のコマンドについていろいろ](#)
 - [フォアグラウンドで動作中のコマンドの一時停止 \(Ctrl+z\)](#)
 - [一時停止中のコマンドをバックグラウンドで動かす \(bg\)](#)
 - [一時停止中/バックグラウンドで動作中のコマンドをフォアグラウンドで動かす。\(fg\)](#)
 - [一時停止中/バックグラウンドで動作中のコマンドの様子を見る \(jobs\)](#)
 - [今動いているプロセスの一覧を見る。\(ps\)](#)
 - [実行中のコマンドを強制終了する](#)
 - [フォアグラウンドで実行中 \(Ctrl+c\)](#)
 - [バックグラウンドで実行中 \(kill\)](#)
 - [関連リンク](#)
- [FAQ/Tips](#)
 - [実行したソフトウェアをログインシェルの子プロセスにたくない](#)
 - [全ファイル、ディレクトリのサイズが大きいもの上位20を表示させる \(パイプ\)](#)
 - [2ch 関連スレッド](#)

シェルって何？

シェル（「殻」という意味）はコマンドラインからCUIプログラムを実行するソフト。

ライン入力を読み取って、適切なプログラムを適切な呼び出しで実行する。

コンソール端末でも、ターミナルエミュレータでも、CUIプログラムを起動するときにはまずこいつが起動して、そこに入力を打ち込むことで間接的にCUIプログラムが起動する、という形になる。

入力したい内容をファイルにまとめて書けば一気に実行もできる。

BashはUNIX標準シェルのshと互換のフリーなシェル。Linuxで標準的に使われている。

- <http://ja.wikipedia.org/wiki/%E3%82%B7%E3%82%A7%E3%83%AB>

PATH って何ですか？

よく CUI の話題に出てくる環境変数 PATH については以下の文献を参照のこと。

- [PATH HOWTO](#)

特定のディレクトリ(下記の例では /usr/local/hoge と /usr/local/fuga)を PATH に加えたい場合は

```
export PATH=$PATH:/usr/local/hoge:/usr/local/fuga
```

みたいな感じでユーザの ~/.bashrc や ~/.bash_profile などのシェルの環境設定ファイルに追加しておく。

PATH にディレクトリを新たに追加する事を俗に「PATH を通す」と言ったりします。

一時的に環境変数を設定してコマンドを実行したい

export コマンドを使わなくても、以下のように一時的に環境変数を設定することができます。

```
$ LANG=C man cp
```

これで環境変数 LANG が一時的に C に設定されます。(結果、cp に関する英語の man ページが表示されます)

何か操作が違う。ひょっとして bash じゃないのかもしれない・・・？

今自分が使っているシェルの種類を確認するには、環境変数 SHELL を表示することで確認できます。

```
echo $SHELL
```

~/.bashrc, ~/.bash_profile, /etc/profile, /etc/bashrc ってどう違うのよ？

全部 bash で読み込まれる初期化設定ファイル。けど、読み込まれるタイミングがちょっと違う。

ホームディレクトリにある .bashrc と .bash_profile はユーザーの個人用の設定で、/etc にある2つ

はシステムの標準設定。/etc/profile と ~/.bash_profile はログイン時のシェル(ログインシェル)で1回だけ読み込まれる設定で、/etc/bashrc と ~/.bashrc はシェルの起動時に毎回読み込まれる設定。

普通は .bashrc か .bash_profile のどちらかに書いておけば大丈夫。 [PATH HOWTO](#) とかで起動時の流れがちょっとだけ解説してあるよ。

コマンド履歴の数を変更したい。

コマンド入力欄で `Ctrl+R` キーを押した時に出る、入力したコマンド履歴の数を変更したいという時は、ユーザの ~/.bashrc や ~/.bash_profile ファイルに

```
export HISTFILESIZE=100
export HISTSIZE=100
```

と追加しましょう。

また

```
export HISTCONTROL=ignoreboth
```

を加えると同じコマンドを重複して記録しないようになります。

コマンド補完機能を使おう。

コマンド補完って何?

コマンドを入力する時 Linux などの Bash, Tcsh, Zsh などのシェルにあって、Windows の cmd.exe にはない [\(*1\)](#) [\(*2\)](#) ものにコマンド補完というのがある。

これはディレクトリ名やファイル名を途中まで打ち込んだところで `Tab` キーを押すと、シェルが途中まで入力した文字と合致するものを自動的に補完することができる便利な機能です。

候補が多数ある場合は合致した候補の一覧を表示することができるので、今度は自分が1、2文字程度補完してやってまた `Tab` キーを押すと残り全部を補完してくれます。

つまり、自分でコマンド全部を入力しなくても良いと言う事。タイプ数と入力間違いを大幅に減らすことができます。

```
vi /usr/src/linux/Documentation/sound/README.OSS
```

こんなのいちいち入力して作業をしていたら日が暮れます。補完を使いましょう。

使用例 (cd /usr/src/linux)

以下はコマンド補完を使って /usr/src/linux にディレクトリ移動する場合の例です。

```
$ cd /u
```

ここまで入力したて `Tab` キーを押すと・・・

```
$ cd /usr/
```

sr/ が補完されます。

```
$ cd /usr/s
```

ここでまた Tab キーを押すと・・・

```
  sbin  share  src
```

このように **名前が s から始まる候補が sbin, share, src の三つある**とシェルが教えてくれます。

```
$ cd /usr/sr
```

目的の名前は src なので r を入力してまた Tab キーを押すと・・・

```
$ cd /usr/src/
```

rc/ が補完されます。

```
$ cd /usr/src/l
```

l を押して Tab キーを押すと・・・

```
$ cd /usr/src/linux/
```

/usr/src/linux/ と目的のディレクトリが表示されたので Enter キーを押せばコマンド実行できます。

関連リンク

- [bash\(1\) - プログラム補完](#)

ジョブ制御

Windows のヘビーユーザや MS-DOS 時代からパソコンを使っていた人には CUI の画面はお馴染みだと思います。

MS-DOS などでは一つのコマンドを実行するとそのコマンドが終了するまでは他の作業ができないシングルタスクでしたが、UNIX 互換 OS (Linux 含む) は CUI の時代からマルチタスクを実現していたので CUI 環境でもマルチタスクを利用できます。

コマンドを実行する

フォアグラウンドで実行

単にコマンドを入力します。

```
cp /dokokano/dekai/file /home/hoge
```

cp コマンドはファイルをコピーするコマンドです。

上記のように普通にコマンドを実行することを **フォアグラウンド** で実行するといいます。

フォアグラウンドでコマンドを実行した場合は Windows のコマンドプロンプトや MS-DOS プロンプトと同じように、**そのコマンドが終了するまでは他の作業をする事はできません**。

複数の作業を同時に行いたい場合は、[バックグラウンドでコマンドを実行](#)します。

バックグラウンドで実行 (&)

コマンド実行時、& を付けると **バックグラウンド** で実行されます。

```
cp /dokokano/dekai/file /home/hoge &
```

こうすることで、例えば巨大なファイルをコピーしている間、別の作業が可能になります。

実行中のコマンドについていろいろ

フォアグラウンドで動作中のコマンドの一時停止 (Ctrl+z)

& を付けずに **フォアグラウンド** でコマンドを実行してしまった場合、Ctrl + z を押すと一時停止状態にできます。

```
$ cp /dokokano/dekai/file /home/hoge
[1]+  Stopped                  cp /dokokano/dekai/file /home/hoge
```

一時停止されて待ち状態になります。

一時停止中のコマンドをバックグラウンドで動かす (bg)

bg コマンドを使います。[jobs コマンド](#)で表示される番号で制御するコマンドを指定できます。

```
$ jobs
[1]+  Stopped                  cp /dokokano/dekai/file /home/hoge
$ bg 1
[1]+  cp /dokokano/dekai/file /home/hoge
```

一時停止中/バックグラウンドで動作中のコマンドをフォアグラウンドで動かす (fg)

fg コマンドを使います。使い方は [bg コマンド](#) とほぼ同じです。

```
$ fg 1
cp /dokokano/dekai/file /home/hoge
```

一時停止中/バックグラウンドで動作中のコマンドの様子を見る (jobs)

[バックグラウンドで実行したコマンド](#)や[一時停止したコマンド](#)は jobs コマンドを使うと見ることが出来ます。

```
$ jobs
[1]+  Stopped                  cp /dokokano/dekai/file /home/hoge
```

今動いているプロセスの一覧を見る, (ps)

ps コマンドを使います。

```
$ ps
PID TTY          TIME CMD
 404 pts/1        00:00:00 bash
 460 pts/1        00:00:00 ps
```

単純に ps コマンドを実行すると今使っているシェルで起動したプログラムが表示されます。上記の例ではシェル自身(bash)とpsコマンド自身(ps)が表示されています。

PID という数字は**プロセス ID**といい、重要な役目があるので覚えておきましょう。

実行中のコマンドを強制終了する

フォアグラウンドで実行中 (Ctrl+c)

& をつけずに**フォアグラウンド**で実行しているコマンドを停止する場合は Ctrl + c を押すことで強制終了できます。

```
$ cp /dokokano/dekai/file /home/hoge
(ここで Ctrl + c を押すと強制終了されてシェルに戻ります)
$
```

バックグラウンドで実行中 (kill)

& をつけて**バックグラウンド**で実行しているコマンドの場合、Ctrl + c を受けつけてくれないので kill コマンドを使って強制終了します。

```
$ cp /dokokano/dekai/file /home/hoge &
$ ps
PID TTY          TIME CMD
 404 pts/1        00:00:00 bash
 460 pts/1        00:00:00 ps
 480 pts/1        00:00:00 cp
$ kill 480
```

バックグラウンドで実行したコマンドの **プロセス ID** を kill コマンドを使う際に指定することで動いているコマンドを強制終了(殺す)ことができます。例のようにコピーするコマンドなどの場合に強制終了した場合、当然ながらコピーは完全に行われないので注意。

関連リンク

- [bash\(1\) - ジョブ制御](#)
-

FAQ/Tips

実行したソフトウェアをログインシェルの子プロセスにたくない

[nohup](#)や[disown](#)の組み込みコマンドを使うことでログアウトに影響されないようにできます。

全ファイル、ディレクトリのサイズが大きいもの上位20を表示させる（パイプ）

```
du -a / | sort -rn | head -n 20
```

最初の du コマンドでルートからの全ファイル、ディレクトリのディスク使用量を表示しています。ただ、それだけだと順番に表示されないなのでソートさせたい。

せっかく sort というソート用のコマンドがあるんだからこれでなんとかしたい。

そこで使うのがパイプ。du と sort のあいだにある「|」です。これは名前の通り、パイプのようにコマンドの出力結果を次のコマンドにわたす記号です。というわけで、コマンドとコマンドのあいだに「|」を置いて、結果を渡しているわけです。

du で表示されたファイル使用量の結果を sort でソートできたら、あとはソートの結果から上位20を表示させるだけ。head は ファイルの最初の部分を表示させるコマンドなので、大きいもの順に並んだ出力結果を最初から20行表示させると、それがトップ20になるわけです。

2ch 関連スレッド

- Linux板

シェルスクリプト総合@LINUX

- 5 <http://pc11.2ch.net/test/read.cgi/linux/1238764663/>
- 4 <http://pc11.2ch.net/test/read.cgi/linux/1210999497/>
- 3 <http://pc11.2ch.net/test/read.cgi/linux/1184077033/>
- 2 <http://pc11.2ch.net/test/read.cgi/linux/1154578200/>
- 1 <http://pc8.2ch.net/test/read.cgi/linux/1121994321/>

- Unix板

シェルスクリプト総合

- 18 <http://hibari.2ch.net/test/read.cgi/unix/1308195527/>
- 17 <http://hibari.2ch.net/test/read.cgi/unix/1290209379/>
- 16 <http://hibari.2ch.net/test/read.cgi/unix/1266642605/>
- 15 <http://pc12.2ch.net/test/read.cgi/unix/1246408968/>
- 14 <http://pc12.2ch.net/test/read.cgi/unix/1233179688/>
- 13 <http://pc11.2ch.net/test/read.cgi/unix/1224085718/>
- 12 <http://pc11.2ch.net/test/read.cgi/unix/1218277263/>
- 11 <http://pc11.2ch.net/test/read.cgi/unix/1211284684/>
- 10 <http://pc11.2ch.net/test/read.cgi/unix/1202725267/>
- 9 <http://pc11.2ch.net/test/read.cgi/unix/1187130302/>
- 8 <http://pc11.2ch.net/test/read.cgi/unix/1171517324/>
- 7 <http://pc10.2ch.net/test/read.cgi/unix/1157601611/>
- 6 <http://pc10.2ch.net/test/read.cgi/unix/1143302182/>
- 5 <http://pc8.2ch.net/test/read.cgi/unix/1137801629/>
- 4 <http://pc8.2ch.net/test/read.cgi/unix/1131026501/>

- 3 <http://pc8.2ch.net/test/read.cgi/unix/1124889646/>
-
- 2 <http://pc8.2ch.net/test/read.cgi/unix/1113664637/>
- 1 <http://pc8.2ch.net/test/read.cgi/unix/1101820646/>