

# システムプログラム 第9回

---

創域理工学部 情報計算科学科

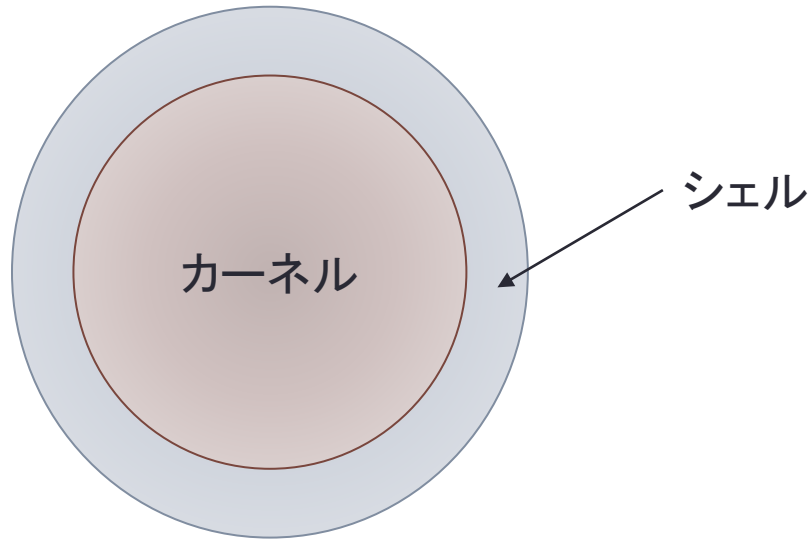
松澤 智史

# 本日の内容

- シェル(shell)
- シェルスクリプト(シェルプログラミング)

# シェル

- ターミナル等で動作しユーザと対話的な作業を行うプログラム
- 貝，殻などの意味を持つが，下図のようにカーネルを覆う役割として命名



# カーネルとシェル

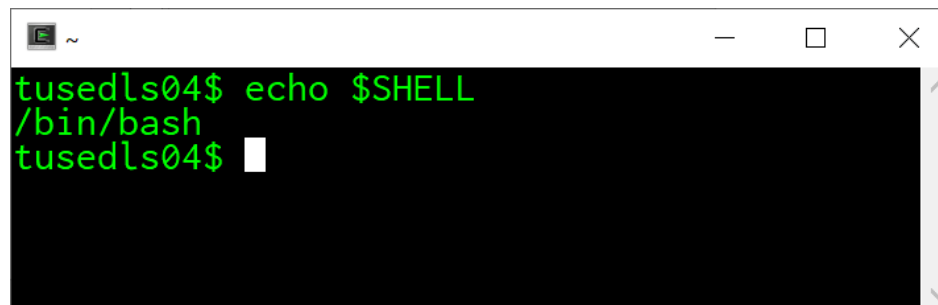
- カーネル(復習)
  - プロセス管理
  - メモリ管理
  - ファイル管理, ファイルシステム
  - ネットワーク等のデバイス管理
  - etc
- シェル
  - ユーザとカーネルのインタフェース
  - インタプリタとして動作し, 対話型でユーザの入力を受理
  - 非対話型でも動作可能

# シェルがないと・・・

- 計算機を操作するにはプログラムをメモリに格納させないといけない
- OSの起動時にプログラムをロードするように読み取り媒体に格納する必要がある
- 新しいプログラムを動作させるには毎回OSの起動が必要となる
- OS(カーネル)稼働中に、任意のタイミングで任意の命令やプログラムを動作させるために存在するソフトウェアである
- OS起動させてやらせたいプログラムを自動で動かして、終わったらシャットダウンするようなシステムならシェルはいらない
- 汎用OSの場合は、OS起動時に最低一つのシェルを起動する

# ターミナル, コマンドプロンプト

- シェルへの入出力をサポートするソフトウェア
- 使用するシェルが固定のターミナル, 切り替え可能なターミナルが存在
- Linux, MacOSのターミナルの場合は  
\$ echo \$SHELL  
で現在のシェルのプログラムを表示
- WindowsのPowerShellは, PowerShell固定



```
tusedls04$ echo $SHELL
/bin/bash
tusedls04$
```

# シェルの種類

- Bourne シェル系
  - UNIX系シェルのベースシェル
    - sh
    - bash(最近のLinux標準シェル)
    - ksh
    - zsh( Bourneシェル+Cシェル)
- C シェル系
  - C言語ライクなシェル
    - csh
    - tcsh

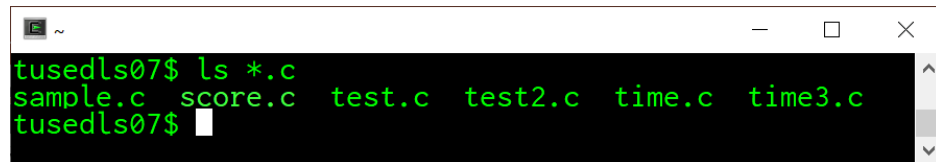
# シェルはスクリプト言語

- スクリプト言語
  - 簡易的なプログラミング言語
  - インタプリタ形式が多い(コンパイルが不要)
- シェルスクリプト
  - コマンドラインインタプリタ向けに書かれたスクリプト言語
  - ユーザとの対話もユーザはシェルスクリプトで記述
  - シェルの関数なども存在

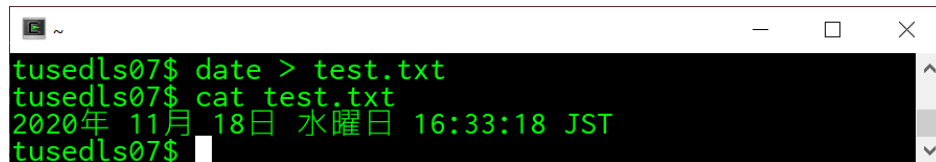


# シェル(シェルスクリプト)の機能

- グロブ
  - ワイルドカードによる文字列マッチング
- パイプとリダイレクト
  - 入出力接続



```
tusedls07$ ls *.c
sample.c score.c test.c test2.c time.c time3.c
tusedls07$
```



```
tusedls07$ date > test.txt
tusedls07$ cat test.txt
2020年 11月 18日 水曜日 16:33:18 JST
tusedls07$
```

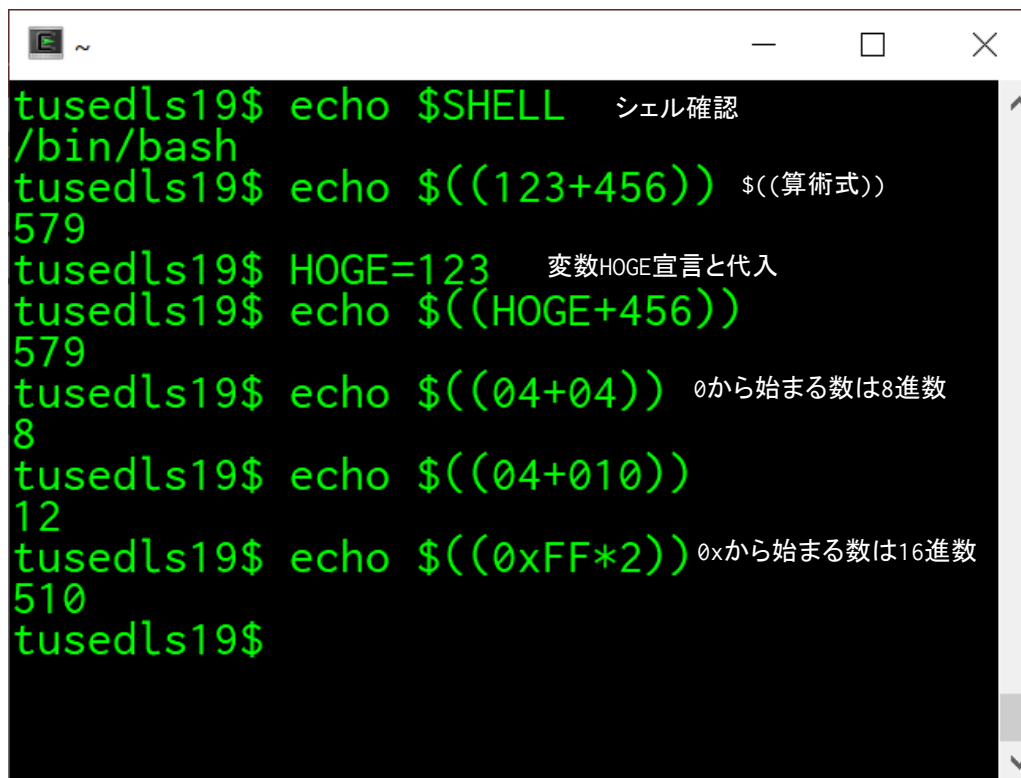
- 置換
  - 後述
- 変数や条件分岐
  - 一般的なプログラム言語と同様(後述)

etc

# シェル(bash)の演算

- 現在のLinux標準シェル
- 新しいプロセスを立ち上げずに整数演算可能

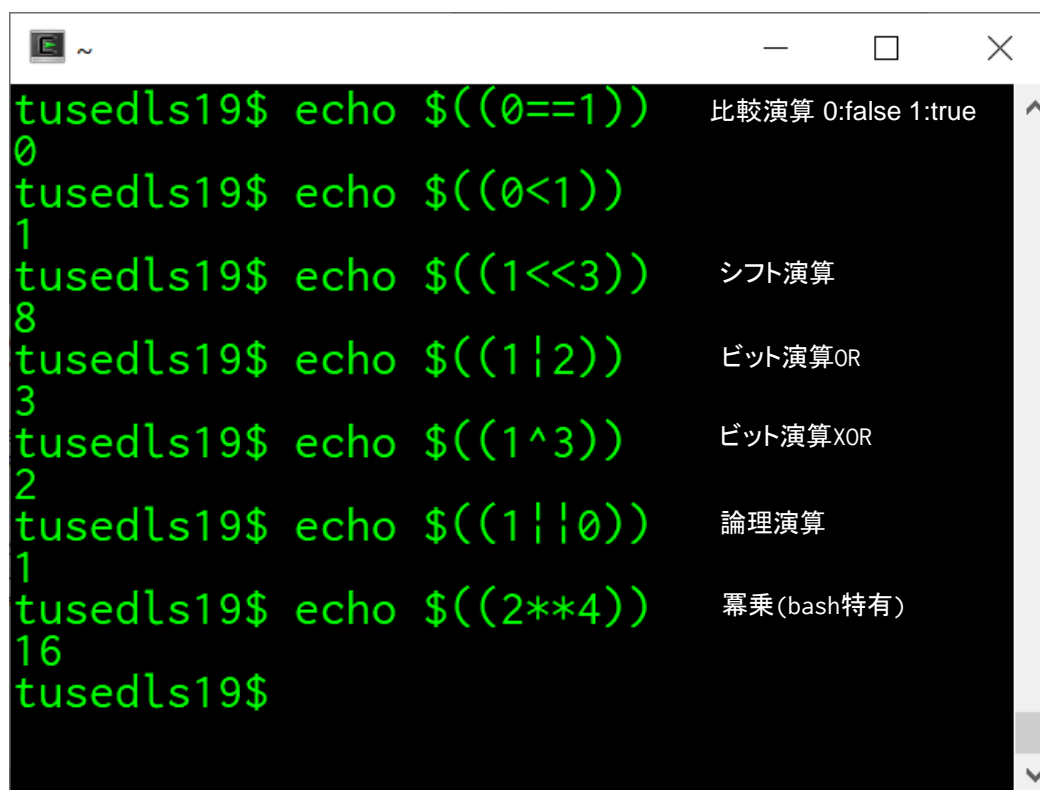
## 四則演算

A terminal window with a black background and green text. The window title bar shows a file icon, a tilde (~), and standard window controls (minimize, maximize, close). The terminal content shows a series of commands and their outputs, with Japanese annotations in gray text. The commands demonstrate basic arithmetic, variable assignment and use, and octal/hexadecimal calculations.

```
tusedls19$ echo $SHELL      シェル確認
/bin/bash
tusedls19$ echo $((123+456))  $((算術式))
579
tusedls19$ HOG=123          変数HOG宣言と代入
tusedls19$ echo $((HOG+456))
579
tusedls19$ echo $((04+04))   0から始まる数は8進数
8
tusedls19$ echo $((04+010))
12
tusedls19$ echo $((0xFF*2))  0xから始まる数は16進数
510
tusedls19$
```

# シェル(bash)の演算

## 他の演算



```
tusedls19$ echo $((0==1))
0
tusedls19$ echo $((0<1))
1
tusedls19$ echo $((1<<3))
8
tusedls19$ echo $((1|2))
3
tusedls19$ echo $((1^3))
2
tusedls19$ echo $((1||0))
1
tusedls19$ echo $((2**4))
16
tusedls19$
```

比較演算 0:false 1:true

シフト演算

ビット演算OR

ビット演算XOR

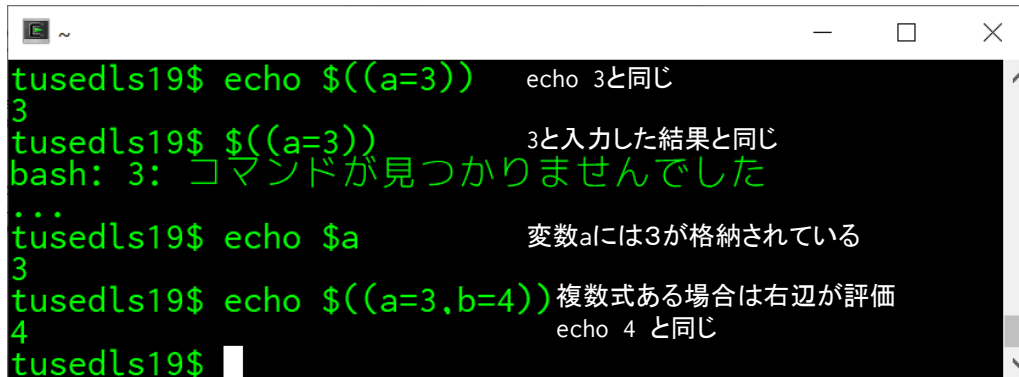
論理演算

冪乗(bash特有)

# シェル(bash)の構文

- 算術式展開

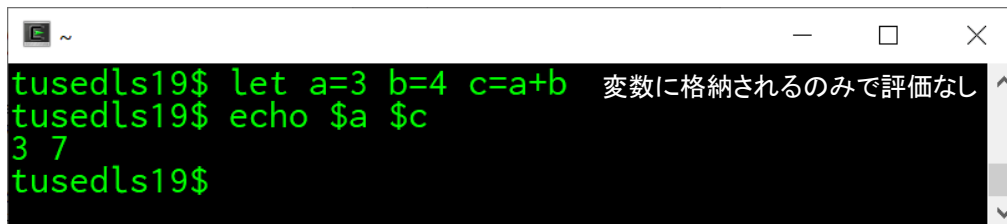
- `$(( 式 ))`



```
tusedls19$ echo $((a=3))  echo 3と同じ
3
tusedls19$ $((a=3))      3と入力した結果と同じ
bash: 3: コマンドが見つかりませんでした
tusedls19$ echo $a       変数aには3が格納されている
3
tusedls19$ echo $((a=3,b=4)) 複数次ある場合は右辺が評価
4                           echo 4 と同じ
tusedls19$
```

- let 式

- `let a=3 b=4 c=a+b`



```
tusedls19$ let a=3 b=4 c=a+b 変数に格納されるのみで評価なし
tusedls19$ echo $a $c
3 7
tusedls19$
```

# シェル(bash)の構文

- for文
  - for((式1;式2;式3));
    - 式1が評価され, 式2が真である間繰り返す, ループ1回ごとに式3が評価
  - for 変数 in 値のリスト;
    - 値のリストの順に変数に格納されてループ

```
tusedls19$ for ((i=0;i<10;i++));  
> do  
> echo $i  
> done  
0  
1  
2  
3  
4  
5  
6  
7  
8  
9  
tusedls19$
```

```
tusedls19$ for i in {0..9};  
> do  
> echo $i  
> done  
0  
1  
2  
3  
4  
5  
6  
7  
8  
9  
tusedls19$
```

```
tusedls19$ for i in 0 1 2 3 4 5 6 7 8 9;  
> do  
> echo $i  
> done  
0  
1  
2  
3  
4  
5  
6  
7  
8  
9  
tusedls19$
```

# シェル(bash)のfor文 応用例

```
tusedls19$ ls
test1.txt  test2.txt
tusedls19$ for f in *.txt;
> do
> echo $f
> done
test1.txt
test2.txt
tusedls19$ for f in *.txt;
> do
> mv $f $f.bak
> done
tusedls19$ ls
test1.txt.bak  test2.txt.bak
tusedls19$
```

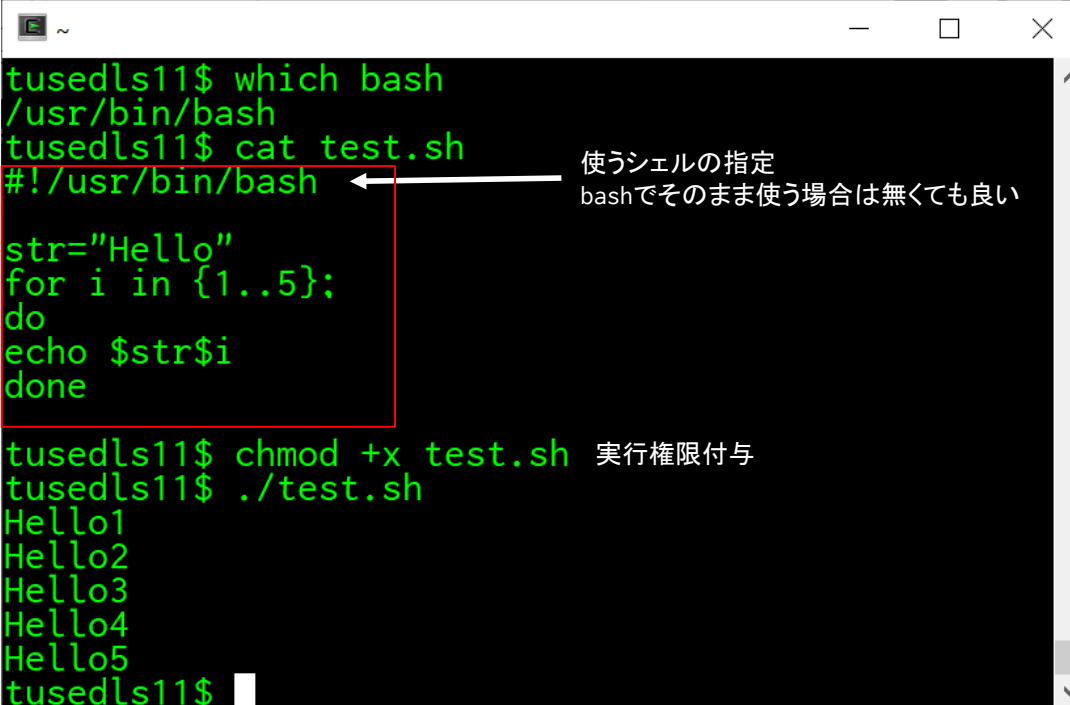
ファイルのバックアップ

```
tusedls19$ seq 4
1
2
3
4
tusedls19$ for i in `seq 4`;
> do
> echo $i
> done
1
2
3
4
tusedls19$
```

seqなどのコマンドと組み合わせる

# シェルのスクリプトファイル

- シェルスクリプトをファイルにしておき、スクリプトファイルから起動することも可能



```
tusedls11$ which bash
/usr/bin/bash
tusedls11$ cat test.sh
#!/usr/bin/bash
str="Hello"
for i in {1..5};
do
echo $str$i
done
tusedls11$ chmod +x test.sh
tusedls11$ ./test.sh
Hello1
Hello2
Hello3
Hello4
Hello5
tusedls11$
```

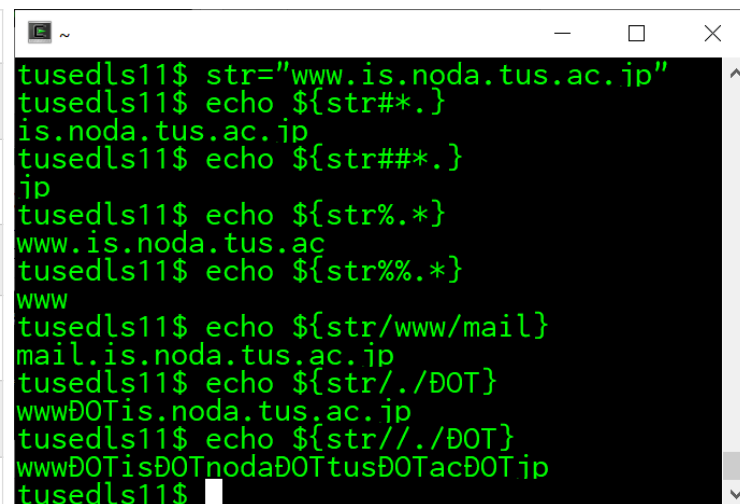
使うシェルの指定  
bashでそのまま使う場合は無くても良い

実行権限付与

# 文字列置換

- シェルの評価は文字列として行われるため、文字列置換のテクニックは使用勝手が良い

パターン	動作
<code>\${変数#パターン}</code>	文字列先頭の最短マッチ部分を削除
<code>\${変数##パターン}</code>	文字列先頭の最長マッチ部分を削除
<code>\${変数%パターン}</code>	文字列末尾の最短マッチ部分を削除
<code>\${変数%%パターン}</code>	文字列末尾の最長マッチ部分を削除
<code>\${変数/検索文字列/置換文字列}</code>	最初にマッチしたもののみ文字列を置換
<code>\${変数//検索文字列/置換文字列}</code>	全ての文字列を置換



```
tusedls11$ str="www.is.noda.tus.ac.jp"
tusedls11$ echo ${str#*.}
is.noda.tus.ac.jp
tusedls11$ echo ${str##*.}
ip
tusedls11$ echo ${str%.*}
www.is.noda.tus.ac
tusedls11$ echo ${str%%.*}
www
tusedls11$ echo ${str/www/mail}
mail.is.noda.tus.ac.jp
tusedls11$ echo ${str/./DOT}
wwwDOTis.noda.tus.ac.jp
tusedls11$ echo ${str//./DOT}
wwwDOTisDOTnodaDOTtusDOTacDOTip
tusedls11$
```



# 変数と環境変数

- 変数

- 新規に定義可能
- そのシェル(プロセス)でのみ有効
- 変数名="Hello"のように定義

- 環境変数

- 新規に定義可能
- 事前にある程度の変数が用意されている
- 子プロセスにも引き継がれる
- `export 変数名` で変数を環境変数に変更する



```
tusedls11$ val1="ABCD"
tusedls11$ val2="EFGH"
tusedls11$ export val1
tusedls11$ bash
tusedls11$ echo $val1
ABCD
tusedls11$ echo $val2
tusedls11$
```

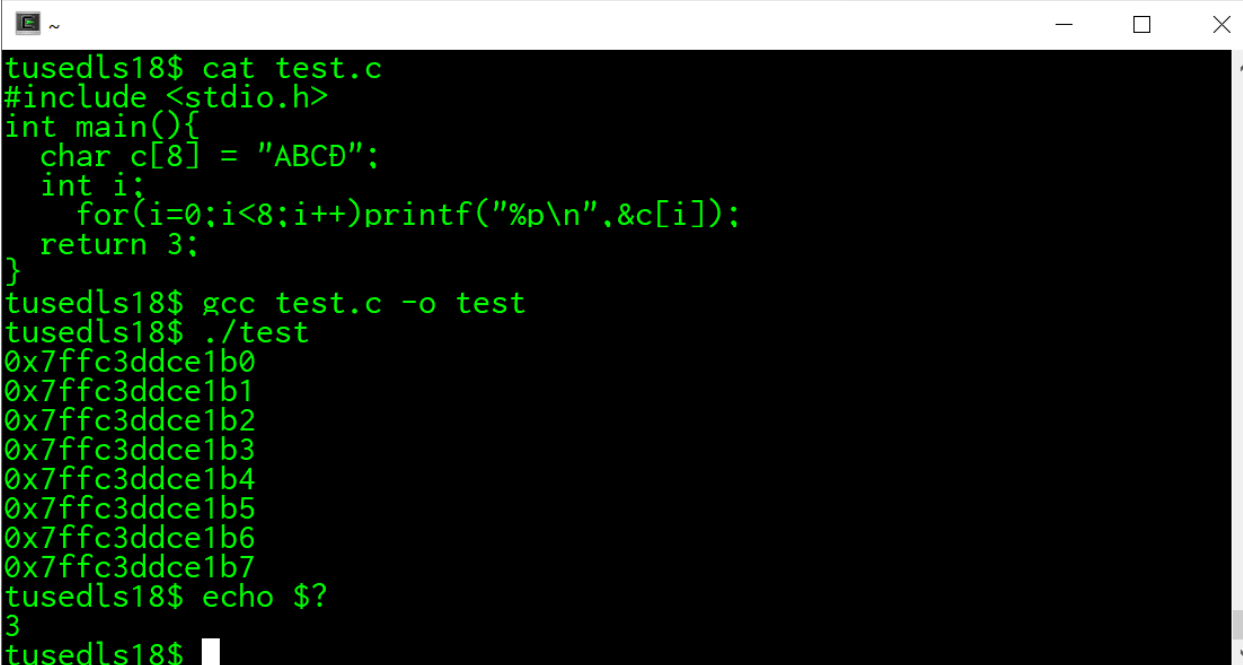
val1を環境変数へ

← 新規にbash起動

val2は空文字

# 余談 C言語 mainの返り値

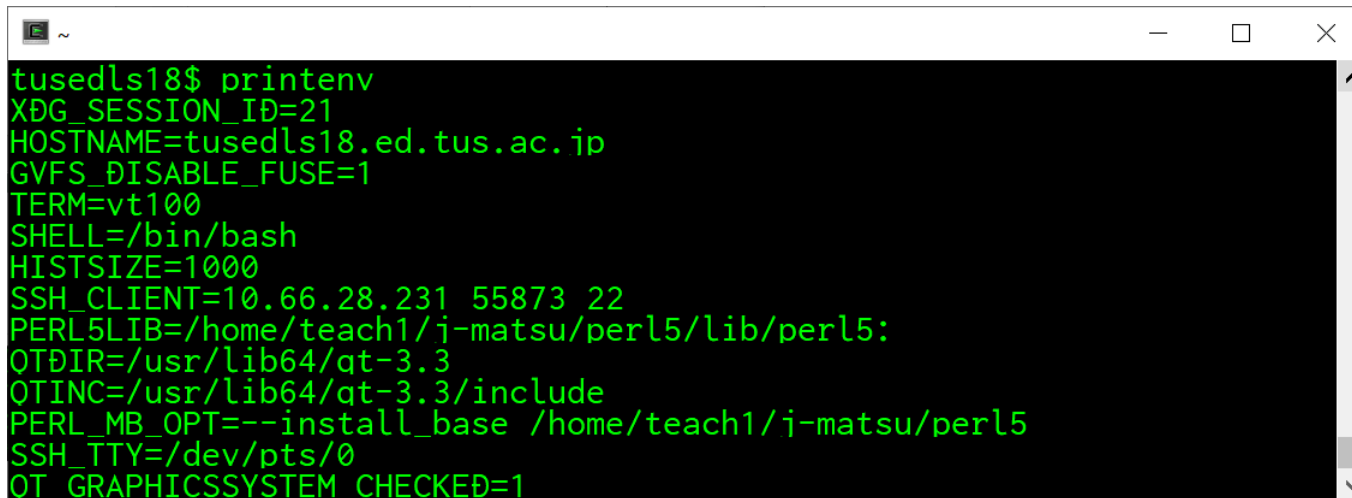
- C言語等のプログラムの返り値もシェルの変数\$?で保持



```
tusedls18$ cat test.c
#include <stdio.h>
int main(){
    char c[8] = "ABCĐ";
    int i;
    for(i=0;i<8;i++)printf("%p\n",&c[i]);
    return 3;
}
tusedls18$ gcc test.c -o test
tusedls18$ ./test
0x7ffc3ddce1b0
0x7ffc3ddce1b1
0x7ffc3ddce1b2
0x7ffc3ddce1b3
0x7ffc3ddce1b4
0x7ffc3ddce1b5
0x7ffc3ddce1b6
0x7ffc3ddce1b7
tusedls18$ echo $?
3
tusedls18$
```

# 事前に用意されている環境変数

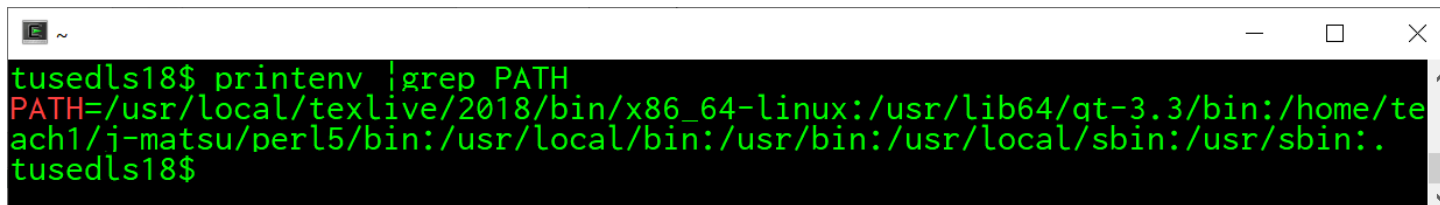
- printenv 定義されている環境変数一覧



```
tusedls18$ printenv
XDG_SESSION_ID=21
HOSTNAME=tusedls18.ed.tus.ac.jp
GVFS_DISABLE_FUSE=1
TERM=vt100
SHELL=/bin/bash
HISTSIZE=1000
SSH_CLIENT=10.66.28.231 55873 22
PERL5LIB=/home/teach1/j-matsu/perl5/lib/perl5:
QTDIR=/usr/lib64/qt-3.3
QTINC=/usr/lib64/qt-3.3/include
PERL_MB_OPT=--install_base /home/teach1/j-matsu/perl5
SSH_TTY=/dev/pts/0
QT_GRAPHICSSYSTEM_CHECKED=1
```

- PATH変数

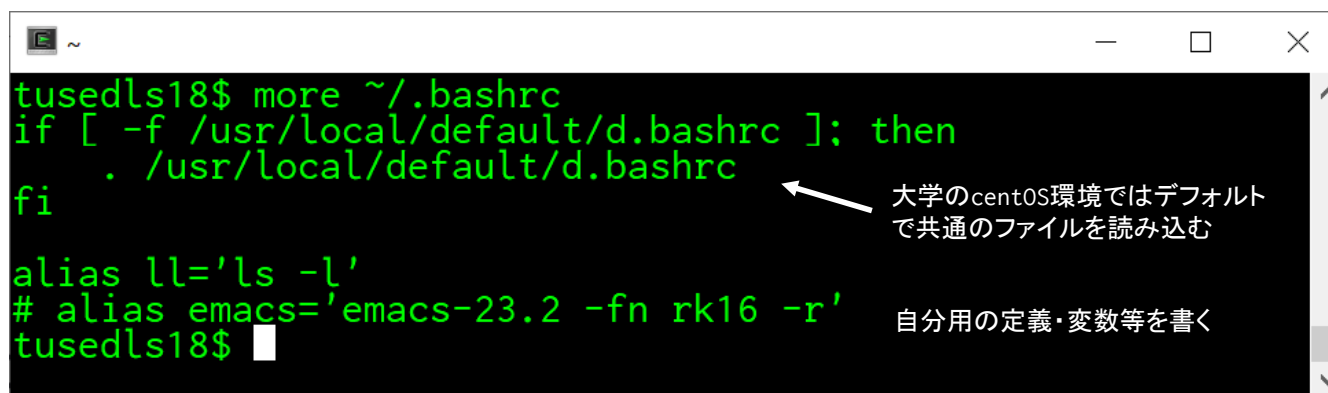
- 評価した文字列(コマンド)を実行する際に調べるディレクトリ



```
tusedls18$ printenv | grep PATH
PATH=/usr/local/texlive/2018/bin/x86_64-linux:/usr/lib64/qt-3.3/bin:/home/teach1/j-matsu/perl5/bin:/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:.
```

# シェル(bash)の設定

- ~/.bashrc
  - シェル起動時に読み込まれる
  - よく使うオリジナル変数などはここに書いておいても良い



```
tusedls18$ more ~/.bashrc
if [ -f /usr/local/default/d.bashrc ]; then
    . /usr/local/default/d.bashrc
fi

alias ll='ls -l'
# alias emacs='emacs-23.2 -fn rk16 -r'
tusedls18$
```

大学のcentOS環境ではデフォルトで共通のファイルを読み込む

自分用の定義・変数等を書く

- alias 定義したコマンドを右辺のコマンドに文字列置き換えて評価
  - bashにもある

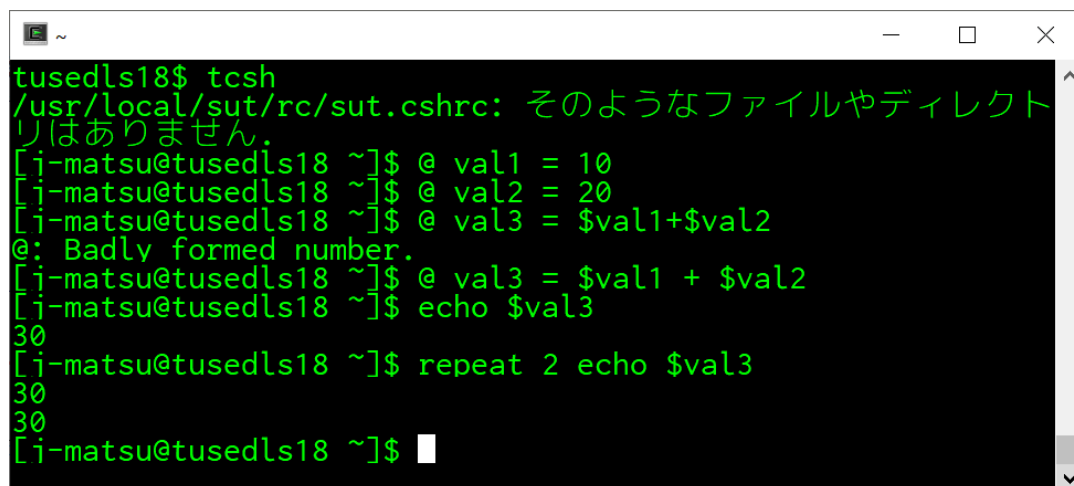
# tcsch

- Cシェル(C言語ライク)系のシェル
- cshの後継シェル
- ヒストリ機能, 文字列補完・編集機能等を備える
  - 現在はbashも同様の機能を保持
- UNIX系標準シェルとして使われていた
- シェル比較
  - <http://www.faqs.org/faqs/unix-faq/shell/shell-differences/>

	sh	csh	ks	bash	tcsch	zsh	rc	es
Job control	N	Y	Y	Y	Y	Y	N	N
Aliases	N	Y	Y	Y	Y	Y	N	N
Shell functions	Y(1)	N	Y	Y	N	Y	Y	Y
"Sensible" Input/Output redirection	Y	N	Y	Y	N	Y	Y	Y
Directory stack	N	Y	Y	Y	Y	Y	F	F
Command history	N	Y	Y	Y	Y	Y	L	L
Command line editing	N	N	Y	Y	Y	Y	L	L
Vi Command line editing	N	N	Y	Y	Y(3)	Y	L	L
Emacs Command line editing	N	N	Y	Y	Y	Y	L	L
Rebindable Command line editing	N	N	N	Y	Y	Y	L	L
User name look up	N	Y	Y	Y	Y	Y	L	L
Login/Logout watching	N	N	N	N	Y	Y	F	F
Filename completion	N	Y(1)	Y	Y	Y	Y	L	L
Username completion	N	Y(2)	Y	Y	Y	Y	L	L
Hostname completion	N	Y(2)	Y	Y	Y	Y	L	L
History completion	N	N	N	Y	Y	Y	L	L
Fully programmable Completion	N	N	N	N	Y	Y	N	N
Mh Mailbox completion	N	N	N	N(4)	N(6)	N(6)	N	N
Co Processes	N	N	Y	N	N	Y	N	N
Builtin arithmetic evaluation	N	Y	Y	Y	Y	Y	N	N
Can follow symbolic links invisibly	N	N	Y	Y	Y	Y	N	N
Periodic command execution	N	N	N	N	Y	Y	N	N
Custom Prompt (easily)	N	N	Y	Y	Y	Y	Y	Y
Sun Keyboard Hack	N	N	N	N	N	Y	N	N
Spelling Correction	N	N	N	N	Y	Y	N	N
Process Substitution	N	N	N	Y(2)	N	Y	Y	Y
Underlying Syntax	sh	csh	sh	sh	csh	sh	rc	rc
Freely Available	N	N	N(5)	Y	Y	Y	Y	Y
Checks Mailbox	N	Y	Y	Y	Y	Y	F	F
Tty Sanity Checking	N	N	N	N	Y	Y	N	N
Can cope with large argument lists	Y	N	Y	Y	Y	Y	Y	Y
Has non-interactive startup file	N	Y	Y(7)	Y(7)	Y	Y	N	N
Has non-login startup file	N	Y	Y(7)	Y	Y	Y	N	N
Can avoid user startup files	N	Y	N	Y	N	Y	Y	Y
Can specify startup file	N	N	Y	Y	N	N	N	N
Low level command redefinition	N	N	N	N	N	N	N	Y
Has anonymous functions	N	N	N	N	N	N	Y	Y
List Variables	N	Y	Y	N	Y	Y	Y	Y
Full signal trap handling	Y	N	Y	Y	N	Y	Y	Y
File no clobber ability	N	Y	Y	Y	Y	Y	N	F
Local variables	N	N	Y	Y	N	Y	Y	Y
Lexically scoped variables	N	N	N	N	N	N	N	Y
Exceptions	N	N	N	N	N	N	N	Y

# tcsh

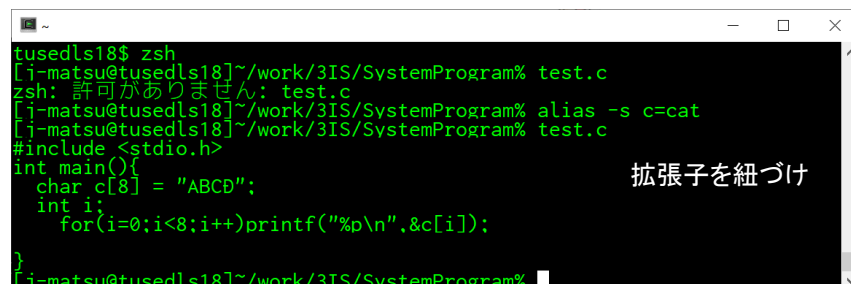
- 環境変数
  - setenv 環境変数名 値
  - unsetenv 環境変数
- 演算
  - @ 式
- tcshのみサポートするコマンドもあり
  - repeat など
- 設定ファイル
  - ~/.cshrc



```
tusedls18$ tcsh
/usr/local/sut/rc/sut.cshrc: そのようなファイルやディレクトリはありません.
[i-matsu@tusedls18 ~]$ @ val1 = 10
[i-matsu@tusedls18 ~]$ @ val2 = 20
[i-matsu@tusedls18 ~]$ @ val3 = $val1+$val2
@: Badly formed number.
[i-matsu@tusedls18 ~]$ @ val3 = $val1 + $val2
[i-matsu@tusedls18 ~]$ echo $val3
30
[i-matsu@tusedls18 ~]$ repeat 2 echo $val3
30
30
[i-matsu@tusedls18 ~]$
```

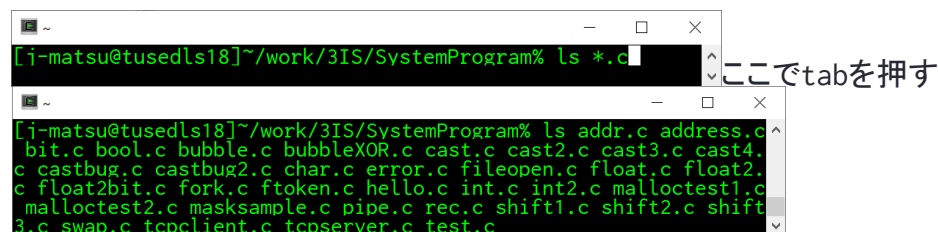
# zsh

- Z Shell(ジーシェル)と呼ぶ
- bash系とcsh系の融合シェル
- macOSはCatalinaから標準シェル
- 起動時少し重い
- zsh専用の機能あり
  - alias -s 拡張子と起動プログラムを結びつける
  - 便利なtabキー
    - \*を展開
    - 候補をtabキー押す順に変更
  - 専用エディタzed
    - 軽い
    - Emacs風に使える
      - ctrl+x, wで保存
      - ctrl+c で保存せず終了



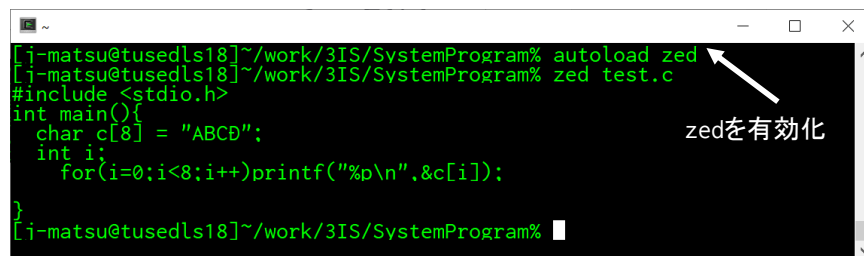
```
tusedls18$ zsh
[j-matsu@tusedls18]~/work/3IS/SystemProgram% test.c
zsh: 許可がありません: test.c
[j-matsu@tusedls18]~/work/3IS/SystemProgram% alias -s c=cat
[j-matsu@tusedls18]~/work/3IS/SystemProgram% test.c
#include <stdio.h>
int main(){
    char c[8] = "ABCD";
    int i;
    for(i=0;i<8;i++)printf("%p\n",&c[i]);
}
[j-matsu@tusedls18]~/work/3IS/SystemProgram%
```

拡張子を紐づけ



```
[j-matsu@tusedls18]~/work/3IS/SystemProgram% ls *.c
[j-matsu@tusedls18]~/work/3IS/SystemProgram% ls addr.c address.c
bit.c bool.c bubble.c bubbleXOR.c cast.c cast2.c cast3.c cast4.
c castbug.c castbug2.c char.c error.c fopen.c float.c float2.
c float2bit.c fork.c ftoken.c hello.c int.c int2.c malloc1.c
malloc2.c masksample.c pipe.c rec.c shift1.c shift2.c shift
3.c swap.c tcpclient.c tcpserver.c test.c
```

ここでtabを押す



```
[j-matsu@tusedls18]~/work/3IS/SystemProgram% autoload zed
[j-matsu@tusedls18]~/work/3IS/SystemProgram% zed test.c
#include <stdio.h>
int main(){
    char c[8] = "ABCD";
    int i;
    for(i=0;i<8;i++)printf("%p\n",&c[i]);
}
[j-matsu@tusedls18]~/work/3IS/SystemProgram%
```

zedを有効化

# まとめ

## シェル

- カーネルを対話的に使うためのソフトウェア
- シェルにはいくつかの種類がある
  - B shell系 bash
  - C shell系 tcsh
  - 複合型 zsh
- シェル自体がスクリプトの言語になっている
  - 変数・構文



質問あればどうぞ