名前

screen – VT100/ANSI 端末エミュレーション機能を持つ画面管理ソフトウェア

書式

screen [**−***options*] [*cmd* [*args*]]
screen −r [[*pid***.**]*tty*[**.***host*]]
screen −r *sessionowner***/**[[*pid***.**]*tty*[**.***host*]]

書式

screen は、ひとつの物理的な端末を複数のプロセス (特に対話シェル) で共有化できるようにする、フルスクリーンウィンドウ管理ソフトウェアである。各仮想端末は DEC VT100 端末の機能に加え、ANSI X3.64 (ISO 6429) や ISO 2022 規格に含まれる制御機能 (例えば行の挿入/削除や複数の文字集合のサポート) のいくつかを備えている。各仮想端末にはスクロールバック用の履歴バッファがある。またコピー&ペースト機能もあり、ウィンドウ間でテキスト領域をやりとりできる。

screen は呼び出されるとウィンドウを 1 つ生成し、その中でシェル (または指定されたコマンド) を実行する。そしてユーザからは見えなくなり、ユーザは通常通りにプログラムを実行できるようになる。その後はいつでも、新しい (全画面) ウィンドウの生成およびそこでの他のプログラムの実行 (別のシェルも含む)、いまあるウィンドウの削除、ウィンドウリストの表示、ログ出力の有効化・無効化、ウィンドウ間でのテキストのコピー&ペースト、スクロールバック履歴の閲覧、望む通りのウィンドウ切り替え、等ができる。各々のウィンドウは、それぞれ自分のプログラムを他からは完全に独立したかたちで動作させる。ウィンドウが現在見えなくても、あるいは *screen* セッション全体がユーザの端末から切り離された場合でも、各プログラムは動作を続ける。プログラムが終了すると、 *screen* は (デフォルトでは) そのプログラムが動作していたウィンドウを削除する。このウィンドウがフォアグラウンドにあった場合は、ひとつ前のウィンドウに表示が切り替わる。ウィンドウが残っていなかった場合には、 *screen* は終了する。

ユーザのキー入力はすべて、現在のウィンドウで動作しているプログラムに送られる。この唯一の例外は、ウィンドウマネージャへ送るコマンドの開始を意味する、ある 1 つのキーストロークである。デフォルトでは、各コマンドは Ctrl-a (以降は C-a と省略) で始まり、その後には別のキーストロークが 1 つ続く。コマンド文字とすべてのキー割り当ては完全に望みの通りにカスタマイズできる。ただし長さは必ず 2 文字でなければならない。

screen はプレフィクス “C-” をコントロールキーの意味だとは理解しない。コマンドへの引数にはキャレット表記 (“C-a” の代わりに “^A”) を用いること (例えば *escape* コマンドや *-e* オプションへの引数など)。 *screen* 自身も制御文字の出力にはキャレット表記を用いる。

新しいウィンドウを生成する標準的な方法は、“C-a c” とタイプすることである。これにより新しいウィンドウが生成されてシェルが実行され、現在のウィンドウで動作中のプロセスの状態にかかわらず、ただちにそのウィンドウへ切り替わる。同様に、あらかじめコマンドをキー操作に割り当てておき (割り当ては .screenrc またはコマンドラインから “C-a :” で行う)、その後これを “C-a c” のように用いれば、特定のコマンドが実行される新たなウィンドウを生成できる。さらに、既存ウィンドウのプロンプトから

```
screen emacs prog.c
```

のようなコマンドを実行しても、新たなウィンドウを生成できる。これは別の *screen* を起動するのではなく、コマンド名と引き数を (環境変数 $STY で指定される) ウィンドウマネージャに渡し、新しいウィンドウを生成させる。上の例では (prog.c を編集する) emacs エディタが起動し、そのウィンドウに切り替わる。

screen が “/etc/utmp” に書き込み可能な場合は、各ウィンドウに関する適切な記録がこのファイルに書き込まれ、そしてウィンドウの終了時に削除される。これは “talk”, “script”, “shutdown”, “rsend”, “sccs” 等、utmp ファイルを見てユーザの居場所を調べるプログラムを使う際に便利である。 *screen* が端末上でアクティブであるうちは、その端末自身のレコードは utmp ファイルから削除される。 “C-a L” も参照すること。

はじめよう

他の termcap/terminfo プログラムの場合と同様、 *screen* を使い始める前には、端末のタイプを正しく選んでいるかを確認すること (*tset* プログラム等で確認できる)。

大量の文書を読まずに使い始めたいせっかちな人も、 "C-a ?" コマンドだけは覚えておくように。この 2 文字を入力すると *screen* で使えるコマンドとキー割り当てのリストが表示される。それぞれのキー操作については、『デフォルトのキー割り当て』の節で説明する。このマニュアルの『カスタマイズ』の節では、.screenrc の内容を扱う。

使っている端末が「本当の」自動マージン端末 (画面をスクロールさせないと画面の末尾のカラムを更新できない) の場合、その端末の termcap には自動マージンが「オフ」であるものを使う方が良い。これにより、正確かつ最適な画面更新がどんな環境でも保証される。今日のほとんどの端末は "magic" マージン (自動マージンで、かつ最後のカラムも利用できる) を備えている。これは VT100 の形式で、 *screen* にも完璧に適している。「本当の」自動マージン端末しか使えない場合、 *screen* はこれを受け入れて使用するが、画面の最後のカラムに置かれた文字は、画面がスクロールするか、その文字が何らかの方法で安全な位置に移動するまで更新できない。文字挿入 (insert-character) の機能を持つ端末を使えば、この遅れを短くすることができる。

コマンドラインオプション

screen では以下のコマンドラインオプションが使用できる:

–a temcap の *all* 機能の実装に、ディスプレイ各部の再描画が必要とされる場合であっても、各ウィンドウの termcap に (一部の例外を除いて) *all* 機能を含める。

–A すべてのウィンドウの大きさを現在の端末の大きさに合わせる。デフォルトでは *screen* はサイズ変更が可能な端末 (記述に "WS" があるもの。例としては suncmd や xterm の一部) へアタッチする際には、古いウィンドウの大きさを復元しようとする。

–c *file*
設定ファイルをデフォルトの "$HOME/.screenrc" ではなく *file* にする。

–d|–D [*pid.tty.host*]
screen を起動せず、他のところで動作している *screen* のセッションをデタッチする。これは、 *screen* の制御端末で "C-a d" を入力したときの効果と同じである。 **–D** は強制デタッチのキーと同じである。デタッチできるセッションが無いときには、このオプションは無視される。 **–r/–R** オプションと組み合わせれば、より強力な機能を実現できる。

–d –r セッションを再アタッチする。必要ならばまずデタッチする。

–d –R セッションを再アタッチする。必要ならばまずデタッチするか、あるいはセッションを生成する。

–d –RR セッションを再アタッチする。必要ならばまずデタッチするか、あるいはセッションを生成する。複数のセッションをアタッチ可能な場合は、最初のセッションを使う。

–D –r セッションを再アタッチする。必要ならばリモートのデタッチとログアウトを先に行う。

–D –R アタッチを今ここで行う。具体的には、セッションが走っている場合には再アタッチを行い、必要ならばまずリモートのデタッチとログアウトを行う。動作中のセッションがなければ生成してユーザに知らせる。これは筆者の好みである。

–D –RR
アタッチを今ここで行う。詳細はともかく、とにかくこれを使えばよろしい。

参考: ときどき "screen –list" を使って、自分のセッションの状態をチェックすると良いだろう。

–e *xy*
コマンド文字を *x* にする。またリテラルなコマンド文字を生成する文字を *y* にする (これをコマンド文字の後に入力する)。デフォルトは "C-a" と 'a' であり、これらは "-e^Aa" のように指定できる。 *screen* セッションを生成するとき、このオプションはデフォルトのコ

マンド文字を設定する。マルチユーザセッションでは、追加されたすべてのユーザは、最初はこのコマンド文字を使うことになる。しかし、既に動作中のセッションにアタッチする時には、このオプションはアタッチするユーザのコマンド文字しか変更しない。このオプションはそれぞれ “defescape” および “escape” と同じである。

–f, **–fn**, **–fa**
フロー制御の有効、無効を切り替える。また「自動切り替えモード (automatic switching mode)」を設定する。これは .screenrc の “defflow” コマンドを使っても定義できる。

–h *num*
履歴のスクロールバッファの大きさを最大 *num* 行に設定する。

–i フロー制御が有効である時に、割り込みキー (通常は C-c) によって即座にディスプレイに割り込みがかかるようにする。詳しくは .screenrc のコマンドである “defflow” を参照のこと。このオプションはあまり使わない方が良い。

–l および **–ln**
(/etc/utmp の更新に関する) ログインモードを有効/無効にする。これは .screenrc のコマンドの “deflogin” でも指定できる。

–ls および **–list**
screen を起動せず、ユーザの *screen* セッションに対応する *pid.tty.host* 形式の文字列をリスト表示する。 ‘detached’ と記されているセッションは “screen -r” で再開できる。 ‘attached’ と記されているセッションは動作中であり、これを制御している端末がある。セッションがマルチユーザモードで動作している場合は、‘multi’ と記される。 ‘unreachable’ と記されているセッションは、別のホストで動作しているか ‘dead’ の状態にある。 unreachable なセッションのうち、名前がローカルホストや指定したパラメータがあればそれにマッチするものは、 dead であると思われる。マッチの作りかたの説明は **-r** フラグの項目を見てほしい。 ‘dead’ とされたセッションは、厳しくチェックしてから削除すべきである。よくわからない場合はシステム管理者に相談すること。セッションは **-wipe** オプションで削除する。

–L ウィンドウの自動出力ログを有効にするよう *screen* に伝える。

–m *screen* に $STY 環境変数を無視させる。 “screen -m” とすると、この *screen* が他の *screen* セッションから呼び出されたかどうかに関わらず、新たなセッションの生成が強制される。このフラグは、‘-d’ オプションと組み合わせると特別な意味を持つ。

–d –m *screen* を “detached” モードで起動する。新たなセッションが生成されるが、そのセッションへアタッチしない。これはシステムのスタートアップスクリプトで便利である。

–D –m これも screen を “detached” モードで起動するが、新たなプロセスをフォークしない。このコマンドは、セッションが終わったら終了する。

–O 完全な VT100 エミュレーションではなく、使っている端末に対してより最適な出力モードを選択する (‘LP’ のない自動マージン端末にのみ影響する)。これは .screenrc の “termcap” コマンドに ‘OP’ を指定することによっても設定できる。

–p *(番号または名前)*
あらかじめウィンドウを選択する。これは特定のウィンドウにアタッチしなおしたいときや、 “-X” オプションを用いて特定のウィンドウにコマンドを送りたいときに便利である。スクリーン選択コマンドと同じく、 “-” を用いるとブランクのウィンドウが選択される。再アタッチの場合においては、 “=” によってブランクのウィンドウにウィンドウのリストが表示される。

–q エラーメッセージを表示しない。 “-ls” と同時に用いた場合の返り値は次の通り: 9 はセッションのないディレクトリ、 10 は動作中だがアタッチできないセッションのあるディレクトリ、 11 (またはそれ以上) は 1 (以上) の利用できるセッションのあるディレクトリ。 “-r” と同時に用いた場合の返り値は次の通り: 10 はレジュームできるセッションがない、 12 (またはそれ以上) は 2 つ (以上) のレジューム可能なセッションがあり、どちらかを選ばなければならない。これ以外の場合には、 “-q” には何の効果もない。

–r [*pid.tty.host*]
–r *sessionowner/*[*pid.tty.host*]
デタッチされている *screen* セッションをレジュームする。他のオプション (“-d”/“-D” と同時に用いる場合を除く) を指定することはできないが、複数の *screen* セッションがデタッチされている場合は、それらを区別するために [*pid.*]*tty.host* という余分なプレフィックスが必要かもしれない。二番目の形式は、マルチユーザモードで動作している他のユーザの screen セッションに接続する場合に用いる。この場合 screen は、他のユーザのディレクトリでセッション探索ができなければならない。これには setuid-root が必要とされる。

–R デタッチされている *screen* セッションのうち最初に見つかったものを再開しようと試みる。成功すれば他のコマンドラインオプションはすべて無視される。デタッチされているセッションがひとつも無い場合には、 **–R** が指定されていなかったかのように、他のオプションを使って新しいセッションを開始する。このオプションは、 *screen* がログインシェルとして実行される場合にはデフォルトで設定される (実際にはこの場合 screen は “-xRR” を使う)。 **–d/–D** オプションとの組み合わせについては、これらのオプションに関する説明の部分を参照すること。

–s デフォルトのシェルとして、環境変数 $SHELL の値 (これが定義されていなければ “/bin/sh”) ではなく、指定されたプログラムを設定する。これは .screenrc コマンドの “shell” を使っても定義できる。

–S セッション名
新しいセッションを作る際、このオプションを使えばセッションに意味のある名前を付けることができる。この名前を使うと、 “screen -list” や “screen -r” の使用時にセッションを識別できる。これはデフォルトの [*tty.host*] サフィックスに置き換わる。

–t 名前
デフォルトのシェルまたは指定プログラムに対してタイトル (別名) を設定する。.screenrc コマンド “shelltitle” も参照のこと。

–U UTF-8 モードで screen を動作させる。このオプションは、ユーザの端末が UTF-8 エンコードされた文字を理解し、また送信してくることを screen に伝える。また新規ウィンドウのデフォルトエンコーディングが ‘utf8’ になる。

–v バージョン番号を表示する。

–wipe [マッチ]
“screen -ls” とほぼ同じ動作であるが、破棄されたセッションに ‘dead’ と印を付けるのではなく、これを削除する。到達できないセッションのうち、ローカルホストの名前か、パラメータを与えた場合はそれにマッチしたものは、 dead とみなされる。マッチの作り方については **-r** フラグの項を参照のこと。

–x デタッチされていない *screen* セッションへのアタッチを行う (マルチディスプレイモード)。

–X 動作中の screen セッションに特定のコマンドを送る。 **-d** や **-r** オプションを用いれば、アタッチされているセッションやデタッチされているセッションのみを探すよう、screen に伝えることもできる。ただしこのコマンドは、セッションがパスワードで保護されている場合には動作しないことに注意。

デフォルトのキー割り当て

既に説明したように、 *screen* の各コマンドは “C-a” の後に、もう 1 文字を続けたものである。ユーザの利便のため、小文字に割り当てられたすべてのコマンドは、それぞれ対応する制御文字にも割り当てられる (ただし “C-a a” は例外である。これについては後述する)。したがって、“C-a c” でも “C-a C-c” でもウィンドウを生成できる。コマンドの説明については『カスタマイズ』のセクションを参照すること。

以下の表はデフォルトのキー割り当てである:

C-a '	(select)	切り替え先のウィンドウ名またはウィンドウ番号を問い合わせる。
C-a "	(windowlist -b)	選択できるウィンドウのリストを表示する。
C-a 0	(select 0)	
...	...	
C-a 9	(select 9)	
C-a -	(select -)	番号 0 – 9 のウィンドウ、またはブランクウィンドウへ切り替える。
C-a tab	(focus)	入力フォーカスを次のリージョンに切り替える。
C-a C-a	(other)	直前に表示していたウィンドウにトグルする。この割り当ては、変更しない限り、デフォルトではコマンド文字の 2 回の繰り返しである。例えば、オプション "**–e]x**" を使うと、このコマンドは "]C-a" でなく "]]" となる。
C-a a	(meta)	コマンド文字(C-a)をウィンドウに送る。*escape* コマンドを参照のこと。
C-a A	(title)	現在のウィンドウの名前をユーザに入力させる。
C-a b		
C-a C-b	(break)	ブレーク信号をウィンドウに送る。
C-a B	(pow_break)	端末ラインを再オープンし、ブレーク信号を送る。
C-a c		
C-a C-c	(screen)	新しいウィンドウとシェルを生成し、そのウィンドウに切り替える。
C-a C	(clear)	画面をクリアする。
C-a d		
C-a C-d	(detach)	現在の端末から *screen* をデタッチする。
C-a D	(pow_detach)	デタッチとログアウトを行う。
C-a f		
C-a C-f	(flow)	フロー制御の *on*, *off*, *auto* をトグルする。
C-a F	(fit)	ウィンドウを現在のリージョの大きさにリサイズする。
C-a C-g	(vbell)	*screen* の可視ベルモードをトグルする。
C-a h	(hardcopy)	現在のウィンドウのハードコピーをファイル "hardcopy.*n*" に書き込む。
C-a H	(log)	現在のウィンドウのファイル "screenlog.*n*" へのログ出力を開始/終了する。
C-a i		
C-a C-i	(info)	現在のウィンドウに関する情報を表示する。
C-a k		
C-a C-k	(kill)	現在のウィンドウを破棄する。
C-a l		
C-a C-l	(redisplay)	現在のウィンドウ全体をリフレッシュする。
C-a L	(login)	このウィンドウのログインスロットをトグルする。これが可能なのは、 *screen* が utmp データベースを更新するよう設定されている場合に限られる。
C-a m		
C-a C-m	(lastmsg)	メッセージ行に表示された直前のメッセージを繰り返す。

C-a M	(monitor)	現在のウィンドウの監視をトグルする。
C-a space **C-a n** **C-a C-n**	(next)	次のウィンドウに切り替える。
C-a N	(number)	現在のウィンドウの番号(とタイトル)を表示する。
C-a backspace **C-a h** **C-a p** **C-a C-p**	(prev)	前のウィンドウに切り替える (**C-a n** の反対)。
C-a q **C-a C-q**	(xon)	現在のウィンドウに Control-q を送る。
C-a Q	(only)	現在のリージョンを残して他はすべて消す。
C-a r **C-a C-r**	(wrap)	現在のウィンドウの行折り返しに関する設定をトグルする (現在のウィンドウの自動マージンのオン/オフを切り替える)。
C-a s **C-a C-s**	(xoff)	現在のウィンドウに Control-s を送る。
C-a S	(split)	現在のリージョンを 2 つに分ける。
C-a t **C-a C-t**	(time)	システム情報を表示する。
C-a v	(version)	バージョンとコンパイルされた日時を表示する。
C-a C-v	(digraph)	二重字 (digraph) を入力する。 (訳注: 二重字とはアクセント記号等が付いた文字のこと)
C-a w **C-a C-w**	(windows)	ウィンドウのリストを表示する。
C-a W	(width)	80/132 カラムをトグルする。
C-a x **C-a C-x**	(lockscreen)	この端末をロックする。
C-a X	(remove)	現在のリージョンを破棄する。
C-a z **C-a C-z**	(suspend)	*screen* をサスペンドする。システムが BSD 形式のジョブ制御をサポートしていなければならない。
C-a Z	(reset)	仮想端末を “power-on” の値にリセットする。
C-a .	(dumptermcap)	“.termcap” ファイルを出力する。
C-a ?	(help)	キー割り当てを表示する。
C-a C-	(quit)	すべてのウィンドウを破棄し、 *screen* を終了する。
C-a :	(colon)	コマンドラインモードに入る。
C-a [**C-a C-[** **C-a esc**	(copy)	コピー/スクロールバックモードに入る。
C-a]	(paste .)	ペーストバッファの内容を現在のウィンドウの標準入力のキューに書き込む。
C-a { **C-a }**	(history)	前の (コマンド) 行をコピー&ペーストする。

C-a >	(writebuf)	ペーストバッファをファイルに書き出す。
C-a <	(readbuf)	画面交換ファイルをペーストバッファに読み込む。
C-a =	(removebuf)	**C-a <** と **C-a >** で使ったファイルを削除する。
C-a ,	(license)	*screen* の由来を表示する。これは *screen* が来た道であり、あなたが *screen* を利用できている理由である。
C-a _	(silence)	現在のウィンドウが非アクティブかどうかの監視を開始/停止する。
C-a *	(displays)	現在アタッチしているすべての画面のリストを表示する。

カスタマイズ

「ソケットディレクトリ」は、デフォルトでは $HOME/.screen または単に /tmp/screens だが、コンパイル時に選択すれば /var/run/screen にもできる。 *screen* を setuid-root でインストールする場合、システム管理者は適切な (NFS マウントではない) ソケットディレクトリを使うよう *screen* をコンパイルしなければならない。 *screen* が setuid-root でない状態で動作する場合なら、ユーザは環境変数 $SCREENDIR を使って、モード 700 の任意のディレクトリを指定できる。

screen は起動されると、2 つのファイル (“/etc/screenrc” とそのユーザのホームディレクトリの “.screenrc”) から初期化コマンドを読み込んで実行する。これらは「プログラマのデフォルト値」であり、以下のようにすれば変更できる: システム全体の screenrc として、 *screen* は環境変数 $SYSSCREENRC を検索する (この変更機能はコンパイル時に無効にできる)。ユーザ固有の screenrc ファイルは $SCREENRC, $HOME/.screenrc の順で検索される。コマンドラインオプション **-c** は、これらのユーザ用 screenrc ファイルよりも優先される。

これらのファイルに書かれたコマンドは、オプションの設定やキーへの機能の割り当て、 *screen* セッション開始時の自動的なウィンドウ生成 (複数可) に用いられる。コマンドは 1 行に 1 つ書かれ、空の行は無視される。コマンドの引き数はタブまたは空白で区切られ、シングルクォートまたはダブルクォートで括ることができる。 ‘#’ があると、その行の残りの部分はコメントとなる。ただし ‘#’ がクォートの内部にある場合は除く。判読できない行には警告が出され、無視される。コマンドでは環境変数を参照できる。記法はシェルと同様に "$VAR " または "${VAR}" の形式である。これは *screen* の古いバージョンとの非互換性の原因になっており、現在は変数代入を行わせない場合 ’$’ は ’\’ で保護しなければならない。

screen の配布パッケージには、 2 つの設定ファイル、 “etc/screenrc” と “etc/etcscreenrc” とが例として同梱されている。これらには様々なコマンドの便利な例がたくさん載っている。

カスタマイズは「オンライン」で行うこともできる。コマンドモードに入るには ‘C-a :’ を入力する。 “def” で始まるコマンドはデフォルト値を変更し、他のコマンドは現在の設定を変更する。

利用できるコマンドを以降に示す:

acladd *usernames* [*crypted-pw*]
addacl *usernames*

ユーザに現在の screen セッションへの完全なアクセスを与える。 *usernames* は 1 人のユーザ、またはコンマ区切りのユーザリストである。このコマンドは *screen* セッションにアタッチできるようにし、 ‘aclchg *usernames* +rwx "#?"’ と同じ内容を実行する。アクセス権を制限した状態でユーザを追加するには、後述の ‘aclchg’ を用いること。 2 番目のパラメータは省略可能だが、これを与えると、これは指定したユーザ (ないし複数のユーザ) の暗号化パスワードになる。 ‘addacl’ は ‘acladd’ と同機能の別名である。このコマンドはマルチユーザモードでのみ使用できる。

aclchg *usernames permbits list*
chacl *usernames permbits list*

コンマ区切りのリストで指定したユーザの権限 (permission) を変更する。 permbits は ‘r’, ‘w’, ‘x’ で表し、 ‘+’ を前に置くと権限が与えられ、 ‘-’ を置くと権限が取り上げられる。 3 番目のパラ

メータは、コマンドやウィンドウ (番号またはタイトルで指定) をコンマで区切って並べたリストである。特別なリストである '#' はウィンドウすべてを意味し、'?' はコマンドすべてを意味する。 *usernames* が '*' 1 つだけの場合、認識されているユーザすべてが影響を受ける。ユーザがあるコマンドに対する 'x' ビットを持っていると、そのコマンドを実行できる。あるウィンドウに対して 'w' ビットが設定されており、他のユーザがそのウィンドウの書き込みロックを取得していないとき、ユーザはそのウィンドウへの入力を行える。他のビットは現在は無視される。ウィンドウ 2 にいる他のユーザの書き込みロックを取り消すには、 'aclchg *username* -w+w 2' コマンドを実行する。セッションに対する読み込み専用のアクセスを許可するには、 'aclchg *username* -w "#"' コマンドを実行する。ユーザ名が *screen* に識別されると、直ちにそのユーザはセッションにアタッチでき、 (デフォルトでは) すべてのコマンドとウィンドウに対する完全な権限を与えられる。 acl コマンドや 'at' 等に対する実行許可は削除しておくべきである。さもないと、そのユーザは書き込み権限を再取得できてしまう。特殊なユーザ **nobody** に対する権限は変更できない ("su" コマンドを見よ)。 'chacl' は 'aclchg' と同義の別名である。マルチユーザモードでのみ使用できる。

acldel *username*

ユーザを *screen* のアクセス制御リストから削除する。現在アタッチされていれば、そのユーザのすべてのディスプレイはそのセッションからデタッチされる。そのユーザは再びアタッチすることはできない。マルチユーザモードでのみ使用できる。

aclgrp *username* [*groupname*]

同じアクセス権限を共有するユーザのグループを作成する。このグループの名前は、グループのリーダーのユーザ名になる。グループの各メンバーはグループリーダーに与えられた権限を継承する。すなわち、あるユーザがあるアクセスのチェックに失敗すると、続いて別のチェックがグループリーダーに対してなされる。 *groupname* に特殊な値 "none" が用いられると、ユーザはすべてのグループから削除される。 2 番目のパラメータが省略されると、ユーザのいるすべてのグループがリストされる。

aclumask [[*users*]+*bits* |[*users*]-*bits* ....]
umask [[*users*]+*bits* |[*users*]-*bits* ....]

このコマンドを呼んだ者が生成するウィンドウに対して、他のユーザが持つアクセス権限を指定する。 *users* は、なし、一人、既知のユーザ名のコンマ区切りリスト、のいずれかである。 *users* を指定しないと、現在わかっているすべてのユーザのリストを指定したことになる。 *bits* は "aclchg" コマンドで定義されたアクセス制御ビットの任意の組み合わせである。ユーザ名 "?" は特殊な意味を持ち、まだ登場していないユーザに対して、事前に任意のウィンドウに対するアクセス権限を付与するために用いる。ユーザ名 "??" は特殊な意味を持ち、まだ登場していないユーザに対して、任意のコマンドに対するアクセス権限を付与するために用いる。特殊なユーザ名である **nobody** の権限を変更することはできない ("su" コマンドを見よ)。 'umask' は 'aclumask' と同義の別名である。

activity *message*

監視状態にあるバックグラウンドウィンドウのどれかで何らかの動作があった場合、 *screen* はメッセージ行で通知を行う。この通知メッセージは "activity" コマンドで再定義できる。 *message* に含まれる '%' は、それぞれ動作が起こったウィンドウの番号に置換され、'^' は、それぞれ termcap に書かれているベルの定義 (通常は音を鳴らすベル文字) に置換される。デフォルトのメッセージは

'Activity in window %n'

である。デフォルトではすべてのウィンドウに対して監視はオフになっているが、 "monitor" コマンド(C-a M) で変更できる。

allpartial on|off

on にすると、ウィンドウが変化した時に現在のカーソル行だけがリフレッシュされる。これはすべてのウィンドウに影響し、端末ラインが遅い場合に役に立つ。各ウィンドウにおける全体/部分リフレッシュの直前の設定は、 ‘‘allpartial off’’ で復元される。これはグローバルなフラグであり、即座にすべてのウィンドウに影響を及ぼし、‘‘partial’’ の設定を変更する。これ以降に新しく生成されるウィンドウのデフォルトの再描画の動作は、これによっては変わらない。

altscreen on|off

on にすると仮想端末で「別画面 (alternate screen) 」のサポートが有効になる。これは xterm のと同様のものである。初期設定では off になっている。

at [*identifier*][**#|*|%**] *command* [*args ...*]

他の画面やウィンドウで (あたかもそこで入力したかのように) コマンドを実行する。 ‘‘at’’ はコマンドのコンテクスト (「現在のウィンドウ」や「現在の画面」の設定) を変更する。最初のパラメータでコンテクストがひとつに定まらない場合には、そのコマンドは複数回実行される。最初のパラメータが ‘*identifier**’ の形式ならば、ユーザ名に対して identifier のマッチを試みる。指定されたコマンドは、選択されたユーザ (複数可) の画面それぞれにおいて 1 度ずつ実行される。最初のパラメータが ‘*identifier%*’ の形式ならば、画面に対して identifier のマッチを試みる。画面の名前はアタッチしている tty に基づいて付けられる。プレフィックス ‘/dev/’ や ‘/dev/tty’ は identifier から省略できる。 *identifier* に ‘#’ が付いた (あるいは末尾に何も付いていない) 場合は、ウィンドウ番号とタイトルにマッチする。‘*’, ‘%’, ‘#’ 文字の前に置く識別子を省略すると、すべてのユーザ、画面、ウィンドウが選択される。影響を受ける画面 (複数のこともある) では、起こったことが短いメッセージで表示される。 ‘‘at’’ コマンドの実行者に対しては権限のチェックが行なわれ、影響される画面の所有者に対しては行なわれない。’#’ 文字の後に空白文字が置かれるとコメントの開始になるが、これは ’\’ を前置することでエスケープできる。
注意: ウィンドウに対するマッチでは、コマンドは各ウィンドウで最低 1 回は実行される。ウィンドウの内部順序を変更するようなコマンド (‘‘other’’ など) は、複数回呼び出されるかもしれない。共有ウィンドウでは、コマンドはアタッチしている画面ごとに繰り返される。 ‘‘login’’ のような、状態をトグルするコマンドを実行するときには注意すること! コマンドによっては、対象ウィンドウがディスプレイに関連づけられていないと機能しないものがある (‘‘process’’ など)。これらのコマンドは、‘‘at’’ でウィンドウを巡回させる場合には機能しないかもしれない。

attrcolor *attrib* [*attribute/color-modifier*]

このコマンドを用いると、テキストの色を変えて属性をハイライトできる。属性 *attrib* がすでに利用中の場合には、指定した属性/色修正が同時に適用される。修正指定を与えないと、現在のものを削除する。修正指定の文法については『文字エスケープ』を参照のこと。 screen は 2 つの疑似属性を理解する: ‘‘i’’ は高輝度の前景色を意味し、 ‘‘I’’ は高輝度の背景色を意味する。

例:

```
attrcolor b "R"
```

ボールドのテキストを表示するとき、色を明るい赤に変更する。

```
attrcolor u "-u b"
```

アンダーラインの代わりに青のテキストを用いる。

```
attrcolor b ".I"
```

ボールドのテキストに明るい色を使う。ほとんどの端末エミュレータでは、すでにこの指定はなされているだろう。

```
attrcolor i "+b"
```

高輝度色のテキストをボールドにもする。

autodetach on|off

ハングアップの際に、自動的にデタッチを *screen* にさせるかどうかを設定する。デタッチを行うと、実行中のプログラムはすべて保存され、 **screen -r** コマンドで再開できる。このオプションを off にすると、ハングアップシグナルによって *screen* および *screen* 内で実行されているすべてのプロセスが終了する。 autodetach はデフォルトでは on になっている。

autonuke on|off

画面クリアのシーケンスが与えられたとき、まだ端末に書き出されていない出力すべてを破棄するかどうか指定する。 ‘‘obuflimit’’ も参照のこと。

backtick *id lifespan autorefresh cmd args...*
backtick *id*

(数値の) ID が *id* となる backtick コマンドをプログラムする。このようなコマンドの出力は、文字列エスケープ ‘‘%‘’ に代入される。指定された *lifespan* は、出力が有効とみなされる期間の秒数である。この秒数が経過した後に、対応する文字列エスケープが現われた場合には、このコマンドが再び実行される。 *autorefresh* パラメータで指定する秒数が過ぎると、キャプション文字列とハードステータス文字列とが自動的にリフレッシュされる。出力の最後の行だけが代入に用いられる。
lifespan と *autorefresh* の両方のパラメータが 0 だと、その backtick プログラムはバックグラウンドに留まり、ときどき出力を生成するものとみなされる。この場合、そのコマンドは直ちに実行され、 screen はその出力の最後の行を保存する。新しい行が表示されると、 screen はハードステータスまたはキャプションをリフレッシュする。
2 番目の書式は、数値 ID が *id* の backtick コマンドを削除する。

bce [**on|off**]

背景色消去 (background-color-erase) の設定を変更する。 ‘‘bce’’ を on にすると、erase/insert/scroll/clear 操作でクリアされるすべての文字は、現在の背景色で表示されることになる。 off にするとデフォルトの背景色が用いられる。

bell_msg [*message*]

ベル文字がバックグラウンドのウィンドウに送られると、 *screen* はメッセージ行に通知を出す。この通知メッセージは、このコマンドによって再定義できる。*message* に ‘%’ があると、それらはベルを受け取ったウィンドウの番号に置換される。また ‘^G’ は termcap で定義されているベル文字 (通常は音声ベル) に置換される。デフォルトのメッセージは次の通り：

’Bell in window %n’

‘‘bell_msg’’ コマンドには空のメッセージを与えることもでき、その場合メッセージ行の出力はされなくなる (bell_msg "")。パラメータを与えないと、現在のメッセージが表示される。

bind [**-c** *class*] *key* [*command* [*args*]]

コマンドをキーに割り当てる。デフォルトでは、 *screen* に用意されているほとんどのコマンドは 1 つないし複数のキーに割り当てられており、これらは『デフォルトのキー割り当て』のセクションに示されている。例えば新しいウィンドウを生成するコマンドは ‘‘C-c’’ と ‘‘c’’ とに割り当てられている。 ‘‘bind’’ コマンドを用いると、キー割り当てを再定義したり、新しい割り当てを定義したりできる。引き数 *key* には、文字 1 つ、‘‘^x’’ 形式(‘‘C-x’’ の意味)の 2 文字からなるシーケンス、バックスラッシュの後に 8 進数を続けたもの (文字の ASCII コードを示す)、バックスラッシュの後に 2 番目の文字を続けたもの (‘‘\^’’ や ‘‘\\’’ 等) のいずれかを与える。引き数はクォートしてもよい。 *key* 以外の引き数が与えられなかった場合は、以前にこのキーに対して与えられた割り当てが削除される。 *command* 引き数には、このセクションにリストされている任意のコマンドを指定できる。

‘‘-c’’ オプションによってコマンドクラスが指定された場合には、そのキーは指定されたクラス

に割り当てられる。クラスを有効にするには “command” コマンドを用いる。コマンドクラスは、コマンドキーを複数使いたいときや、二文字以上のシーケンスを割り当てたい場合に利用できる。

指定例をいくつか示す:

```
bind ' ' windows
bind ^k
bind k
bind K kill
bind ^f screen telnet foobar
bind \033 screen -ln -t root -h 1000 9 su
```

これは、まずウィンドウのリスト表示のコマンドをスペースキーに割り当てる (通常は “C-a C-w” で呼び出されるコマンドが “C-a スペース” でも呼び出せるようになる)。続く 3 行では、デフォルトで kill が割り当てられている “C-a C-k” と “C-a k” の両方を無効にし、そして “C-a K” を kill コマンドに割り当てている。次の行は、「ウィンドウを生成してホスト foobar への TELNET 接続を行う」コマンドを “C-f” に割り当てている。最後の行は、別名 “root” を持ち、番号が 9 で、スーパーユーザのシェルを持ち、スクロールバック用のバッファのサイズが 1000 行であるような、非ログインウィンドウを生成するコマンドをエスケープキーに割り当てる。

```
bind -c demo1 0 select 10
bind -c demo1 1 select 11
bind -c demo1 2 select 12
bindkey "^B" command -c demo1
```

“C-b 0” をウィンドウ 10 の選択、“C-b 1” をウィンドウ 11 の選択、のようにする。

```
bind -c demo2 0 select 10
bind -c demo2 1 select 11
bind -c demo2 2 select 12
bind - command -c demo2
```

“C-a - 0” をウィンドウ 10 の選択、“C-a - 1” をウィンドウ 11 の選択、のようにする。

bindkey [**-d**] [**-m**] [**-a**] [[**-k**|**-t**] *string* [*cmd args*]]

このコマンドは screen の入力変換テーブルを管理する。テーブルのどれか 1 つに含まれる各エントリは、特定の文字シーケンスに出会ったときに行うべき応答を screen に指示する。このようなテーブルは 3 つ存在する。ユーザがプログラムした動作を含むテーブル、端末エミュレーションで使われるデフォルトの動作を含むテーブル、 screen のコピーモードでカーソル移動を行うためのテーブル、である。デフォルトのキー割り当てのリストは、『入力の変換』のセクションを参照すること。

-d オプションを与えると bindkey はデフォルトのテーブルを変更し、 **-m** オプションはコピーモードのテーブルを変更し、どちらのオプションも与えないとユーザテーブルが選択される。引き数 *string* は動作を割り当てる文字シーケンスである。これは固定の文字列でもよいし、 termcap のキーボード機能名でもよい (**-k** オプションで選択できる)。

VT100 端末でアプリケーションモードが有効であるとき、一部のキーは異なる文字列を送ってくることがある (例えばカーソルキー)。このようなキーは変換テーブル中にエントリを 2 つ持つ。アプリケーションモード時のエントリは **-a** オプションの指定によって選択できる。

-t オプションを指定すると、screen は文字間タイミング調整を行わない。 termcap の機能を使う場合には、タイミング調整を無効にはできない。

cmd には任意の screen のコマンドと、これに対する任意の数の引き数 *args* を指定できる。 *cmd* が省略された場合は、そのキー割り当てがテーブルから削除される。

以下にキーボード割り当ての例をいくつか示す:

```
bindkey -d
```

デフォルトのキー割当をすべて表示する。アプリケーションモードのエントリには [A] が付く。

bindkey -k k1 select 1

"F1" キーでウィンドウ 1 に切り替わるようにする。

bindkey -t foo stuff barfoo

"foo" を単語 "barfoo" の省略形とする。タイムアウトを無効にしてあるので、ユーザがゆっくり入力できる。

bindkey "\024" mapdefault

このキー割り当ては、“^T” をキー割り当て用のエスケープ文字にする。先程挙げた “stuff barfoo” の設定がされている場合、 “^Tfoo” とタイプすれば単語 “foo” が入力できる。 “^T” を挿入したい場合には、このキーを 2 回押す必要がある (つまりエスケープの割り当てをエスケープする)。

bindkey -k F1 command

F11 (F1 ではない!) キーにも screen のエスケープ機能 (^A と同じもの) を与える。

break [*duration*]

このウィンドウにブレーク信号を *duration**0.25 秒間送る。非 Posix システムでは送信時間は 1 秒単位に丸められる。シェルプロセスの場合よりは、キャラクタ型デバイスがウィンドウにアタッチされている場合に非常に役に立つ (『ウィンドウの種類』の章も参照のこと)。ブレーク信号の最大継続時間は 15 秒に制限されている。

blanker

画面のブランク機能を有効にする。まず最初に画面はクリアされる。ブランク用のプログラムが定義されていない場合は、カーソルも消える。定義されている場合はそのプログラムが起動され、出力が画面に書き出される。画面ブランク機能は、何かのキーが押された段階で終了し、そのとき読み込んだキー入力は捨てられる。
このコマンドは通常 “idle” コマンドと共に用いられる。

blankerprg [*program args*]

ブランク機能用のプログラムを定義する。引き数を与えないと、ブランク用のプログラムを無効にする。

breaktype [*tcsendbreak*|*TIOCSBRK* |*TCSBRK*]

端末デバイスに対してブレーク信号を生成する方法を、可能なものの中から選ぶ。このコマンドは現在のウィンドウにのみ影響するべきだが、現在は “defbreaktype” と同じように振舞う。これは将来は変更される予定である。 “breaktype” をパラメータ無しで呼ぶと、現在のウィンドウにおけるブレーク信号の発生方法を表示する。

bufferfile [*exchange-file*]

ペーストバッファを使った読み書きに使うファイル名を変更する。 “bufferfile” コマンドへの引き数を省略した場合には、デフォルトの設定 (“/tmp/screen-exchange”) が再び使われるようになる。以下の例は、システムのパスワードファイルを *screen* のウィンドウにペーストするものである (ペーストバッファを使う。ここにはコピーが残る):

C-a : bufferfile /etc/passwd
C-a < C-a]

C-a : bufferfile

c1 [**on**|**off**]

c1 コードの処理を変更する。“c1 on” を指定すると、 screen は 128 から 159 までの入力文字を制御機能として扱う。このような 8 ビットコードは、通常は ESC の後に対応する 7 ビットコードを続けたものと同じ意味になる。デフォルトの設定では c1 コードを処理する。これは “defc1” コマンドで変更できる。 c1 領域に利用可能な文字があるフォントを使っているユーザは、このオプションを off にするとよい。

caption always|**splitonly** [*string*]
caption string [*string*]

このコマンドはウィンドウのキャプションの表示を制御する。通常キャプションはディスプレイに複数のウィンドウが現れたときにのみ用いられるが、ここの属性を **always** に設定すると、ひとつしかウィンドウがないときでも screen はキャプションを表示する。デフォルトの設定は **splitonly** である。

二番目の形式はキャプションに用いられるテキストを変更する。 “文字エスケープ” の章にあるすべてのエスケープが利用できる。 screen がデフォルトで用いるのは ‘%3n %t’ である。

文字列を追加引数として、両方の形式を混ぜることもできる。

charset *set*

文字セットのスロット指定と文字セットのマッピングを変更する。 *set* の先頭 4 文字は、文字セット指定として扱われる。 5-6 番目の文字は 0 から 3 までの値を取り、GL/GR 文字セットのマッピングを指定する。すべての位置には ’.’ を置くことができ、その場合は現在の文字セット/マッピングを変更しない (*set* が 6 文字に満たない場合は、内部で ’.’ を追加する)。新しいウィンドウのデフォルトの文字セットは、 “encoding” コマンドが有効になっていなければ "BBBB02" になる。

現在の設定は “info” コマンドで閲覧できる。

chdir [*directory*]

screen のカレントディレクトリを指定されたディレクトリに変更する。引き数なしで呼ばれた場合には、カレントディレクトリをホームディレクトリ (環境変数 $HOME の値) にする。 “.screenrc” 内部の “screen” コマンド、 “C-a : screen ...”、 “C-a c” のいずれかによって新しく生成されるウィンドウは、これをデフォルトのディレクトリとして使用する。 chdir コマンドを使わないと、デフォルトのディレクトリは *screen* が起動されたディレクトリになる。ハードコピーとログファイルが書き出されるのは、常に「そのウィンドウの」デフォルトディレクトリであり、ウィンドウ内で動作しているプロセスのカレントディレクトリではない。このコマンドを .screenrc 内で複数回用いれば、別々のデフォルトディレクトリでウィンドウをいくつも起動することができるが、ユーザが対話的に生成したすべてのウィンドウは、最後の chdir の値の影響を受ける。

clear

現在のウィンドウをクリアし、そのイメージをスクロールバックバッファに格納する。

colon [*prefix*]

“.screenrc” のコマンドラインを入力できるようにする。 screen の動作中にキー割り当てを変えたり、特定のウィンドウの生成・設定の変更などに便利である。なお “set” キーワードは無くなってしまった。コマンドは通常現在のウィンドウに影響し、将来生成されるウィンドウのデフォルト設定にはならない。デフォルトを変更するには ’def...’ で始まるコマンド群を用いること。

これを *screen* の「Ex コマンドモード」とするなら、 “C-a esc” (コピーモード) は「Vi コマンドモード」と言えるだろう。

command [**-c** *class*]

このコマンドは screen のエスケープ文字 (^A) を打ち込んだ場合と同じ動作を行う。これが役に立つのはキー割り当ての場合だけだろう。 ‘‘-c’’ オプションが与えられると、指定したコマンドクラスを選択する。 ‘‘bind’’ と ‘‘bindkey’’ も参照すること。

compacthist [**on**|**off**]

テキストをヒストリバッファの領域へスクロールアップするときに、末尾の空行の出力を抑制するかどうか指定する。

console [**on**|**off**]

マシンコンソールのウィンドウへの出力をグラブまたはアングラブする。 注意: コンソール出力をグラブできるのは /dev/console の所有者だけである。このコマンドは、 ioctl TIOCCONS をサポートしているマシンでしか使えない。

copy

コピー/スクロールバックモードに入る。ここでは、現在のウィンドウおよびその履歴から、テキストをペーストバッファへとコピーできる。このモードでは、vi に似た「フルスクリーンエディタ」がアクティブになる:

移動キー:
h, **j**, **k**, **l** は、カーソルを上下左右に移動する。
0, ^, **$** は、行の内部で、先頭・空白でない最初の文字・空白でない最後の文字、へとカーソルを移動する。
H, **M**, **L** は、カーソルをウィンドウの最上行・中央・最下行の、それぞれ先頭に移動する。
+ および **–** は、位置を 1 行上・1 行下へ移動する。
G は指定された絶対行へ移動する (デフォルト値はバッファの末尾)。
| は指定された絶対桁へ移動する。
w, **b**, **e** は、カーソルを単語単位で移動させる。
B と **E** は、カーソルを「単語」単位で移動させる (vi と同様)。
C-u と **C-d** は、指定された行数だけ画面を上下にスクロールさせる。この際にカーソル位置は変わらない (デフォルト値は全画面の半分)。
C-b と **C-f** は、画面を 1 画面分スクロールアップ・ダウンさせる。
g はバッファの先頭へ移動する。
% は、バッファ中の指定したパーセンテージの位置へジャンプする。

注意:
.screenrc のコマンドを使って Emacs 形式の移動をさせるようなカスタマイズも できる (例: markkeys "h=^B:l=^F:$=^E")。ただし、完全な Emacs 形式のキーマップには複数個の文字からなるコードが含まれているが、これを簡単に実現することはできない。

マーキング:
コピー領域は、2 つのマークを設定することによって指定する。これらのマークの間のテキストはハイライト表示される。
スペースキーを押すと、1 番目・2 番目のマークが設定される。
Y を押すと 1 行全体が、 **y** を押すと行頭から現在の位置までがマークされる。
W は 1 単語をマークする。

繰り返し回数:
以上のコマンドはすべて、数字キーによって繰り返し回数を指定できる。
0..**9** は繰り返し回数として扱われる。
例: ‘‘C-a C-[H 10 j 5 Y’’ は、 11 行目から 15 行目までをペーストバッファにコピーする。

検索:

/ は *Vi* と同様の検索コマンドである。
? は *Vi* と同様の検索コマンドである。
C-a s は *Emacs* 形式の前方へのインクリメンタル検索である。 **C-r** は *Emacs* 形式の後方へのインクリメンタル検索である。

特殊コマンド: 一部のキーは *vi* とは異なる動作をする。 *vi* ではテキストの矩形ブロックのヤンクを行えないが、 *screen* ではできる。

c や **C** キーを押して、左端と右端をそれぞれ設定する。繰り返し回数を与えないと、どちらもデフォルトでは現在のカーソル位置となる。

例: 以下のコマンドをテキスト画面全体で試してみよ: "C-a [M 20 l SPACE c 10 l 5 j C SPACE"

これにより、カーソルはスクリーンの中央の行に移動し、そこから 20 桁左に移動し、ペーストバッファの最初のマークを行い、左端の設定を行い、5 桁右に移動し、右端を設定し、最後にペーストバッファの終わりのマークを行う。以下のコマンドを試し、コピーされるテキスト量の違いを見ること:

"C-a [M 20 l SPACE 10 l 5 j SPACE"

J は行を連結する。これは 4 つのモードのトグルとなる。それぞれのモードは、改行文字 (012) が行区切り、行区切りなし、行区切りが空白、行区切りがカンマ、である。なお "crlf on" を実行すると、改行文字の前に復帰文字を置ける。

v は ":set numbers" を使っている *vi* ユーザ向けのものである。すなわち、左マージンを 9 桁目と 1 桁目との間でトグルする。

a を二番目のスペースの前に押すと、追加モードに切り替わる。こうするとペーストバッファの内容を上書きせず、追加する。

A は追加モードへの切り替えを行い、(2 番目の) マークを設定する。

> は (2 番目の) マークをセットし、コピーモードが終了した時にペーストバッファの内容を画面交換ファイル (デフォルトでは /tmp/screen-exchange) に書き出す。
次の例は、スクロールバックバッファ全体をこのファイルにダンプする方法を示したものである: "C-A [g SPACE G $ >"

C-g は現在の行と桁に関する情報を与える。

x は最初のマークと現在のカーソル位置を交換する。これを用いると、既に配置したマークを調整できる。

@ は何もしない。コピーモードの終了も行わない。

ここで説明していないすべてのキーは、コピーモードを終了させる。

copy_reg [*key*]

もう無い。代わりに "readreg" を使うこと。

crlf [**on**|**off**]

'C-a [' コマンドを用いたテキスト領域のコピーに影響を与える。これを 'on' に設定すると、各行は 2 文字からなるシーケンス 'CR' - 'LF' によって分割される。'off' だと 'LF' だけが使われる。

debug on|off

実行時デバッグのオン/オフを行う。 *screen* にオプション -DDEBUG を付けてコンパイルするとデバッグが使えるようになり、デフォルトでオンになる。このコマンドが影響を与えるのは、メインの "SCREEN" のプロセスからのデバッグ出力だけである。アタッチしたプロセスのデバッグ出力は、一度無効にすると再度有効にすることはできない。

defc1 on|off

c1 コマンドとほぼ同じであるが、新しいウィンドウに対するデフォルト設定を変える点が異なる。初期設定は 'on' である。

defautonuke on|off

autonuke コマンドとほぼ同じであるが、新しいディスプレイに対するデフォルト設定を変える点が異なる。初期設定は 'off' である。端末のタイプに対する依存性を持たせたい場合は、特殊な端末機能である 'AN' を用いると良い。

defbce on|off

bce コマンドとほぼ同じであるが、新しいディスプレイに対するデフォルト設定を変える点が異なる。初期設定は 'off' である。

defbreaktype

端末デバイスに対してブレーク信号を生成する方法を、可能なものの中から選ぶ。 *tcsendbreak* および *TIOCSBRK* にすることを勧める。三番目の *TCSBRK* は、ブレークの間 *screen* セッションを完全にブロックしてしまうが、長いブレークを発生させるにはこれしかないこともある。 *tcsendbreak* や *TIOCSBRK* では、長いブレークの際にスパイクが出てしまう (例えば毎秒 4 回など) ことがある (そうでないこともある)。これはシステムに依存するだけでなく、シリアルボードドライバによっても異なる。 "defbreaktype" をパラメータ無しで呼ぶと、現在の設定を表示する。

defcharset [*set*]

charset コマンドと似ているが、新しいウィンドウに対するデフォルト設定を変える点が異なる。引き数なしで呼び出された場合には、現在のデフォルト値を表示する。

defescape *xy*

デフォルトのコマンド文字を設定する。これは "escape" とほぼ同じであるが、マルチユーザのセッションでしか役に立たない点が異なる。マルチユーザセッションでは、"escape" は呼び出したユーザのコマンド文字を変更する。一方 "defescape" は、これ以後に追加されるユーザのコマンド文字列のデフォルト値を変更する。

defflow on|off|auto [**interrupt**]

flow コマンドとほぼ同じであるが、新しいウィンドウに対するデフォルト設定を変える点が異なる。初期設定は 'auto' である。 "defflow auto interrupt" と指定すると、コマンドラインオプションに **–fa** と **–i** とを与えた場合と同じになる。

defgr on|off

gr コマンドとほぼ同じであるが、新しいウィンドウに対するデフォルト設定を変える点が異なる。初期設定は 'off' である。

defhstatus [*status*]

以降作られる新しいウィンドウのハードステータス行を *status* にする。このコマンドは、すべてのウィンドウのハードステータス行に、ウィンドウ番号やタイトル等を表示させるのに便利であ

る。 *status* にはウィンドウメッセージの場合と同様の指定機能が利用できるが、このエスケープ文字は '%' ではなく '^E' (8 進値で 005) である。こうなっているのは、プログラムの生成したハードステータス行を、間違って解釈しないようにするためである。パラメータ *status* を省略すると、現在のデフォルト文字列が表示される。デフォルトでは、新しいウィンドウのハードステータス行は空である。

defencoding *enc*

encoding コマンドとほぼ同じであるが、新しいウィンドウに対するデフォルト設定を変える点が異なる。初期設定では、エンコーディングは端末から取得する。

deflog on|off

log コマンドとほぼ同じであるが、新しいウィンドウに対するデフォルト設定を変える点が異なる。初期設定は 'off' である。

deflogin on|off

login コマンドとほぼ同じであるが、新しいウィンドウに対するデフォルト設定を変える点が異なる。配布状態での初期設定は 'on' である (config.h.in を見よ)。

defmode *mode*

新しく割り当てられる仮想 tty のモードを *mode* に設定する。 *mode* は 8 進数の値である。 ''defmode'' コマンドが与えられていない場合は、モード 0622 が使われる。

defmonitor on|off

monitor コマンドとほぼ同じであるが、新しいウィンドウに対するデフォルトの設定を変える点が異なる。初期設定は 'off' である。

defnonblock on|off|*numsecs*

nonblock コマンドとほぼ同じであるが、新しいウィンドウに対するデフォルトの設定を変える点が異なる。初期設定は 'off' である。

defobuflimit *limit*

obuflimit コマンドとほぼ同じであるが、新しいディスプレイに対するデフォルト設定を変える点が異なる。初期設定は 256 バイトである。端末の種類への依存性を持たせたい場合には、特殊な端末機能である 'OL' が利用できる。

defscrollback *num*

scrollback コマンドとほぼ同じであるが、新しいウィンドウに対するデフォルトの設定を変える点が異なる。初期設定値は 100 である。

defshell *command*

shell コマンドの同義語である。そちらを参照のこと。

defsilence on|off

silence コマンドとほぼ同じであるが、新しいウィンドウに対するデフォルトの設定を変える点が異なる。初期設定値は 'off' である。

defslowpaste *msec"*

slowpaste コマンドとほぼ同じであるが、新しいウィンドウに対するデフォルトの設定を変える点が異なる。初期設定値は 0 ミリ秒であり、これは 'off' の意味になる。

defutf8 on|off

utf8 コマンドとほぼ同じであるが、新しいウィンドウに対するデフォルトの設定を変える点が異なる。 screen の起動時に ‘‘-U’’ を指定した場合の初期設定値は ‘on’ になり、それ以外の場合の初期設定値は ‘off’ になる。

defwrap on|off

wrap コマンドとほぼ同じであるが、新しいウィンドウに対するデフォルト設定を変える点が異なる。初期設定では行の折り返しは有効であり、‘‘wrap’’ コマンド (‘‘C-a r’’) や "C-a : wrap on|off" を使うとトグルできる。

defwritelock on|off|auto

writelock コマンドとほぼ同じであるが、新しいウィンドウに対するデフォルト設定を変える点が異なる。初期状態では、書き込みロックは無効になる。

defzombie [*keys*]

zombie コマンドと同じである。現在はどちらもデフォルト値を変更する。 **zombie** コマンドの説明を参照すること。

detach [**-h**]

screen のセッションをデタッチする (セッションを端末から切り離し、バックグラウンドに回す)。これにより、ユーザは *screen* を起動したシェルに戻る。デタッチされた *screen* は、 **–r** オプションを付けて *screen* を実行すれば復元できる (『コマンドラインオプション』を参照)。 **–h** オプションを指定すると、screen は直ちに端末との接続を閉じる (‘‘hangup’’)。

dinfo

screen があなたの端末について想定している内容を表示する。色付けや文字セット指定が効かない原因を調査したい場合に便利。

displays

現在接続されているユーザのフロントエンド (ディスプレイ) を表にして示す。これはマルチユーザセッションにおいて特に便利。

digraph [*preset*]

このコマンドはユーザに二重字 (digraph sequence) を入力させる。この次に入力された 2 文字は、組み込みのテーブルで検索され、得られた文字が入力ストリームに挿入される。例えばユーザが ’a"’ を入力すると、ウムラウト付きの a が挿入される。最初に入力された文字が 0 (ゼロ)ならば、 *screen* はこれに続く文字 (3 文字まで) を文字としてではなく 8 進数の値として扱う。 (省略可能な) 引き数 *preset* はユーザの入力として扱われるので、これを利用すると「ウムラウト」キーが作れる。例えば "bindkey ^K digraph ’"’" というコマンドを使うと、 CTRL-K a という入力によってウムラウト付きの a を生成できる。

dumptermcap

現在アクティブなウィンドウに対して最適化された仮想端末用の termcap エントリを、ユーザの ‘‘$HOME/.screen’’ ディレクトリ (あるいは *screen* がソケットを格納している場所のどこか。詳しくは後述の『ファイル』セクションを参照) に書き出す。この termcap エントリは、 *screen* が各ウィンドウに対して設定する環境変数 $TERMCAP の値と同じである。 terminfo ベースのシステムでは、 *captoinfo* のような変換プログラムを実行し、 *tic* を使ってそのエントリをコンパイルする必要があるだろう。

echo [**-n**] *message*

echo コマンドを使うと、「今日のメッセージ」で *screen* ユーザに嫌がらせができる。通常はシステムグローバルの /etc/screenrc に設定する。 ‘‘-n’’ を使うと行送りを抑制できる。‘‘sleep’’ も参照すること。 echo は環境変数をオンラインで調べる時にも便利である。

encoding *enc* [*enc*]

入出力の解釈方法を *screen* に伝える。最初の引き数はカレントウィンドウのエンコーディングを設定する。各ウィンドウは別々のエンコーディングをエミュレートできる。二番目のパラメータは省略可能であるが、これを指定すると接続している端末のエンコーディングを上書き設定する。 screen はエンコーディングの検知にロケール設定を利用するので、これが必要になることは無いはずである。端末のエンコーディングを端末のタイプによって変更する方法としては、termcap の ‘‘KJ’’ エントリを用いる方法もある。

サポートされているエンコーディングは次の通り: eucJP, SJIS, eucKR, eucCN, Big5, GBK, KOI8-R, CP1251, UTF-8, ISO8859-2, ISO8859-3, ISO8859-4, ISO8859-5, ISO8859-6, ISO8859-7, ISO8859-8, ISO8859-9, ISO8859-10, ISO8859-15, jis

‘‘defencoding’’ も参照のこと。これは新規ウィンドウに対するデフォルト値を変更する。

escape *xy*

コマンド文字を *x* に設定し、 (‘‘meta’’ コマンドを呼んで) リテラルコマンド文字を生成する文字を *y* に設定する (−e オプションと同様)。各引き数には、文字 1 つ、 ‘‘^x’’ 形式 (‘‘C-x’’ の意味) の 2 文字からなるシーケンス、バックスラッシュの後に 8 進数を続けたもの (その文字の ASCII コードを指定する)、バックスラッシュの後に 2 番目の文字を続けたもの (‘‘\^’’ や ‘‘\\’’ など)、のいずれかを指定できる。デフォルト値は ‘‘^Aa’’ である。

eval *command1* [*command2 ...*]

各引き数をパースして、別々のコマンドとして実行する。

exec [[*fdpat*] *newcommand* [*args ...*]]

実行パス *newcommand* およびその引き数 (省略可) で指定された unix サブプロセスを現在のウィンドウで実行する。 newcommand と標準入力/標準出力/標準エラー出力の間のデータフロー、もともとこのウィンドウで開始されたプロセス (これを「アプリケーションプロセス」と呼ぼう)、screen そのもの (ウィンドウ)、の三つは、ファイルディスクリプタパターン fdpat で制御する。このパターンは基本的に 3 つの文字からなるシーケンスで、 newcommand の標準入力、標準出力、標準エラー出力を表す。ドット (.) はファイルディスクリプタを *screen* に接続する。感嘆符 (!) はファイルディスクリプタを既に実行中のプロセスに接続する。コロン (:) は両方を結合させる。ユーザ入力は newcommand に送られる。ただし newcommand が「アプリケーションプロセス」の出力を受けていたり (fdpats の最初の文字が ‘!’ または ‘:’)、パイプ (|) が fdpat の最後に (4 番目の文字として) 追加されている場合は別である。

引き数を付けずに ‘exec’ を実行すると、現在実行中のサブプロセスの名前と引き数とがそのウィンドウに表示される。各ウィンドウで一度に動作できるサブプロセスはひとつだけである。

サブプロセスの実行中には、‘kill’ コマンドはウィンドウのプロセスではなくサブプロセスに影響する。

この 21 通りもある組み合わせを描いた難解な図については、 PostScript 形式のファイル ‘doc/fdpat.ps’ を参照のこと。それぞれの図には、newcommand の三つのファイルデスクリプタを表す数字 2,1,0 が書かれている。‘W’ の印が付いた四角は普通の pty で、このスレーブ側にアプリケーションプロセスがある。 ‘P’ の印が付いた四角はセカンダリの pty で、このマスター側に *screen* が属することになる。

省略形:
‘exec’ と fdpat とコマンドの間の空白は省略できる。末尾のドットと、ドットだけからなる fdpat は省略できる。単に ‘|’ と書くとパターン ‘!..|’ と同義になる。この場合には exec を省略でき、常

に ‘!’ で置き換え可能である。

実行例:

```
exec ... /bin/sh
exec /bin/sh
!/bin/sh
```

同じウィンドウで、元のシェルを実行させたまま別のシェルを生成する。両方のシェルの出力が表示され、ユーザの入力は新しい /bin/sh に送られる。

```
exec !.. stty 19200
exec ! stty 19200
!!stty 19200
```

ウィンドウの tty の速度を設定する。 stty コマンドが標準出力を操作する場合には、もうひとつ ‘!’ を追加すること。

```
exec !..| less
|less
```

これはウィンドウ出力にページャを追加する。ユーザ制御をページャ越しに行い、しかしページャにウィンドウのプロセスから入力を取得させるには、この特殊文字 ‘|’ が必要である。これが動作するのは、標準入力が tty でないとき、 *less* が標準エラー出力を監視する (‘|’ が無ければ *screen* が想定しない動作) からである。バージョンが 177 より新しい *less* はここではまともに動作しない。この場合でも古き良き *pg* なら動作する。

```
!:sed -n s/.*Error.*/\007/p
```

ウィンドウの出力をユーザと sed コマンドの両方に送る。sed は、 *screen* の表示するウィンドウ出力にベル文字 (8 進値で 007) を追加挿入する。これを実行すると、このウィンドウに "Error" という文字列が現われると必ず "Bell in window x" というメッセージが出力される。

fit

ウィンドウのサイズを現在のリージョンのサイズにする。このコマンドが必要なのは、screen はウィンドウが複数表示されている場合には自動的なウィンドウサイズ調整を行わないためである。

flow [**on**|**off**|**auto**]

このウィンドウのフロー制御モードを設定する。パラメータを与えないと、現在のウィンドウのフロー制御設定を "automatic", "on", "off" の順に切り替える。詳細および注意点については、本ドキュメントで後述する『フロー制御』での議論を参照すること。これは将来のリリースでは変更されることになっている。デフォルトでは ‘defflow’ に設定されている。

focus [**up**|**down**|**top**|**bottom**]

入力フォーカスを次のリージョンに移す。これは循環的に行われるので、一番下のリージョンの次には一番上のリージョンが選択される。サブコマンドを省略したときのデフォルトの動作は ‘down’ である。‘up’ とすると逆順に回り、‘top’ では一番上の、‘bottom’ では一番下のリージョンに移動する。便利なキーバインディングを以下に示す (j や k は vi 流):

```
bind j focus down
bind k focus up
bind t focus top
bind b focus bottom
```

gr [**on**|**off**]

GR 文字集合のオン/オフを切り替える。 screen は、8 ビット目が立っている入力文字を見つけると、必ず GR スロットに格納されている文字集合を使用し、 8 番目のビットを取り除いて文字を出力する。デフォルトでは (‘‘defgr’’ も参照) GR 切り替えは処理しない (処理してしまうと

ISO8859-1 文字集合が使えないためである)。

hardcopy [**-h**] [*file*]

現在表示されている画面イメージを Ifile に書き込む。ファイル名を指定しないと、デフォルトディレクトリの *hardcopy.n* というファイルに書き込む (*n* はカレントウィンドウの番号)。ファイルが存在する場合には、追加か上書きのどちらかが行われる。詳しくは次項を参照。**-h** オプションを与えると、スクロールバックバッファの内容も書き込まれる。

hardcopy_append on|off

"on" に設定されていると、 *screen* はコマンド “C-a h” で生成される "hardcopy.n" ファイルに対して追加書き込みを行う。 "off" の場合には、このファイルは毎回上書きされる。デフォルト値は ‘off’ である。

hardcopydir *directory*

ハードコピーファイルが置かれるディレクトリを定義する。設定されていない場合には、ハードコピーは *screen* のカレントワーキングディレクトリにダンプされる。

hardstatus [**on**|**off**]
hardstatus [**always**]**lastline**|**message**|**ignore** [*string*]
hardstatus string [*string*]

端末のハードステータス行の利用とエミュレーションに関する設定をする。最初の形式では、 *screen* がハードステータス行をメッセージ表示に用いるかどうかをトグルする。このフラグが "off" ならば、メッセージは表示行に反転ビデオモードでオーバーレイ表示される。デフォルトの設定は ‘on’ である。

二番目の形式は、ハードステータス行が端末に無い (つまり termcap/terminfo の "hs", "ts", "fs", "ds" といった機能が設定されていない) 場合にどうするかについて *screen* に指示する。“lastline” を指定すると、 *screen* は画面の一番下の行を反転させてハードステータスとして用いる。 “message” を指定すると *screen* のメッセージ機構が用いられ、“ignore” とするとハードステータスは一切表示されなくなる。これらの指定の前に “always” をつける (例えば “alwayslastline”) と、 *screen* は端末がハードステータス行をサポートしている場合でもその指定を用いる。

三番目の形式はハードステータス行の内容を指定する。 (“ESC]0;<string>^G” や “ESC_<string>ESC\” によって設定できる)。この文字列は好きなように設定でき、『文字エスケープ』の章にあるエスケープも利用できる。引き数 *string* を省略すると、現在の文字列が表示される。

二番目の形式に *string* 引き数を付加することにより、三番目の形式の動作を同時に指定することもできる。

height [**-w**|**-d**] [*lines* [*cols*]]

ディスプレイの高さを指定された行数に設定する。引き数を与えないと、24 行表示と 42 行表示の間をトグルさせる。同時に指定すれば幅も変更できる。 **-w** オプションは、ディスプレイのサイズはそのままで、ウィンドウのサイズだけを設定したい場合に指定する。 **-d** はその逆である。

help [**-c** *class*]

本当の意味でのオンラインヘルプとは言えないが、キー割り当てをすべて示したヘルプ画面を表示する。最初のページでは、内部コマンドとその現在のキー割り当てがリスト表示される。次のページにはカスタムコマンドが、キーごとに 1 つのコマンドの形式で表示される。各ページを読み終えたらスペースキーを押せば次のページが見られ、リターンキーを押せば即座にヘルプが終了する。他の文字はすべて無視される。“-c” オプションを与えると、指定したコマンドクラスに結びつけられているすべてのコマンドを表示する。『デフォルトのキー割り当て』の節も参照すること。

history

通常ユーザはシェルを使って作業していると、以前のコマンドに簡単にアクセスできる。例えば csh には直前に実行したコマンドを繰り返す "!!" というコマンドがある。 *screen* にも、「 ~ で始まるコマンド」を再呼び出しするための原始的な方法が用意されている。そのコマンドの最初の文字を打ち込み、その後に 'C-a {' を打つと、 *screen* はカーソルの左にある「プロンプト文字」にマッチする以前の行を見つけようとする。この行はウィンドウの入力キューにペーストされる。これにより、 (可視ウィンドウとそのスクロールバッファからなる) 単純なコマンド履歴が使えることになる。

hstatus *status*

ウィンドウのハードステータス行を文字列 *status* に変更する。

idle [*timeout* [*cmd args*]]

指定した秒数の間作業がない時に実行されるコマンドを設定する。普通ここにはスクリーンセーバを生成する "blanker" コマンドを指定することになるだろうが、screen コマンドならなんでも指定できる。コマンドを何も指定しないと、タイムアウトだけが設定される。 timeout に 0 を指定する (あるいは **off**を指定する) とタイマを無効にする。一切引き数を与えないと、現在の設定を表示する。

ignorecase [**on**|**off**]

検索時に大文字小文字を区別しないようにする。デフォルトは 'off'。

info

メッセージ行を使って現在のウィンドウに関するいくつかの情報を表示する： "(カラム,行)" 形式のカーソル位置 ("(1,1)" から始まる)、端末の幅と高さ、およびスクロールバッファの大きさ (行数)。例えば "(80,24)+50" のようになる。ウィンドウの XON/XOFF フローコントロールの現在の状態は、次のように表示される (『フロー制御』の章も参照) :

+flow	自動フローコントロール、現在は on。
−flow	自動フローコントロール、現在は off。
+(+)flow	フローコントロール有効。自動コントロールと一致。
−(+)flow	フローコントロール無効。自動コントロールとは不一致。
+(−)flow	フローコントロール有効。自動コントロールとは不一致。
−(−)flow	フローコントロール無効。自動コントロールと一致。

現在の折り返しモードも表示される ('+wrap は有効、'−wrap' は無効)。
'ins', 'org', 'app', 'log', 'mon', 'nored' の各フラグは、それぞれウィンドウがインサートモード、オリジンモード、アプリケーションキーパッドモード、出力ログあり、アクティビティ監視中、部分再描画有効、の状態のときに表示される。

現在有効な文字集合 (*G0*, *G1*, *G2*, *G3* のいずれか) と、現在 *G0* から *G3* に指定されている端末の文字集合 (角括弧で括られている) が表示される。ウィンドウが UTF-8 モードのときは、これらの代わりに "UTF-8" が表示される。

ウィンドウのタイプに応じて、追加モードがステータス行の最下行に表示される (『ウィンドウの種類』の章も見よ)。
端末エミュレータのステートマシンがデフォルトの状態にないときは、 info 行の先頭には、まず現在の状態を示す文字列が表示される。
システムの情報を見るには "time" コマンドを用いること。

ins_reg [*key*]

もう存在しない。代わりに "paste" を用いること。

kill

現在のウィンドウを kill する。
実行中の 'exec' コマンドがあると、これが kill される。それ以外の場合には、そのウィンドウで動作中のプロセス (シェル) が HANGUP 信号を受け取り、そのウィンドウの関連データは削除され、 *screen* (あなたの画面) は別のウィンドウへ切り替わる。最後のウィンドウが破棄された時には、 *screen* は終了する。kill の後には、 *screen* は直前に表示していたウィンドウに切り替わる。
注意: *Emacs* ユーザは、行を消すときに、このコマンドのことを念頭に置いておくように。 ''C-a'' を *screen* のエスケープキーとして用いない、あるいは kill を ''C-a K'' に割り当てないことをお勧めする。

lastmsg

メッセージ/ステータス行の最後の内容を再表示する。これが便利なのは、メッセージが表示されたときにちょうど文字を入力していたような場合である。 (端末にハードウェアステータス行が無ければ) キーを押すとメッセージは消えてしまうからである。この微調整には ''msgwait'' コマンドや ''msgminwait'' コマンドを参照のこと。

license

免責事項のページを表示する。これはオプション無しで *screen* を起動すると必ず実行される (たいていこれで十分だろう)。 ''startup_message'' コマンドも参照のこと。

lockscreen

このディスプレイをロックする。スクリーンロックのプログラム (/local/bin/lck か /usr/bin/lock、あるいは使えるものがなければ組み込みのもの) を呼び出す。 screen はこのプログラムが終了するまで一切のコマンドキーを受け付けない。ウィンドウは「デタッチされた」状態となるので、各ウィンドウで実行中のプログラムは継続して動作する。スクリーンロックプログラムは、環境変数 $LOCKPRG で変更でき (これは *screen* を実行するシェルで設定されていなければならない)、ユーザの UID と GID で実行される。
注意: 他のシェルをロックせず、 *screen* にパスワードを与えていないと、このロックには意味がなく、ロックされていないシェルから簡単に再アタッチできてしまう。この機能は 'lockterminal' と呼ぶ方が良いかもしれない。

log [**on**|**off**]

現在のウィンドウからの出力を、ファイル ''screenlog.*n*'' へ書き込む動作を開始/終了させる。このファイルはウィンドウのデフォルトディレクトリにあり、 *n* は現在のウィンドウの番号である。このファイル名は 'logfile' コマンドで変更できる。パラメータを与えなかった場合には、ログ出力の状態がトグルされる。前のセッションログのファイルがあれば、セッションログはこのファイルに追加される。現在の内容とスクロールバック履歴の内容はセッションログには含まれない。デフォルト値は 'off' である。

logfile *filename*
logfile flush *secs*

ログファイルにつく名前を定義する。デフォルト値は ''screenlog.%n'' である。二番目の書式は、*screen* がログファイルバッファをファイルシステムにフラッシュするまでの待ち時間 (秒) を変更する。デフォルトの値は 10 秒である。

login [**on**|**off**]

utmp データベースファイル中の、現在のウィンドウのエントリを追加・削除する。これは、ウィンドウが「ログインされている」かどうかを制御する。パラメータが与えられないと、そのウィンドウのログイン状態がトグルされる。このようなトグルに加え、「ログイン」や「ログアウト」のキーがあると便利である。例えば 'bind I login on' と 'bind O login off' とすれば、これらの機能を C-a I と C-a O に割り当てできる。 *screen* が root に suid されて動作する場合は、デフォルトの設定 (config.h.in にある) は ''on'' のはずである。新しいウィンドウのデフォルトのログイ

ン状態を変えるには ‘‘deflogin’’ コマンドを用いること。これら二つのコマンドは、 *screen* のコンパイル時に utmp のサポートを組み込んである場合にしか存在しない。

logtstamp [**on**|**off**]
logtstamp after [*secs*]
logtstamp string [*string*]

このコマンドは *screen* のログファイルタイムスタンプ機能を制御する。タイムスタンプを ‘‘on’’ にすると、アクティビティがないまま 2 分間が過ぎたときに、 *screen* は現在時刻を含む文字列をログファイルに追加する。さらに 2 分が経過した後に出力がなされると、この出力の再開を記録する二番目のタイムスタンプが追加される。このタイムアウト時間は、二番目の形式でこのコマンドを実行すると変更できる。三番目の形式は、タイムスタンプ文字列をカスタマイズする (デフォルトは ‘-- %n:%t -- time-stamp -- %M/%d/%y %c:%s --\n’)。

mapdefault

次の入力文字を、デフォルトのキー割り当てテーブルでのみ検索するよう *screen* に指示する。 ‘‘bindkey’’ も参照すること。

mapnotnext

mapdefault と似ているが、デフォルトのキー割り当てテーブルも検索しない。

maptimeout [*timo*]

入力シーケンス検出の文字間タイマのタイムアウト値を *timo* ミリ秒に設定する。引き数を与えずに maptimeout を実行すると、現在の設定を表示する。 ‘‘bindkey’’ も参照すること。

markkeys *string*

これはコピー/履歴モードで使われるキーマップを変更する方法である。ここで string は *old-char=newchar* のペアからなり、ペア間の区切りは ‘:’ である。例: ‘‘B=^B:F=^F’’ を string に指定すると、 ‘C-b’ と ‘C-f’ の各キーは vi 形式の割り当て (ページ全体のスクロールアップ/ダウン) に変更される。これらは、実は ‘B’ と ‘F’ に対するデフォルトの割り当てであるが。コマンド ‘‘markkeys h=^B:l=^F:$=^E’’ は、このモードを emacs 形式の割り当てに設定する。端末が文字を送るタイプのもので、これによってコピーモードが終了してしまうような場合、このコマンドを使って、それらの文字に何もさせないようにすると良い。 no-op 文字は ‘@’ で、 ‘‘markkeys @=L=H’’ のようにすれば、 ‘H’ や ‘L’ コマンドは以後使えなくなる。この例で示したように、ひとつの機能に複数のキーを (一行で) 割り当てできる。

maxwin *num*

screen の生成できる最大ウィンドウ数を設定する。既に存在しているウィンドウには影響しない。この数は減らすことしかできない。

meta

コマンド文字 (C-a) を現在のウィンドウの入力ストリームに挿入する。

monitor [**on**|**off**]

ウィンドウのアクティビティ監視をトグルする。監視が有効で、かつその影響を受けるウィンドウがバックグラウンドに切り替えられている場合、最初の出力の際にアクティビティがあった旨ステータス行にメッセージが表示され、ウィンドウ状態の表示において、そのウィンドウに ‘@’ 印が付けられる。初期状態では、すべてのウィンドウで監視はオフになっている。

msgminwait *sec*

あるメッセージが表示されている時に、新しい (次の) メッセージを表示するまでに *screen* が待つ時間を定義する。デフォルト値は 1 秒である。

msgwait *sec*

screen が他の動作の影響を受けない場合に、メッセージが表示される時間を定義する。デフォルト値は 5 秒である。

multiuser on|off

シングルユーザモードとマルチユーザモードの切り替えを行う。 *screen* の標準動作はシングルユーザモードである。マルチユーザモードでは、コマンド 'acladd', 'aclchg', 'acldel' を使って、現在の *screen* セッションに対する他のユーザのアクセスを許可・禁止できる。

nethack on|off

screen が使うエラーメッセージの路線を変える。 ''nethack'' というゲームに親しんでいるユーザなら、このゲームスタイルのメッセージを楽しめるかもしれない。これらのメッセージは事実を少々曲げているけれども、読むにはずっと面白い。どうせ標準のメッセージだってそんなに分かりやすいわけではないし。
このオプションが利用できるのは、 *screen* のコンパイル時に NETHACK フラグが定義されている場合だけである。そして、デフォルトの設定は環境変数 $NETHACKOPTIONS があるかどうかで決まる。

next

次のウィンドウに切り替える。このコマンドを繰り返し用いると、ウィンドウを循環的に切り替えできる。

nonblock [**on**|**off**|*numsecs*]

出力の受け付けを停止したユーザインターフェース (ディスプレイ) をどのように扱うかを screen に指示する。このようなことが起こるのは、ユーザが ^S を押したときや、 TCP またはモデムの接続が切れたが hangup を受信していないような場合である。 nonblock を **off** にしておくと (こちらがデフォルト)、 screen はディスプレイが出力を受理できるようになるまで待つ。 nonblock が **on** だと、screen はタイムアウトになるまで待つ (**on** では 1s として扱われる)。待ってもディスプレイが文字を受け付けないと、 screen はディスプレイがブロックされていると判断し、そのディスプレイへの文字送信を止める。その後ディスプレイが文字の受信を再開すると、 screen はディスプレイのブロックを止め、更新されたウィンドウの内容を再描画する。

number [*n*]

現在のウィンドウ番号を変更する。指定した番号 *n* が既に他のウィンドウに使われている場合には、 2 つのウィンドウは番号を交換する。引き数が指定されていない場合は、現在のウィンドウ番号 (とタイトル) が表示される。

obuflimit [*limit*]

指定した上限値より多いバイト数が出力バッファに含まれている場合、このウィンドウからはそれ以上のデータは読み出せない。このデフォルト値は 256 である。 (xterm のように) 高速なディスプレイを使っている場合には、もう少し大きな値を設定しても良いだろう。引き数を指定しないと、現在の設定が表示される。

only

現在のものを除き、すべてのリージョンを kill する。

other

前に表示されていたウィンドウに切り替える。そのウィンドウがもう存在していなければ、*other* は *next* と同じ動作をする。

partial on|off

現在のウィンドウに切り替えた後、(*redisplay* を使ったように) 画面をリフレッシュするかどうかを定義する。このコマンドが効果を及ぼすのは現在のウィンドウだけである。すぐにウィンドウすべてに効果を及ぼすには、 *allpartial* コマンドを用いること。デフォルト値は当然 'off' である。現在 *defpartial* コマンドはないので、このデフォルト値は固定である。

password [*crypted_pw*]

".screenrc" ファイル中に暗号化されたパスワードを置くことで、デタッチされたセッションを誰かが再開しようとした際に、 *screen* にこれを用いた認証を行わせる。これは特権を持つプログラムを *screen* 下で動作させているときに、自分の UID を偽装した他のユーザ (つまりスーパーユーザの誰か) によるそのセッションの再アタッチを防ぎたい場合に便利である。暗号化されたパスワードが指定されていないと、 *screen* は 2 度パスワードの入力を求め、これを暗号化したものをペーストバッファに置く。デフォルト値は 'none' であり、パスワードの確認は無効になっている。

paste [*registers* [*dest_reg*]]

指定されたレジスタの (連結された) 内容を、現在のウィンドウの標準入力のキューに書き込む。レジスタ '.' はペーストバッファとして扱われる。パラメータをひとつも与えないと、ペーストを行うレジスタ (1 つ) をユーザに問い合わせる。ペーストバッファには *copy*, *history*, *readbuf* コマンドで書き込める。他のレジスタには *register*, *readreg*, *paste* コマンドで書き込める。 2 番目の引き数付きで *paste* が呼び出されると、指定されたレジスタの内容はそのウィンドウではなく、指定された出力先のレジスタにペーストされる。 2 番目の引き数として '.' を用いると、そのディスプレイのペーストバッファが出力先となる。 "paste" は様々な種類のリソースを使うことに注意: 2 番目の引き数が指定されていれば、カレントウィンドウは必要ない。入力元指定に (ペーストバッファではなく) レジスタしか含まれなければ、カレントディスプレイ (アタッチされている端末) は必要ない。レジスタはグローバルなリソースだからである。ペーストバッファはユーザ 1 人に 1 つずつしか存在しない。

pastefont [**on|off**]

フォント情報をペーストバッファに含めるよう screen に指示する。デフォルトではこれは行われない。このコマンドが特に役立つのは、漢字のように文字フォントが複数個ある場合である。

pow_break

ウィンドウの端末ラインを再オープンし、ブレーク信号を送る。 'break' を参照すること。

pow_detach

強制デタッチ。基本的には *detach* と同じだが、 *screen* の親プロセスにも HANGUP シグナルを送る。注意: *screen* がログインシェルから実行されているときには、これを行うとログアウトする。

pow_detach_msg [*message*]

「強制デタッチ」を行うときに、ここで指定した *message* が出力される。これはログアウトメッセージの代わりや、ボーレートのリセットなどに使える。

prev

現在のウィンドウの次に小さい番号を持つウィンドウに切り替える。このコマンドを繰り返し用いると、ウィンドウを循環できる。

printcmd [*cmd*]

cmd が空文字列でない場合、ANSI 印刷シーケンス **ESC [5 i** が検出されても screen は端末機能 ‘‘po/pf’’ を使わず、その出力を *cmd* にパイプする。*cmd* には通常、 ‘‘lpr’’ や ‘‘’cat > /tmp/scr-print’’’ のようなコマンドを与える。コマンド指定なしで **printcmd** を実行すると、現在の設定が表示される。ANSI シーケンス **ESC ** は印刷を終わらせ、パイプを閉じる。
警告: このコマンドは注意して使うこと! 他のユーザが端末への書き込み権限を持っていると、印刷コマンドを勝手に起動できてしまう。

process [*key*]

指定されたレジスタの内容を *screen* の入力キューに入れる。引き数を与えないと、ユーザにレジスタ名を尋ねるプロンプトを出す。テキストはユーザがキーボードから入力したかのように解析される。このコマンドを用いると、複数の動作を 1 つのキーに割り当てできる。

quit

すべてのウィンドウを kill し、 *screen* を終了する。VT100 形式の端末では、C-4 キーと C-\ キーは同一であることに注意。このため、デフォルトの割り当ては危険である: ウィンドウ番号 4 を選択するときに、C-a C-4 と入力しないように注意すること。キー割り当てを削除するには、空の bind コマンドを使えばよい (‘‘bind ’^\’’’ など)。

readbuf [**-e** *encoding*] [*filename*]

指定したファイルの内容をペーストバッファに読み込む。ファイルのエンコーディングを **-e** オプションで screen に知らせることもできる。ファイルを指定しないと、画面交換ファイルのファイル名が用いられる。 ‘‘bufferfile’’ コマンドも参照のこと。

readreg [**-e** *encoding*] [*register* [*filename*]]

引き数の数に従って、以下の 2 動作のうちのどちらかを行う: 引き数が 0 個または 1 個の場合には、ペーストバッファの内容を、指定されたレジスタか、ユーザに問い合わせて得たレジスタに複製する。引き数が 2 つの場合には、指定されたファイルの内容をレジスタに読み込む (*readbuf* が画面交換ファイルをペーストバッファに読み込むのと同様)。ファイルのエンコーディングを **-e** オプションで screen に知らせることもできる。以下の例はシステムのパスワードファイルを *screen* の画面にペーストしている (レジスタ p を用いている。これはコピーが残るレジスタである):

```
C-a : readreg p /etc/passwd
C-a : paste p
```

redisplay

現在のウィンドウを再描画する。部分再描画モード時に画面全体を書き直す場合に必要となる。

register [**-e** *encoding*] *key string*

指定された *string* をレジスタ *key* に保存する。ファイルのエンコーディングを **-e** オプションで screen に知らせることもできる。 ‘‘paste’’ コマンドも参照すること。

remove

現在のリージョンを kill する。リージョンがひとつしかない場合は no-op となる。

removebuf

“writebuf” コマンドと “readbuf” コマンドが使う画面交換ファイルを削除 (unlink) する。

reset

仮想端末を “電源投入状態” の値にリセットする。アプリケーションが設定 (スクロール領域やグラフィック文字集合など) をおかしくした場合に役に立つ。

resize

現在のリージョンのサイズを変更する。スペースは下方のリージョンから取ったり (に渡したり) するが、もし余裕がなければ上方のリージョンとやり取りする。

resize +N	現在のリージョンの高さを N 増やす
resize -N	現在のリージョンの高さを N 減らす
resize N	現在のリージョンの高さを N にする
resize =	すべてのウィンドウを同じ高さにする
resize max	現在のリージョンの高さを最大にする
resize min	現在のリージョンの高さを最小にする

screen [*-opts*] [*n*] [*cmd* [*args*]]

新しいウィンドウを作る。フロー制御オプション (**–f**, **–fn**,**–fa**)、タイトル (別名) オプション (**–t**)、ログインオプション (**-l**, **-ln**)、端末タイプオプション (**-T <term>**)、全機能フラグ (**-a**)、スクロールバックオプション(**-h** <num>) を各コマンドと共に指定できる。 **-M** オプションを与えると、このウィンドウの監視が on になる。 **-L** オプションを与えると、このウィンドウの出力ロギングが on になる。 0 から 9 の範囲で数値 *n* を与えると(これは省略可能)、新しく生成されるウィンドウにウィンドウ番号 *n* が割り当てられる (この番号が既に使用中ならば、その次に利用可能な番号が使われる)。 “screen” の後にコマンドを指定すると、そのコマンドは (指定された引き数付きで) ウィンドウ内で実行される。コマンドが指定されていなければ、シェルが生成される。したがって、“.screenrc” に以下のような行がある場合、 *screen* はシェルウィンドウを (番号 1 のウィンドウに) 生成し、またマシン foobar に telnet 接続を行うウィンドウ (フロー制御なし、タイトルは “foobar”, ウィンドウ番号は 2) を生成し、この telnet セッションをログファイル (“screenlog.2”) に出力する。

```
# example for .screenrc:
screen 1
screen -fn -t foobar -L 2 telnet foobar
```

以前のバージョンの *screen* とは異なり、“screen” コマンドが “.screenrc” ファイルに含まれている時には、余分のデフォルトウィンドウは生成されない点に注意すること。初期化が終了すると、 *screen* は .screenrc ファイルで指定した最後のウィンドウに切り替わる。ウィンドウが指定されていないと、デフォルトの番号 0 のウィンドウを開く。
screen には “cu” と “telnet” の機能がいくつか組み込まれている。『ウィンドウの種類』の章も参照のこと。

scrollback *num*

現在のウィンドウのスクロールバックバッファの大きさを *num* 行に設定する。デフォルトのスクロールバックは 100 行である。 “defscrollback” コマンドも参照すること。また、現在の設定を確認するには “C-a i (info)” を用いること。

select [*WindowID*]

WindowID で指定したウィンドウに切り替える。これはウィンドウタイトル (英字・数字のウィンドウ名) の先頭からの文字列か、ウィンドウの番号である。このパラメータは省略でき、その場合はユーザに識別子の問い合わせを行う。新しいウィンドウが生成されると、利用可能な最初の数がこのウィンドウに割り当てられる。したがって、先頭のウィンドウは ‘‘select 0’’ で呼び出せる。ウィンドウの総数は、コンパイル時の設定パラメータ MAXWIN によって制限される。特殊な意味を持つ WindowID が二つ存在する。‘‘-’’ は内部のブランクウィンドウを選択し、‘‘.’’ は現在のウィンドウを選択する。後者は screen の ‘‘-X’’ と一緒に用いると便利だろう。

sessionname [*name*]

現在のセッションの名前を変更する。この名前は、 ‘‘screen -list’’ ではプロセス ID と共に表示される点に注意すること。引き数 ‘‘name’’ を省略すると、このセッションの名前が表示される。注意: この変更後も、環境変数 $STY は古い名前を反映したままである。これにより混乱が起こるかもしれない。デフォルト値は tty とホスト名から作られる。

setenv [*var* [*string*]]

環境変数 *var* の値を *string* に設定する。 *var* だけが指定されるとプロンプトが表示され、値の入力がユーザに求められる。パラメータが全く指定されていなければ、変数名と値の両方がユーザに問い合わせられる。この環境変数は、設定以降に fork されるすべてのシェルに継承される。

setsid [**on**|**off**]

通常 screen は各ウィンドウごとに別々のセッションとプロセスグループとを用いる。 setsid を *off* にすると、以降これは行われないようになり、すべてのウィンドウはバックエンドの screen プロセスと同じプロセスグループに属するようになる。これはジョブ制御を不可能にすることに注意。デフォルトは当然 *on* である。このコマンドが有用なのは、おそらくかなりまれな状況に限られるだろう。

shell *command*

新しいシェルを生成するために使うコマンドを設定する。これは環境変数 $SHELL の値より優先される。これは、$SHELL を起動プログラムの指定に用いるような tty 強化プログラムを使いたい時に便利である。コマンドの最初の文字が ’-’ ならば、シェルはログインシェルとして起動される。

shelltitle *title*

起動時や C-A C-c コマンドによって生成された、すべてのシェルのタイトルを設定する。タイトルに関する詳細は、『タイトル (ウィンドウの命名)』の章での議論を参照のこと。

silence [**on**|**off**|*sec*]

ウィンドウの沈黙監視 (silence monitoring) をトグルする。 silence が on で、かつその影響を受けるウィンドウがバックグラウンドに回されたとき、指定された期間の間アクティビティがなければ (沈黙していれば)、沈黙通知のメッセージがステータス行に表示される。デフォルトのタイムアウトは ‘silencewait’ コマンドで変更できる。あるいは ‘on’ や ‘off’ の代わりに秒数を指定することもできる。初期状態では、すべてのウィンドウで silence は off である。

silencewait *sec*

沈黙を監視するウィンドウにおける、メッセージ表示までの待ち時間を指定する。デフォルト値は 30 秒である。

sleep *num*

このコマンドは、.screenrc ファイルの実行を *num* 秒間一時停止させる。キーボード入力があると sleep は終了する。これを用いると、 ‘‘echo’’ による出力を読むチャンスをユーザに与えられ

る。

slowpaste *msec*

paste コマンド ("C-a]") によってテキストを挿入する時の速度を定義する。 slowpaste の値が 0 でなければ、テキストは 1 文字ずつ書き込まれる。 *screen* は 1 文字書き込むたびに *msec* ミリ秒停止し、アプリケーションがこの入力を処理する時間を与える。 slowpaste コマンドは、大量のテキストをペーストしたときに、下層のシステムでフロー制御の問題が起こる場合にのみ用いること。

source *file*

file からコマンドを読んで実行する。source コマンドはネストでき、最大の再帰レベルは 10 である。ファイルが絶対パスでなく、すでに screen が source コマンドを処理中の場合は、 screen のカレントディレクトリよりも先に、実行中の source コマンドファイルの親ディレクトリが、新たなコマンドファイルの検索に用いられる。

temcap/terminfo/termcapinfo コマンドは起動時か再アタッチの時にしか動作しないことに注意。したがってこれらを機能させるには、デフォルトの screenrc ファイルから呼ばれるところに置いておかねばならない。

sorendition [*attr* [*color*]]

テキストのマーキングや印刷メッセージで、 *screen* が行うハイライト表示のさせかたを変更する。 *attr* 修正指定の文法は『文字エスケープ』の章を見ること。現在のデフォルトは "=s dd" である (スタンドアウト、デフォルト色)。

split

現在のリージョンを 2 つに分割する。新たなリージョンの場所を確保するため、画面上のすべてのリージョンはリサイズされる。新しいリージョンにはブランクウィンドウが表示される。リージョンの削除には、"remove" コマンドまたは "only" コマンドを用いること。

startup_message on|off

起動時に著作権表示を表示するかどうかを選ぶ。デフォルト値は、おそらく既にお気付きの通り、'on' である。

stuff *string*

文字列 *string* をカレントウィンドウの入力バッファに置く。これは "paste" コマンドと似ているが、こちらの方がオーバーヘッドが小さい。 "stuff" コマンドでは大きなバッファのペーストはできない。このコマンドは、キー割り当てで用いると非常に便利である。 "bindkey" も参照すること。

su [**username** [**password** [**password2**]]]

ディスプレイのユーザを変更する。省略されたパラメータについては、すべて問い合わせのプロンプトを出す。パスワードをパラメータに指定する場合は、暗号化されていないかたちでなければならない。一番目のパスワードはシステムのパスワードデータベースに対してマッチが行われ、二番目のパスワードは "acladd" コマンドや "password" コマンドで設定された *screen* のパスワードに対してマッチが行われる。 "su" は、 *screen* の管理者がマルチユーザ設定をテストする際に便利であろう。認証に失敗すると、そのユーザがアクセスできるコマンドはユーザ **nobody** が使えるもの (すなわち "detach", "license", "version", "help", "displays") に限られる。

suspend

screen をサスペンドする。 *screen* がサスペンドされている間は、ウィンドウは「デタッチされた」状態となる。この機能は、シェルのジョブ制御機能に依存している。

term *term*

screen がオープンした各ウィンドウの環境変数 $TERM には、デフォルトでは “screen” が設定される。しかし、ローカルの termcap または terminfo データベースに “screen” 用の記述がインストールされていないときには、 $TERM には (例えば) “vt100” を設定すると良い。 *screen* は VT100/ANSI 互換なので、これならあまり問題は起きないだろう。デフォルト値を設定する目的以外に “term” コマンドを使うのは避けたほうがよい。例えば、“screen rlogin othermachine” の実行前に特殊な $TERM (vt100 等) を設定したいような場合である。このような時には、デフォルト値を変更・復帰するのではなく、 “screen -T vt100 rlogin othermachine” というコマンドを用いると良い。

termcap *term terminal-tweaks* [*window-tweaks*]
terminfo *term terminal-tweaks* [*window-tweaks*]
termcapinfo *term terminal-tweaks* [*window-tweaks*]

このコマンドを使うと、独自の termcap エントリを作るという面倒な作業を行わずに、端末の termcap エントリを修正できる。さらに、そのウィンドウ用に生成された termcap のカスタマイズもできる。端末エミュレータが起動したらこれらの設定は意味を持たないので、これらのコマンドは screenrc スタートアップファイルのどこかに置かなければならない。

termcap でなく terminfo データベースで動作するシステムでは、 *screen* は ‘terminfo’ コマンドを認識する (このコマンドの効果は ‘termcap’ コマンドと同じ)。 2 つの異なるコマンドが提供されているのは、例えば (‘%’ を用いた) パラメータ挿入が必要な場合などに、文法が微妙に異なるからである。なお、 ‘terminfo’ コマンドでも、各機能の名前には termcap でのものを使わなければならない。

多くの場合、つまり引き数が terminfo と termcap どちらの文法でも有効な場合は、 ‘termcapinfo’ コマンドが使える。このコマンドは、‘termcap’ コマンドと ‘terminfo’ コマンドとの両方を、同じ引き数で実行するものである。

最初の引き数は、この定義がどの端末に影響するかを指定する。端末名を ‘|’ で区切ることにより、複数の端末を指定できる。すべての端末にマッチさせるには ‘*’ を用い、“vt” で始まるすべての端末にマッチさせるには ‘vt*’ を用いれば良い。

各 *tweak* 引き数には、適切な termcap エントリの先頭に挿入される、 1 つ以上の termcap 定義 (‘:’ で区切る) が含まれる。これによりエントリを拡張したり、既定の値を変更したりできる。最初の tweak は使っている端末の termcap を修正するもので、使っている端末が何らかの機能を実現するために用いる定義を含む。これを変更しないのならば空文字列 (つまり ”) を指定すること。 2 番目の (省略可能な) tweak は、全ウィンドウの termcap を変更するもので、 *screen* が理解する定義を含まなければならない (『仮想端末』の章を参照すること)。

いくつか例を示す:

```
termcap xterm*  LP:hs@
```

これは、 ‘xterm’ で始まるすべての端末はちゃんとした自動マージン機能を持っており、画面の最後の位置を更新できる (LP) が、ステータス行を本当に持っていないこと (’hs’ がない – エントリを無効にするには ‘@’ を追加する) を *screen* に知らせる。なお“vt” で始まるすべての端末名には ‘LP’ があるものと想定しているが、これはその端末に対して termcap コマンドを指定していない場合に限られる。

```
termcap vt*  LP
termcap vt102|vt220  Z0=\E[?3h:Z1=\E[?3l
```

最初の行は、‘vt’ で始まるすべての端末に、確実にマージン機能が動作する ‘LP’ 機能があることを指定する。次の行は、端末が VT102 または VT220 の時に、 132 文字/行モードにしたり (Z0)、これを解除する (Z1) エスケープシーケンスを追加する (画面幅変更コマンドを用いるには、 termcap ファイルに Z0 と Z1 を指定しなければならない)。

```
termcap vt100  ""  l0=PF1:l1=PF2:l2=PF3:l3=PF4
```

これは VT100 の termcap はそのままにしておき、各ウィンドウの termcap エントリにファンク

ションキーのラベルを追加する。

```
termcap h19|z19  am@:im=\E@:ei=\EO  dc=\E[P
```

h19 または z19 termcap での自動マージンを無効にし(am@)、挿入モード (im) と挿入終了 (ei) 機能を有効にする (‘im’ の文字列中の ‘@’ は ‘=’ の後にあるので、この文字列の一部である)。 ‘im’ と ‘ei’ の定義を使っている端末の termcap に入れると、 *screen* は各ウィンドウの termcap の文字挿入機能を自動的に告知する。各ウィンドウの termcap には、文字削除機能 (dc) も追加され、 *screen* はその端末での行更新機能に変換する (つまり文字削除をサポートしていないものとみなす)。

各ウィンドウの termcap エントリをすべて定義したければ、このコマンドを用いるのではなく、 *screen* を実行する前に $SCREENCAP 変数を設定すること。 termcap の定義に関する詳しい情報については、このマニュアルの『仮想端末』での議論と、オンラインマニュアルの **termcap**(5) とを参照すること。

time [*string*]

メッセージ行を用いて、時刻、ホスト名、1, 5, 15 分間のロードアベレージ (使っているシステムで可能なら) を表示する。ウィンドウ固有の情報については “info” を用いること。

string を指定すると、時刻の報告形式を “文字エスケープ” の章で記述する方法に従って変更する。 screen はデフォルトでは "%c:%s %M %d %H%? %l%?" を用いる。

title [*windowtitle*]

現在のウィンドウの名前を *windowtitle* に設定する。名前が指定されないと、 *screen* はユーザに入力を求める。このコマンドは過去のリリースでは ‘aka’ コマンドという名前だった。

unsetenv *var*

環境変数の設定を取り消す。

utf8 [**on**|**off** [**on**|**off**]]

現在のウィンドウが用いるエンコーディングを変更する。 utf8 を on にすると、ウィンドウに送られる文字列は UTF-8 エンコードされる (逆も同じ)。パラメータを省略すると状態をトグルする。二つ目のパラメータを与えると、ディスプレイのエンコーディングも同時に変更される (これは screen の “-U” オプションで指定するほうが良いが)。 “defutf8” も参照のこと。これは新規に生成されるウィンドウのデフォルトを変更する。

vbell [**on**|**off**]

このウィンドウの可視ベル (visual bell) を設定する。可視ベルが on で、使っている端末が可視ベルをサポートしていない場合には、ベル文字 (^G) を受信した際に ‘vbell-message’ がステータス行に表示される。端末が可視ベルをサポートしているかどうかは termcap 変数 ‘vb’ (terminfo では ’flash’) で定義される。

デフォルトでは vbell は off になっており、音声ベル (audible bell) が用いられる。 ‘bell_msg’ も参照のこと。

vbell_msg [*message*]

可視ベルのメッセージを設定する。ウィンドウがベル文字 (^G) を受け取り、かつ vbell が “on” に設定されているが、しかし端末が可視ベルをサポートしていない場合、 *message* がステータス行に表示される。デフォルトのメッセージは “‘Wuff, Wuff!!’” である。パラメータを省略すると、現在のメッセージが表示される。

vbellwait *sec*

screen の可視ベルメッセージが表示された後の待ち時間を定義する。デフォルト値は 1 秒である。

verbose [**on**|**off**]

verbose を on にすると、ウィンドウが生成される (あるいはゾンビ状態から復活する) ときに、コマンド名がエコーされる。パラメータを省略すると、現在の設定が表示される。

version

現在のバージョンとコンパイルされた日時をステータス行に表示する。

wall *message*

すべてのディスプレイにメッセージを書き込む。このメッセージは端末のステータス行に現れる。

width [**-w**|**-d**] [*cols* [*lines*]]

ウィンドウの幅を 80 と 132 カラムの間でトグルする。または引き数を指定すると *num* カラムに設定する。この機能を持った端末と、termcap のエントリ "Z0", "Z1" を必要とする。詳細は "termcap" コマンドを参照すること。同時に高さを指定して変更することもできる。 **-w** オプションを指定すると、ディスプレイのサイズはそのままにウィンドウのサイズだけを変更する。 **-d** はその逆である。

windowlist [**-b**] [**-m**]
windowlist string [*string*]
windowlist title [*title*]

全ウィンドウを表形式で示し、ビジュアルにウィンドウを選択できるようにする。通常の移動キー ("copy" コマンドを見よ) でウィンドウを選択でき、リターンキーで切り替えできる。 **-b** オプションを与えると、screen はリストを表示する前にブランクウィンドウに切り替わるので、カレントウィンドウも選択できるようになる。 **-m** オプションはウィンドウの並び順を変更する。ウィンドウ番号の代わりに、screen が内部で持っている「最終利用時刻」順のリストが用いられる。

表のフォーマットは **string** オプションと **title** オプションとで変更できる。title は表の先頭に表示され、各レコードには string の設定が用いられる。デフォルトの設定は title が "Num Name%=Flags" で、各レコードが "%3n %t%=%f" である。もっと複雑なこと (色設定など) をしたければ、『文字エスケープ』を参照のこと。

windows

メッセージ行を使ってすべてのウィンドウの一覧を表示する。ウィンドウは番号順に並べられ、ウィンドウで起動されたプロセスの名前 (またはウィンドウのタイトル) もそれぞれ表示される。現在のウィンドウには '*' が付く。ひとつ前のウィンドウには '-' が付く。「ログイン」されているすべてのウィンドウには '$' が付く。ベル文字を受け取ったバックグラウンドウィンドウには '!' が付く。監視中で、アクティビティが検出されたバックグラウンドウィンドウには '@' が付く。ログ出力が有効になっているウィンドウには '(L)' が付く。他のユーザが保有しているウィンドウには '&' が付く。ゾンビ状態のウィンドウには 'Z' が付く。この一覧が長すぎて、端末のステータス行ではすべてを表示できない場合には、現在のウィンドウの近辺だけが表示される。

wrap [**on**|**off**]

現在のウィンドウの行折り返しを設定する。行折り返しが有効ならば、ある行の最後のカラムの次にある表示可能文字の出力は、次の行の先頭へ折り返される。追加機能として、バックスペース (^H) も画面の左端から前の行に折り返しで戻る。デフォルト値は 'on' である。

writebuf [**-e** *encoding*] [*filename*]

ペーストバッファの内容を指定したファイルに書き出す。ファイルを指定しないと誰でもアクセスできる画面交換ファイルに書き出す。これは、同じホスト上の *screen* のユーザ間がコミュニケーションするための原始的な方法ともみなせる。 encoding を指定すると、ペーストバッファへの記録は、そのエンコードに合うように自動的に変換される。ファイル名は "bufferfile" コマンドで設定でき、そのデフォルト値は */tmp/screen-exchange* である。

writelock [**on**|**off**|**auto**]

アクセス制御リストによる制限とは別に、必ずしもすべてのユーザが同じウィンドウに同時に書き込みできるわけではない。デフォルトでは、書き込みロックは 'auto' モードになっており、ある特定のウィンドウに最初に切り替えたユーザが、排他的な入力パーミッションを得るようになっている。このユーザがそのウィンドウを離れると、他のユーザが (自動的に) 書き込みロックを取得できるようになる。現在のウィンドウの書き込みロックは "writelock off" コマンドで無効となる。ユーザが "writelock on" コマンドを発行すると、そのユーザは他のウィンドウに切り替えている間も排他的な書き込みパーミッションを保持し続ける。

xoff
xon

CTRL-s / CTRL-q 文字を現在のウィンドウの標準入力キューに挿入する。

zmodem [**off**|**auto**|**catch**|**pass**]
zmodem sendcmd [*string*]
zmodem recvcmd [*string*]

screen の zmodem サポートを定義する。 screen が zmodem 要求を検知した際の動作には、 2 つのモード "pass" と "catch" がある。 "pass" モードでは、screen は通信の末尾に至るまでのすべてのデータをアタッチしているプロセスに転送する。 "catch" モードでは、screen は zmodem の通信端末となり、それぞれ rz/sz コマンドを起動する。"auto" モードでは、 screen はウィンドウが tty (例えばシリアルライン) なら "catch" を用い、それ以外なら "pass" を用いる。この機能はまだ実験段階のものである。

zombie [*keys*]
defzombie [*keys*]

デフォルトでは *screen* のウィンドウはユーザのプロセス (シェル等) が終了すると即座に削除される。 2 つのキーからなる文字列を zombie コマンドに指定すると、「死んだ」ウィンドウもリスト中に残る。このようなウィンドウは **kill** コマンドで削除できる。「死んだ」ウィンドウで 1 番目のキーを押すと同じ効果が得られる。 2 番目のキーを押すと、 *screen* はこのウィンドウを復活させようとする。つまり、このウィンドウで実行されていたプロセスを再び起動する。パラメータを指定せずに **zombie** コマンドを呼び出すと、 zombie の設定がクリアされ、したがってウィンドウは内部のプロセスが終了したときに消えるようになる。

zombie の設定はすべてのウィンドウに対してグローバルに操作されるので、このコマンドは **defzombie** とのみ呼ぶべきである。ウィンドウごとの設定が必要となるまでは、 **zombie** コマンドは **defzombie** コマンドと同じにしておく。

メッセージ行

screen は情報メッセージや診断メッセージをメッセージ行に表示する。この行は通常画面の一番下に配置されるが、コンパイル時の設定で画面の一番上に現れるようにも定義できる。使っている端末の termcap にステータス行の定義がある場合、 *screen* はこれをメッセージ表示に用いる。それ以外の場合には、現在の画面のうち 1 行が一時的に上書きされ、出力がしばらく中断される。数秒後にこのメッセージ行は自動的に消されるが、 (ステータス行の無い端末では) キー入力を行えばすぐに消すこともできる。

メッセージ行の機能は、カレントウィンドウで動作中のアプリケーションからも利用できる。こ

れには ANSI のプライバシーメッセージ *(Privacy message)* 制御シーケンスを使う。例えば、シェルから以下のようなコマンドを試してみよ:

```
echo '<esc>^Hello world from window '$WINDOW'<esc>\\'
```

ここでの '<esc>' は *escape*、'^' は上矢印キーである。また '\\' はバックスラッシュ 1 つに置き換えられる。

ウィンドウの種類

screen は 3 種類のウィンドウを提供している。新しいウィンドウは *screen* の **screen** コマンドで生成される (『カスタマイズ』の章のエントリも参照)。どのタイプのウィンドウが生成されるかは、 **screen** コマンドに与える最初のパラメータによって決まる。各タイプは、それぞれ基本タイプの特殊ケースである。これらは *screen* を 100 以上のウィンドウを扱うコンソールマルチプレクサとして効率良く利用できるように追加されてきた。

- 基本ウィンドウは、シェル (パラメータが与えられない場合のデフォルト) またはシェルから起動される何らかのシステムコマンド (例: **slogin** など) を内部で実行する。

- tty (キャラクタスペシャルデバイス) の名前 (例えば "/dev/ttya") を最初のパラメータに指定すると、そのウィンドウはこのデバイスに直接接続する。このウィンドウタイプは "screen cu -l /dev/ttya" と似ている。指定するデバイスノードには読み書きのアクセス権が必要で、ノードには排他オープンが試みられ、接続ラインを busy 状態にしようとする。追加パラメータを与えることもできる。これはコンマ区切りのフラグの並びで、各フラグの意味は **stty**(1) のものと同様である:

  <baud_rate>
  : 通常は 300, 1200, 9600, 19200 のいずれか。この指定は送信・受信両方の速度に影響する。

  cs8 または cs7
  : 1 バイトあたり 8 ビット (または 7 ビット) を送るよう指定する。

  ixon または -ixon
  : データ送信におけるソフトウェアフローコントロール (CTRL-S/CTRL-Q) を有効に (無効に) する。

  ixoff または -ixoff
  : データ受信におけるソフトウェアフローコントロールを有効に (無効に) する。

  istrip または -istrip
  : 各受信バイトの 8 ビット目を消去 (保持) する。

  これらのパラメータは、できるだけ指定したほうが良いだろう。指定しないオプションの接続時の値は、端末ドライバが決めることになる。これらの値はシステムに依存し、デフォルトが定義されているかもしれないし、前回の接続時の値を保存して使うかもしれない。

  tty ウィンドウでは、 **info** コマンドによってモデムの制御線がいくつかステータス行に表示される。これらには 'RTS', 'CTS', 'DTR', 'DSR', 'CD' などが含まれうる。何が表示されるかは、利用できる ioctl() とシステムのヘッダファイル、シリアルボードの物理的な特性などに依存する。論理 low (inactive) の信号は、名前の前にエクスクラメーション (!) が付く。付いていない信号は論理 high (active) である。ハードウェアがサポートしていないが、ioctl() インターフェースは存在する信号は、通常 low として表示される。

  CLOCAL ステータスビットが真の場合、モデム信号全体が中括弧 ({ と }) の内部に置かれる。CRTSCTS ビットが立っている場合は 'CTS' 信号が括弧で括られ、 TIOCSOFTCAR ビットが立っている場合は 'CD' 信号が括弧で括られる。

  tty ウィンドウでは、 **break** コマンドによってデータ送信線 (TxD) が指定した期間だけ low になる。これは、通信の他端にてブレーク信号と解釈されることが期待できる。 **break** が発行

されると、データ送信は行われず、モデム制御線は変更されない。

- 最初のパラメータが "/telnet" だと、二番目のパラメータはホスト名とされ、三番目のパラメータ (省略可) は TCP のポート番号を指定することになる (デフォルトは 10 進数の 23)。*screen* はリモートホストでリッスンしているサーバに接続し、そのサーバとの通信に telnet プロトコルを用いる。

  telnet ウィンドウでは、 **info** コマンドによって、ステータス行の末尾に接続の詳細が角括弧 ([と]) に括られて表示される。

  b BINARY。この接続はバイナリモードである。

  e ECHO。ローカルエコーは無効になっている。

  c SGA。この接続は「キャラクタモード」である (デフォルトは「ラインモード」)。

  t TTYPE。この端末タイプはリモートホストからリクエストされた。 *screen* は、特に指定がなければ端末名 "screen" を送る ('term' コマンドも参照)。

  w NAWS。リモートサイトはウィンドウサイズの変更を通知される。

  f LFLOW。リモートホストはフロー制御情報を送る (現時点では無視される)。

  他にもデバッグ用のフラグとして、x (XDISPLOC), t (TSPEED), n (NEWENV) が存在する。

  telnet ウィンドウでは、 **break** コマンドを実行すると telnet コード IAC BREAK (10 進 243) がリモートホストに送信される。

  このウィンドウタイプは *screen* のコンパイル時に BUILTIN_TELNET オプションを定義していないと使えない。

文字エスケープ

screen には、現在時刻のメッセージやファイル名などの情報を表示するための、エスケープの仕組みがある。エスケープ文字は '%' であるが、ひとつだけ例外があり、ウィンドウのハードステータスの内部ではこれの代わりに '^%' ('^E') を用いる。

以下にサポートされているエスケープの完全なリストを示す:

% エスケープ文字自身

a 'am' または 'pm' のどちらか

A 'AM' または 'PM' のどちらか

c 現在の時刻 HH:MM (24h 表記)

C 現在の時刻 HH:MM (12h 表記)

d 月内日

D 曜日

f ウィンドウのフラグ

F ウィンドウにフォーカスがあると %? を真にする

h ウィンドウのハードステータス

H システムのホスト名

l システムの現在の負荷 (load)

m 月 (数値)

M 月 (名前)

n ウィンドウ番号

s 秒

t ウィンドウのタイトル

u このウィンドウを使っている他のユーザすべて

w 全ウィンドウの番号と名前。限定詞 '-' をつけると現在のウィンドウまで、限定詞 '+' をつけると現在のウィンドウ以降、となる。

W 現在のウィンドウを除く、全ウィンドウの番号と名前

y 年 (下二桁)

Y 年 (全表示)

? 次の '%?' までの部分が表示される。ただしその内部の '%' エスケープが空でない文字列に展開される場合に限られる。

:

= (TeX の hfill のように) ディスプレイの幅に合わせて文字埋めをする。数値を指定すると、ウィンドウ幅のそのパーセント分を埋める。限定詞 '0' をつけると、screen はその数値を絶対位置として扱う。直前に指定された絶対位置から相対的な文字埋めをするには、限定詞 '+' を付ける。また相対的な右マージンを文字埋めするには限定詞 '-' を付ける。文字埋め動作では、指定位置が現在位置よりも前にあると、文字列を切り捨てて短縮する。これを変更するには限定詞 'L' を加える。

<

> 現在のテキスト位置を、次回の文字列短縮用にマークする。短縮が必要になると、screen はこのマークした位置を出力領域の指定パーセントの位置に移動することによって、文字列を短縮しようとする (この領域は、直前の絶対文字埋め位置から文字列短縮演算子で指定した位置まで)。限定詞 'L' をつけると、screen は切り捨て部を '...' で示すようになる。

{ 次の ``}'' の間にある文字列の属性/色を変更する。

‘ ここでは長さ限定詞がコマンドの特定に用いられる (本来は誤用だが)。

'c' と 'C' の各エスケープには限定詞 '0' をつけることができ、この場合 *screen* は埋め文字としてスペースではなくゼロを用いる。限定詞 '0' は '=' エスケープにも付き、その場合は絶対位置指定を用いさせる。'n' と '=' の各エスケープは長さ限定詞 (例えば '%3n') を理解し、'D' と 'M' は前に 'L' を付けると長い名前になる。'w' と 'W' は 'L' を付けるとウィンドウのフラグも同時に表示する。

属性/色指定は、これらの設定を変更するために用いる。書式は ``[属性指定] [色定義]+U である。属性指定と色定義とが混乱しそうな場合には、属性指定の前に変更タイプの指定をつけなければならない。変更タイプとしては次のものが利用できる:

+ 現在の属性に指定セットを追加する

- 現在の属性から指定セットを削除する

! 現在の属性において、指定セットを反転させる

= 現在の属性を、指定セットに変更する

属性のセットは 16 進の数値か、次に示す文字の組で指定できる:

d 暗くする (dim)
u アンダーライン
b ボールド
r リバース
s 強調 (standout)
B ブリンク

色は 16 進数か、あるいは背景色と前景色を示す 2 つの文字 (背景、前景の順) で指定する。次の色がある:

k ブラック
r レッド
g グリーン
y イエロー
b ブルー
m マゼンダ
c シアン
w ホワイト
d デフォルト色
. 色を変更しない

これらの大文字版は明るい色になる。疑似色として 'i' を指定でき、この場合は色はそのままで明るさだけを変更できる。

数値にせよ文字にせよ、一色だけを指定した場合に、それが前景色として扱われるか背景色になるかは、現在の属性に依存する。リバースが設定されていると、前景色ではなく背景色が変更される。これが嫌なら、色の前に ".'' を付ければ良い。 2 文字の色記述においても同様の動作をさせたければ、同じくそれらの前に ".'' を付ければ良い。

特殊ケースとして、 ''%{-}'' は直前に変更された属性と色とを復元する (すなわち色変更のスタックを 1 レベル pop する)。

例：

''G'' 明るいグリーン

''+b r'' ボールドのレッド

''= yd'' 全属性をクリア、イエローの背景にデフォルト色で書く。

%-Lw%{= BW}%50>%n%f* %t%{-}%+Lw%<
現在のウィンドウを中心として、利用できるウィンドウを、現在の幅に表示できるだけ表示する。現在のウィンドウはブルーの背景にホワイトで書かれる。これは ''hardstatus alwayslastline'' と一緒に使える。

%?%F%{.R.}%?%3n %t%? [%h]%?
ウィンドウの番号・タイトル・ハードステータス (設定されている場合)。また、これがフォーカスされている場合は背景を赤にする。 ''caption string'' に便利。

フロー制御

各ウィンドウにはフロー制御の設定があり、 *screen* はこれを基に、XON 文字と XOFF 文字 (およびおそらく割り込み文字) の扱い方を決める。フロー制御が無効になっていると、 *screen* は XON 文字と XOFF 文字を無視する。この場合、ユーザは単に入力するだけでこれらの文字を実行中のプログラムに送れる (例えば、*emacs* エディタ等ではこの設定が便利である)。その代わり、''普通''のプログラムからの出力が、 XOFF に応答して一時停止するまでに、より長い時間がかかる。フロー制御が有効になっていると、XON 文字と XOFF 文字を使って、現在のウィンドウの出力を即座に停止できる。この場合でも、XON 文字や XOFF 文字を実行中プログラムへ送信することはできるが、これにはそれぞれに対応する *screen* の 2 文字コマンドを使わなければならない (普通は ''C-a q'' (xon) と ''C-a s'' (xoff))。 xon/xoff コマンドは、C-s と C-q を横取りしてしまうような端末越しに、これらの文字を入力する場合にも役立つ。

各ウィンドウはフロー制御に関する初期値を持ち、これは **–f** オプションまたは .screenrc コマンドの ''defflow'' で設定できる。デフォルトでは、各ウィンドウには自動フロー切り替えが設定される。この値は、''flow'' コマンド ("C-a f" に割り当てられている) を用いると、'fixed on', 'fixed off', 'automatic' の 3 つの状態に、順々にトグルできる。

自動フロー切り替えモードは (''rlogin'' のように) TIOCPKT モードを用いてフロー制御を扱う。tty ドライバが TIOCPKT をサポートしていない場合、 *screen* はアプリケーションテンキーの現在の状態に基づいて正しいモードを見つけようとする。つまり、テンキーがアプリケーションモードならフロー制御無効、およびその逆、である。もちろん、必要ならば手動でフロー制御を操作することもできる。

フロー制御を有効にしているにもかかわらず、割り込みキー (通常は C-c) を押してから表示に割り込みがかかるまでに 6-8 行スクロールしてしまうような場合には、“interrupt” オプションを与えて *screen* を起動してみること (.screenrc で “flow” コマンドに “interrupt” フラグを追加するか、コマンドラインオプションの **–i** を用いる)。こうすると、割り込みを受けたプログラムから *screen* が受け取って蓄積していた出力はフラッシュされるようになる。これによる不都合は、仮想端末のメモリが非フラッシュ出力を含むことであり、これによって、まれに出力に多少不正確な部分が現れるかもしれない。例えば画面の切り替えと復帰を行ったり、 “C-a l” を使って画面の更新を行うと、 “interrupt” をオンにしていなかった場合に得られたはずの出力が得られる。また、割り込み文字を入力として期待するプログラムを実行する際には、フロー制御を無効にする (あるいは自動フロー制御を用いてこれを自動的に無効にする) 必要があるかもしれない。なぜなら、フロー制御が有効だと、仮想端末から物理端末への出力に割り込みがかかることがあり得るからである。このようになった場合も、“C-a l” を行って画面をリフレッシュしさえすれば、元に戻るはずである。各モードを試してみて、一番都合が良いと思ったものを使うとよいだろう。

タイトル (ウィンドウの命名)

ウィンドウ表示に現れる各ウィンドウの名前 (“windows” コマンド (C-a w) で表示される) は、title コマンドのどれかを用いればカスタマイズできる。通常この名前は、そのウィンドウで生成されるプログラムの実際のコマンド名になる。しかし、同じ名前の複数のプログラムを区別したり、ウィンドウの現在の状態をその場で名前に反映させたりできれば、便利な場合もあるだろう。

すべてのシェルウィンドウに対するデフォルト名は .screenrc ファイルの “shelltitle” コマンドで設定できる。一方、他のウィンドウはすべて “screen” コマンドで生成されるので、 **–t** オプションを使えば名前を設定できる。対話的に行うには、タイトル文字列用エスケープシーケンス (<esc>k*name*<esc>\) と “title” コマンド (C-a A) がある。前者をアプリケーションから出力させれば、ソフトウェア制御でウィンドウの名前を変更できる。後者は名前の入力をユーザに求める。定義済みの名前を “title” コマンドのキーに割り当てておいて、入力なしに即座に設定を行わせることもできる。

最後に、 *screen* にはシェルの場合にのみ使われるヒューリスティックなタイトル指定方法がある。これを実現するには、ウィンドウの名前を “*search*|*name*” に設定し、プロンプトの一部で空の title エスケープシーケンスが出力されるようにする。 *search* の部分には「プロンプト末尾」の検索文字列を指定し、 *name* の部分にはそのウィンドウのデフォルトのシェル名を指定する。*name* が ‘:’ で終わる場合、 *screen* は、そのウィンドウで現在実行中のコマンドと認識しているものをウィンドウのシェル名の最後に追加する (例えば “*name:cmd*”)。それ以外の場合には、コマンドの実行中にはそのコマンド名がシェル名を上書きする。

以下にその動作を示す: ユーザはシェルのプロンプトを変更して、空の title エスケープシーケンス (<esc>k<esc>\) がプロンプトの一部に出力されるようにしなければならない。プロンプトの末尾は、タイトルの *search* 部分に指定した文字列と同じでなければならない。このようにすると、*screen* はタイトルエスケープシーケンスを使って以前のコマンド名をクリアし、次のコマンドのための準備を行う。次に、シェルから改行文字を受け取ると、プロンプトの末尾を検索する。検索にマッチする部分が見つかると、マッチした文字列の後ろの最初の単語が取り出され、これがコマンド名として使われる。コマンド名が ’!’, ’%’, ’^’ のいずれかで始まる場合、 *screen* は次の行に出てくる最初の単語を (もし見つかれば) いま見つかった名前よりも優先する。これにより、csh ユーザがジョブ制御や履歴の再呼び出しコマンドを使ったときでも、まともなコマンド名を取得できる。

以下に .screenrc での指定例をいくつか示す:

```
screen -t top 2 nice top
```

この行を .screenrc に追加すると、 nice コマンドでプライオリティを設定した “top” コマンドが、ウィンドウ 2 において (“nice” ではなく) “top” という名前で起動する。

```
shelltitle ’> |csh’
screen 1
```

このコマンドでは、与えられた shelltitle を持つシェルが起動される。指定されたタイトルは自動設定タイトルであり、次のような形のプロンプト+コマンド入力を監視する:

```
/usr/joe/src/dir> trn
```

('> ' の後の部分がコマンド名として扱われる)。このコマンドの実行中には、ウィンドウのステータスには "trn" という名前が表示され、実行が終了するとこれは "csh" に戻る。

```
bind R screen -t '% |root:' su
```

このコマンドが .screenrc に書かれていると、キーシーケンス "C-a R" に対して "su" コマンドが割り当てられ、自動設定タイトル名として "root:" が与えられる。この自動タイトル設定が動作するのは、 screen には以下のような表示が渡った場合である:

```
% !em
emacs file.c
```

ここでは、ユーザは csh の履歴コマンド "!em" を入力している。これは以前に入力した "emacs" コマンドを実行するものである。このコマンドの実行中には、ウィンドウのステータスには "root:emacs" が表示され、実行の終了時には "root:" だけの表示に戻る。

```
bind o title
bind E title ""
bind u title (unknown)
```

最初の割り当てには全く引き数がないので、ユーザに対してタイトルの問い合わせが行われる ("C-a o" を入力した時と同様)。 2 番目の割り当ては現在の自動タイトル設定をクリアする(C-a E)。 3 番目の割り当ては現在のウィンドウのタイトルに "(unknown)" を設定する (C-a u)。

空の title エスケープシーケンスをプロンプトに追加する際に注意すべきなのは、一部のシェル (csh など) では、制御文字などもすべてプロンプトの長さに数え入れる点である。このような表示されない文字の数が 8 の倍数でないと、バックスペースでタブを消した時の表示がおかしくなる。これを回避する方法の 1 つは、以下のようなプロンプトを使うことである:

```
set prompt='^[[0000m^[k^[\% '
```

エスケープシーケンス "<esc>[0000m" は文字の属性を統一するだけでなく、表示されない文字の長さを 0 を使って 8 個に合わせる。 bash ユーザは多分、PROMPT_COMMAND 中でエスケープシーケンスを出力するのがよいだろう:

```
PROMPT_COMMAND='echo -n -e "\033k\033\134"'
```

(bash バージョン 1.04 のバグを回避するため、筆者は "134" を使って '\' を出力している)

仮想端末

screen セッションの各ウィンドウは VT100 端末をエミュートしており、いくつかの機能も追加されている。 VT100 エミュレータはハードコーディングされており、他の種類の端末をエミュレートすることはできない。

通常 *screen* はできる限り VT100/ANSI 標準をエミュレートしようとする。しかし端末が特定の機能を持っていない場合には、エミュレーションが完全でないこともある。このような場合、 *screen* は一部の機能が欠けていることをアプリケーションに伝えなければならない。これは termcap を使っているマシンでは問題にはならない。なぜなら、 *screen* は $TERMCAP 変数を使って標準の *screen* termcap をカスタマイズできるからである。

しかし、別のマシンに rlogin した時や、使っているマシンが terminfo しかサポートしていない場合には、この方法はうまくいかない。そのため *screen* にはこのような場合を扱う手段が用意されている。その動作を以下に示す:

screen は自力で端末名を調べようとする時、まず "screen.<term>" という名前のエントリを参照する。ここで <term> は $TERM 変数の内容である。このようなエントリが存在しないと、 *screen* は "screen" (あるいは端末の幅が広ければ (132 カラム以上) "screen-w") を試す。このエン

トリも見つからない場合には、代わりに “vt100” を使う。

つまり、重要な機能 (例えば文字削除や EOS までのクリア) をサポートしていない端末を使っている場合でも、これらの機能を無効にした (“screen.<dumbterm>” という名前の) 新しい *screen* 用 termcap/terminfo エントリをビルドすれば良い、という考え方である。使うマシンにこのエントリをインストールすれば、 rlogin の際にも正しい termcap/terminfo エントリを維持できる。端末名はすべての新規ウィンドウの $TERM 変数に設定される。 *screen* はエミュレートしている仮想端末の機能を反映させた $TERMCAP も設定する。ただし、terminfo データベースを使っているマシンにはこの変数は効果がないことに注意。さらに、変数 $WINDOW には各ウィンドウのウィンドウ番号が設定される。

仮想端末が実際にサポートしている機能のセットは、物理端末がサポートしている機能に依存する。そのため、もし例えば、物理端末が下線モードをサポートしていなければ、 *screen* は ‘us’ や ‘ue’ 機能をそのウィンドウの $TERMCAP 変数に設定しない。しかし、 *screen* を実行するために、端末が最低限サポートしていなければならない機能がいくつか存在する。例えば、スクロール機能、画面クリア、カーソル位置の直接指定などである (さらに、 *screen* はハードコピー端末や、文字の重ね打ちを行う端末では動作しない)。

また、ユーザは .screenrc で “termcap” コマンドを用いたり、起動前に変数 $SCREENCAP を定義することにより、 *screen* が用いる $TERMCAP の値をカスタマイズできる。後者が定義されていると、その値がそのままの形で各ウィンドウの $TERMCAP 変数にコピーされる。これは端末の完全な定義、または端末 “screen” (や “screen-w”) が定義されているファイル名、のいずれかである。

システムが termcap でなく terminfo データベースを用いている場合には、 *screen* は .screenrc コマンドの “terminfo” に従う点に注意すること。

screen が呼び出された端末の termcap エントリ中に、ブール値の ‘G0’ 機能があると、 *screen* の端末エミュレーションは複数の文字集合をサポートする。これによりアプリケーションは、例えば、 VT100 のグラフィックス文字集合や特定の国の文字集合を利用できる。 ISO 2022 に含まれるものとしては、以下の制御機能がサポートされている: ロックシフト *G0* (*SI*), ロックシフト *G1* (*SO*), ロックシフト *G2*, ロックシフト *G3*, シングルシフト *G2*, シングルシフト *G3*。仮想端末の生成時やリセット時には、ASCII 文字集合が *G0* から *G3* までに指定される。‘G0’ 機能が存在すれば、 *screen* は ‘S0’, ‘E0’, ‘C0’ 機能を (存在すれば) 評価する。 ‘S0’ は、(*SI* の代わりに) グラフィックス文字集合を有効にし、開始するために端末が用いるシーケンスである。 ‘E0’ はこれに対応して *SO* の代わりに使うシーケンスである。 ‘C0’ は、セミグラフィックスモード時に使われる、文字ごとの変換文字列を与える。この文字列は ‘acsc’ terminfo エントリのように構築されている。

‘po’ 機能と ‘pf’ 機能が端末の termcap エントリに存在すると、 *screen* のウィンドウ内で実行されているアプリケーションは、出力を端末のプリンタポートに送信できる。これを使うと、ユーザは端末に接続されているプリンタに対し、ウィンドウのアプリケーションの出力を送信できる。この際にも、他のウィンドウはすべてアクティブなままである (ひとまとまりの出力ごとにプリンタポートは繰り返し有効/無効にされる)。この副作用として、別のウィンドウで動作しているプログラムも、同時に出力をプリンタに送信できてしまう。プリンタに送られたデータはウィンドウには表示されない。 *info* コマンドは、プリンタがアクティブな時には行の先頭に ‘PRIN’ を表示する。

screen はウィンドウごとにハードステータス行を管理する。あるウィンドウが選択されると、ディスプレイのハードステータスはそのウィンドウのハードステータス行に一致するよう更新される。ディスプレイにハードステータスがなければ、この行は screen の標準メッセージとして表示される。ハードステータス行は ANSI のアプリケーションプログラムコマンド (Application Program Command, APC) である “ESC_<string>ESC\” で変更できる。xterm ユーザに便利なように、 “ESC]0..2;<string>^G” というシーケンスも使えるようになっている。

一部の機能は、それらが物理端末でちゃんと実装できている場合に限り、仮想端末の $TERMCAP に設定される。例えば ‘dl’ (行の削除)は、端末が行削除または領域のスクロールのどちらかをサポートしている場合にのみ $TERMCAP 変数に設定される。セッションを別の端末から再アタッチした時には、これが混乱の元になりやすいので注意すること。 $TERMCAP の値は親プロ

セスからは変更できないからである。

「別画面 (alternate screen)」機能はデフォルトでは有効になっていない。有効にするには .screenrc コマンド **altscreen** をセットすること。

screen が認識する制御シーケンスの一覧を以下に示す。 ‘‘(V)’’ は VT100 固有の機能であることを示し、 ‘‘(A)’’ は ANSI または ISO 固有の機能であることを示す。

ESC E		次の行
ESC D		インデックス
ESC M		逆順のインデックス
ESC H		水平タブセット
ESC Z		VT100 の識別文字列を送る
ESC 7	(V)	カーソルと属性を保存する
ESC 8	(V)	カーソルと属性を復元する
ESC [s	(A)	カーソルと属性を保存する
ESC [u	(A)	カーソルと属性を復元する
ESC c		初期状態へのリセット
ESC g		可視ベル
ESC Pn **p**		カーソルの可視制御 (97801)
Pn = **6**		不可視
7		可視
ESC =	(V)	アプリケーションキーパッドモード
ESC >	(V)	数値キーパッドモード
ESC # 8	(V)	画面を E で埋める
**ESC **	(A)	文字列の終端子
ESC ^	(A)	プライベートなメッセージ文字列 (メッセージ行)
ESC !		グローバルなメッセージ文字列 (メッセージ行)
ESC k		定義文字列の別名
ESC P	(A)	デバイス制御文字列。文字列を、解釈せず、直接ホストの端末に出力する。
ESC _	(A)	アプリケーションプログラムコマンド (ハードステータス)
ESC] 0 ; string ^G	(A)	オペレーティングシステムのコマンド (ハードステータス、xterm のタイトルのハック)
ESC] 83 ; cmd ^G	(A)	screen コマンドを実行する。これは screen にマルチユーザのサポートが組み込まれている場合にのみ動作する。アクセスコントロールリストのチェックには、疑似ユーザ ‘‘:window:’’ が用いられる。権限を持たず、必要なコマンドだけを実行できるようなユーザを作るには、 ‘‘addacl :window: -rwx #?’’ を用いること。
Control-N	(A)	ロックシフト G1 (SO)
Control-O	(A)	ロックシフト G0 (SI)
ESC n	(A)	ロックシフト G2
ESC o	(A)	ロックシフト G3

ESC N	(A)	シングルシフト G2
ESC O	(A)	シングルシフト G3
ESC (Pcs	(A)	文字集合として G0 を指定
ESC) Pcs	(A)	文字集合として G1 を指定
ESC * Pcs	(A)	文字集合として G2 を指定
ESC + Pcs	(A)	文字集合として G3 を指定
ESC [Pn **;** Pn **H**		カーソルの位置の直接指定
ESC [Pn **;** Pn **f**		カーソルの位置の直接指定
ESC [Pn **J**		ディスプレイの中での削除
Pn = None または **0**		カーソル位置からスクリーン末尾まで
1		スクリーン先頭からカーソル位置まで
2		スクリーン全体
ESC [Pn **K**		行の中での削除
Pn = None or **0**		カーソル位置から行末まで
1		行頭からカーソル位置まで
2		行全体
ESC [Pn **X**		文字削除
ESC [Pn **A**		カーソルを上に移動
ESC [Pn **B**		カーソルを下に移動
ESC [Pn **C**		カーソルを右に移動
ESC [Pn **D**		カーソルを左に移動
ESC [Pn **E**		カーソルを次の行へ移動
ESC [Pn **F**		カーソルを前の行へ移動
ESC [Pn **G**		カーソルを水平位置へ
ESC [Pn **‘**		カーソルを水平位置へ
ESC [Pn **d**		カーソルを垂直位置へ
ESC [Ps **;...;** Ps **m**		グラフィックスの解釈を選択
Ps = None or **0**		デフォルトの解釈を行う
1		ボールド
2	(A)	薄い色で描画
3	(A)	強調モード (ANSI ではイタリック表示)
4		アンダーラインを引く
5		ブリンクさせる
7		イメージを反転させる
22	(A)	通常の明度
23	(A)	強調 モードをオフにする (ANSI: イタリック化のオフ)
24	(A)	アンダーラインを引かない
25	(A)	ブリンクしない

27	(A)	ポジティブ (反転していない) イメージ
30	(A)	前景色をブラックにする
31	(A)	前景色をレッドにする
32	(A)	前景色をグリーンにする
33	(A)	前景色をイエローにする
34	(A)	前景色をブルーにする
35	(A)	前景色をマゼンダにする
36	(A)	前景色をシアンにする
37	(A)	前景色をホワイトにする
39	(A)	前景色をデフォルト色にする
40	(A)	背景色をブラックにする
...		
49	(A)	背景色をデフォルト色にする
ESC [Pn **g**		タブクリア
Pn = None or **0**		現在位置のタブをクリアする
3		すべてのタブをクリアする
ESC [Pn **;** Pn **r**	(V)	スクロール領域を設定する
ESC [Pn **I**	(A)	水平タブ
ESC [Pn **Z**	(A)	後方タブ
ESC [Pn **L**	(A)	行を挿入する
ESC [Pn **M**	(A)	行を削除する
ESC [Pn **@**	(A)	文字を挿入する
ESC [Pn **P**	(A)	文字を削除する
ESC [Pn **S**		スクロール領域を上にスクロールする
ESC [Pn **T**		スクロール領域を下にスクロールする
ESC [Pn **^**		スクロール領域を下にスクロールする
ESC [Ps **;...;** Ps **h**		モード設定
ESC [Ps **;...;** Ps **l**		モード解除
Ps = **4**	(A)	挿入モード
20	(A)	自動ラインフィードモード
34		通常のカーソル表示
?1	(V)	アプリケーションカーソルキー
?3	(V)	端末の幅を 132 カラムに変更
?5	(V)	ビデオ反転
?6	(V)	オリジン *(origin)* モード
?7	(V)	折り返し *(wrap)* モード
?9		X10 マウス追跡
?25	(V)	可視カーソル

?47		別画面 (古い xterm コード)
?1000	(V)	VT200 マウス追跡
?1047		別画面 (新しい xterm コード)
?1049		別画面 (新しい xterm コード)
ESC [5 i	(A)	プリンタへの転送を開始する (ANSI の Media Copy シーケンス)
ESC [4 i	(A)	プリンタへの転送を止める (ANSI の Media Copy シーケンス)
ESC [8 ; Ph **;** Pw **t**		ウィンドウを 'Ph' 行、'Pw' カラムにリサイズする (SunView 用の特別シーケンス)
ESC [c		VT100 識別文字列を送る
ESC [x		端末パラメータ報告を送る
ESC [> c		VT220 のセカンダリデバイス属性文字列を送る
ESC [6 n		カーソル位置の報告を送る

入力の変換

VT100 のエミュレーションを完全に行うには、 *screen* はユーザのキー入力によって生成される入力ストリーム中から文字シーケンスを検出して、 VT100 形式のエスケープシーケンスを挿入しなければならない。 *screen* では、任意のコマンドを任意の文字シーケンスへ割り当てることが可能であり、この動作を非常に柔軟に行える。標準の VT100 エミュレーションの場合、コマンドは常にウィンドウの入力バッファに文字列を挿入する (コマンド表の **stuff** コマンドも参照すること)。キー入力によって生成されるシーケンスは、異なるタイプの端末から再アタッチした後には変わることもあるので、キーの termcap 名にコマンドを割り当てることもできるようになっている。これにより *screen* は再アタッチの後でも必ず正しい割り当てを挿入できる。詳しい文法と使用例については **bindkey** コマンドを参照すること。

以下にデフォルトのキー割り当ての表を示す。(A) はキーボードがアプリケーションモードに切り替えられている場合に実行されるコマンドであることを示す。

キーの名前	termcap 名	コマンド	
カーソルキー上	ku	stuff \033[A	
		stuff \033OA	(A)
カーソルキー下	kd	stuff \033[B	
		stuff \033OB	(A)
カーソルキー右	kr	stuff \033[C	
		stuff \033OC	(A)
カーソルキー左	kl	stuff \033[D	
		stuff \033OD	(A)
ファンクションキー 0	k0	stuff \033[10~	
ファンクションキー 1	k1	stuff \033OP	
ファンクションキー 2	k2	stuff \033OQ	
ファンクションキー 3	k3	stuff \033OR	
ファンクションキー 4	k4	stuff \033OS	
ファンクションキー 5	k5	stuff \033[15~	
ファンクションキー 6	k6	stuff \033[17~	
ファンクションキー 7	k7	stuff \033[18~	
ファンクションキー 8	k8	stuff \033[19~	
ファンクションキー 9	k9	stuff \033[20~	
ファンクションキー 10	k;	stuff \033[21~	
ファンクションキー 11	F1	stuff \033[22~	
ファンクションキー 12	F2	stuff \033[23~	
Home	kh	stuff \033[1~	

End	kH	stuff \033[4~	
Insert	kI	stuff \033[2~	
Delete	kD	stuff \033[3~	
Page up	kP	stuff \033[5~	
Page down	kN	stuff \033[6~	
テンキー 0	f0	stuff 0	
		stuff \033Op	(A)
テンキー 1	f1	stuff 1	
		stuff \033Oq	(A)
テンキー 2	f2	stuff 2	
		stuff \033Or	(A)
テンキー 3	f3	stuff 3	
		stuff \033Os	(A)
テンキー 4	f4	stuff 4	
		stuff \033Ot	(A)
テンキー 5	f5	stuff 5	
		stuff \033Ou	(A)
テンキー 6	f6	stuff 6	
		stuff \033Ov	(A)
テンキー 7	f7	stuff 7	
		stuff \033Ow	(A)
テンキー 8	f8	stuff 8	
		stuff \033Ox	(A)
テンキー 9	f9	stuff 9	
		stuff \033Oy	(A)
テンキー +	f+	stuff +	
		stuff \033Ok	(A)
テンキー -	f-	stuff -	
		stuff \033Om	(A)
テンキー *	f*	stuff *	
		stuff \033Oj	(A)
テンキー /	f/	stuff /	
		stuff \033Oo	(A)
テンキー =	fq	stuff =	
		stuff \033OX	(A)
テンキー .	f.	stuff .	
		stuff \033On	(A)
テンキー ,	f,	stuff ,	
		stuff \033Ol	(A)
テンキー Enter	fe	stuff \015	
		stuff \033OM	(A)

特殊な端末機能

以下の表では、*screen* は認識するが、 **termcap**(5) のマニュアルには含まれていない端末機能をすべて示す。これらの機能は (‘/etc/termcap’ の) termcap エントリに置いたり、 screenrc ファイルの ‘termcap’, ‘terminfo’, ‘termcapinfo’ コマンドで使える。これらの機能を terminfo データベースには入れるのは、不可能な場合が多い。

LP *(bool)* 端末は VT100 形式のマージン (‘magic マージン’) を持つ。 *screen* は標準の ’xn’ の方を使うようになったので、この機能は obsolete になった。

Z0 *(str)* 幅を 132 カラムに変更する。

Z1 *(str)* 幅を 80 カラムに変更する。

WS *(str)* ディスプレイをリサイズする。この機能は指定する幅と高さを引き数に取る。*SunView(tm)* における指定例: '\E[8;%d;%dt'

NF *(bool)* 端末はフロー制御を必要としない。アプリケーションに直接 ^S と ^Q を送信する。'flow off' と同じ意味である。この反対の機能は 'nx' である。

G0 *(bool)* 端末は ISO 2022 のフォント選択シーケンスを使用できる。

S0 *(str)* 文字集合 'G0' を指定した文字集合に切り替える。デフォルト値は '\E(%.' である。

E0 *(str)* 文字集合 'G0' を標準の文字集合に戻す。デフォルト値は '\E(B' である。

C0 *(str)* 文字列をフォント '0' の変換テーブルとして使用する。詳しくは 'ac' 機能を参照すること。

CS *(str)* カーソルキーをアプリケーションモードに切り替える。

CE *(str)* カーソルキーを通常モードに戻す。

AN *(bool)* autonuke をオンにする。詳しくは 'autonuke' コマンドを参照すること。

OL *(num)* 出力バッファの上限値を設定する。詳しくは 'obuflimit' コマンドを参照すること。

KJ *(str)* 端末で使う漢字の種類を設定する。有効な文字列は 'encoding' コマンドの解説を参照すること。

AF *(str)* ANSI 準拠の方法で文字の前景色を変更する。この機能はほぼ常に '\E[3%dm' (terminfo を使うマシンでは '\E[3%p1%dm') に設定される。

AB *(str)* これは 'AF' と同様だが、背景色を変更する。

AX *(bool)* ANSI セットのデフォルトの前景色/背景色 (\E[39m / \E[49m) を認識する。

XC *(str)* 現在のフォントに基づく、複数個の文字から文字列への変換を記述する。詳細は次の節に書く。

XT *(bool)* 端末は xterm の特殊シーケンス (OSC, マウス追跡) を理解する。

C8 *(bool)* 端末は、高輝度色の表示にボールドを必要とする (例 Eterm)。

TF *(bool)* 欠けている機能を termcap/terminfo のエントリに追加する (デフォルトで設定されている)。

文字の変換

screen には、現在のフォントや端末の種類に基づいて、文字(列)を任意の文字列に変換する強力な機構がある。この機能を用いれば、あまり使わない文字群がいろいろな国・言語のフォントページに散っているような端末でも、通常の標準的な文字集合 (例えば ISO8851-latin1) を使える。

文法:

XC=<*charset-mapping*>{**,,**<*charset-mapping*>}
<*charset-mapping*> := <*designator*><*template*>{**,**<*mapping*>}
<*mapping*> := <*char-to-be-mapped*><*template-arg*>

ブレース内のものは任意の数だけ指定できる。

<*charset-mapping*> は、フォント指示子 <*designator*> ('B': ASCII, 'A': イギリス語, 'K': ドイツ語、など) に含まれる文字を、文字列にマップさせるやり方を *screen* に伝える。それぞれの <*mapping*> では、 1 つの文字をどんな文字列にマップさせるかを記述する。大抵の場合、各コードの大部分は共通なので、ここではテンプレート機構が用いられる (例えば、別の文字集合に切り替える文字列など)。 <*template*> に '%' があると、それはすべてその文字と同時に指定された <*template-arg*> に置き換えられる。指定したい文字列が全く似ていない場合には、'%' をテンプレートとして用い、文字列全体は <*template-arg*> に置くこと。'%' そのものも使えるように

するため、クォート機構も後に追加された。'\' は特殊文字 '\', '%', ',' をクォートする。

以下に指定例を示す:

```
termcap hp700 'XC=B\E(K%\E(B,\304[,\326\\\\,\334]'
```

これは ISOlatin1 (文字集合 'B') のウムラウト付きの大文字を、ドイツ語の文字集合を持つ HP700 端末向けに変換する方法を *screen* に指定する。'\304' は '\E(K[\E(B' に変換される (以下同様)。この行は内部参照テーブルが構築されるまでに * 3 回 * 展開されるため、 1 つの '\' を生成するにはいくつものクォートが必要になる点に注意すること。

さらに多くのエミュレーションができるように、別の拡張も加えられている: クォートされていない '%' がマッピングで変換された場合、 *screen* がその *<designator>* に切り替わると、必ずこれが端末に送られる。この特殊な場合には、テンプレートは単に '%' とみなされる。文字集合切り替えシーケンスと文字マッピングは、通常あまり共通部分を持たないからである。

以下の指定例は、この拡張の使い方の 1 つを示したものである:

```
termcap xterm 'XC=K%,%\E(B,[\304,\\\\\326,]\334'
```

この例では、ドイツ語 ('K') 文字集合の一部が xterm 上でエミュレートされる。 *screen* が文字集合 'K' に切り替わらなければならない場合、'\E(B' が端末に送られる。つまり、 ASCII 文字集合がドイツ語文字集合の代わりに使われる。テンプレートは '%' だけになり、したがってマッピングは単純である: '[' は '\304' に、'\' は '\326' に、']' は '\334' にマッピングされる。

環境変数

COLUMNS	端末のカラム数 (termcap エントリより優先される)。
HOME	ファイル .screenrc を探すディレクトリ。
LINES	端末の行数 (termcap エントリより優先される)。
LOCKPRG	画面をロックするプログラム。
NETHACKOPTIONS	nethack オプションを有効にする。
PATH	実行するプログラムの探索に用いられる。
SCREENCAP	端末の TERMCAP 値のカスタマイズ用。
SCREENDIR	別のソケットディレクトリ。
SCREENRC	別のユーザ用 screenrc ファイル。
SHELL	ウィンドウをオープンした時のデフォルトのシェルプログラム (デフォルト値は ``/bin/sh'')。
STY	別のソケット名。
SYSSCREENRC	別のシステム用 screenrc ファイル。
TERM	端末名。
TERMCAP	端末記述。
WINDOW	ウィンドウの番号 (生成時)。

ファイル

.../screen-4.?.??/etc/screenrc	
.../screen-4.?.??/etc/etcscreenrc	*screen* の配布パッケージに含まれている、プライベート用・グローバル用各初期化ファイルの例。
$SYSSCREENRC	
/etc/screenrc	*screen* の初期化コマンド群を含む。
$SCREENRC	
$HOME/.screenrc	/etc/screenrc の後に読み込まれる
$SCREENDIR/S-<login>	
/var/run/screen/S-<login>	ソケットディレクトリ (デフォルト)

/usr/tmp/screens/S-<login>	別のソケットディレクトリ
<socket directory>/.termcap	"termcap" 出力関数が書き込む
/usr/tmp/screens/screen-exchange または /tmp/screen-exchange	
	screen の「プロセス間通信バッファ」
hardcopy.[0-9]	hardcopy 関数が生成する画面イメージ
screenlog.[0-9]	log 関数が生成するログファイル
/usr/lib/terminfo/?/* または /etc/termcap	
	端末機能データベース
/var/run/utmp	ログイン記録
$LOCKPRG	端末をロックするプログラム

関連項目

termcap(5), utmp(5), vi(1), captoinfo(1), tic(1)

著者

最初に作成したのは Oliver Laumann であり、最新版を作ったのは Wayne Davison, Juergen Weigert, Michael Schroeder である。

COPYLEFT

Copyright (C) 1993-2003
Juergen Weigert (jnweiger@immd4.informatik.uni-erlangen.de)
Michael Schroeder (mlschroe@immd4.informatik.uni-erlangen.de)
Copyright (C) 1987 Oliver Laumann
This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2, or (at your option) any later version.
This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
You should have received a copy of the GNU General Public License along with this program (see the file COPYING); if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA

貢献者

Ken Beal (kbeal@amber.ssd.csd.harris.com),
Rudolf Koenig (rfkoenig@immd4.informatik.uni-erlangen.de),
Toerless Eckert (eckert@immd4.informatik.uni-erlangen.de),
Wayne Davison (davison@borland.com),
Patrick Wolfe (pat@kai.com, kailand!pat),
Bart Schaefer (schaefer@cse.ogi.edu),
Nathan Glasser (nathan@brokaw.lcs.mit.edu),
Larry W. Virden (lvirden@cas.org),
Howard Chu (hyc@hanauma.jpl.nasa.gov),
Tim MacKenzie (tym@dibbler.cs.monash.edu.au),
Markku Jarvinen (mta@{cc,cs,ee}.tut.fi),
Marc Boucher (marc@CAM.ORG),
Doug Siebert (dsiebert@isca.uiowa.edu),
Ken Stillson (stillson@tsfsrv.mitre.org),
Ian Frechett (frechett@spot.Colorado.EDU),
Brian Koehmstedt (bpk@gnu.ai.mit.edu),
Don Smith (djs6015@ultb.isc.rit.edu),
Frank van der Linden (vdlinden@fwi.uva.nl),
Martin Schweikert (schweik@cpp.ob.open.de),
David Vrona (dave@sashimi.lcu.com),

E. Tye McQueen (tye%spillman.UUCP@uunet.uu.net),
Matthew Green (mrg@eterna.com.au),
Christopher Williams (cgw@pobox.com),
Matt Mosley (mattm@access.digex.net),
Gregory Neil Shapiro (gshapiro@wpi.WPI.EDU),
Johannes Zellner (johannes@zellner.org),
Pablo Averbuj (pablo@averbuj.com).

バージョン

これはバージョン 4.0.2 である。これは Wayne Davison による 2.3PR7 カスタム版をマージしたものと、 Oliver Laumann のバージョン 2.0 へのいくつかの改良とがもとになっている。 2.x の番号が付いているバージョンはすべて Oliver Laumann の著作物である。

入手方法

screen の最新の公式リリースは、gnudist.gnu.org, nic.funet.fi その他の *GNU* 配布サイトから anonymous ftp で取得できる。 *screen* のホームサイトは ftp.uni-erlangen.de のディレクトリ pub/utilities/screen である。ここのサブディレクトリ 'private' には、最新のベータテスト版が置かれる。開発に協力したい場合は、気付いた点を screen@uni-erlangen.de に送ってほしい。

バグ

- 'dm' (削除モード)と 'xs' を正しく扱うことができない (これらは無視される)。'xn' は magic-margin 指示子として扱われる。
- *screen* は 2 倍高や 2 倍幅の文字を扱うことができない。しかし、 *vttest* の失敗が許されるのはここだけである。
- 異なる端末タイプから再アタッチを行ったとき、環境変数 $TERMCAP を変更できない。
- terminfo ベースのシステムのサポートが非常に限られている。機能を $TERMCAP に追加しても全く効果が現われないことがある。
- *screen* はハードウェアタブを利用していない。
- *screen* は所有者 root に setuid してインストールしなければならない。これは各ウィンドウに対する tty デバイスファイルの所有者を正しく変更するためである。''/etc/utmp'' ファイルに書き込みを行うためにも特殊な権限が必要である。
- *screen* が SIGKILL で kill された時には、''/etc/utmp'' のエントリが削除されない。この場合、一部のプログラム ("w" や "rwho" 等) は、実際にはログインしていないユーザをログイン中であると表示する。
- ユーザが使っている tty に utmp エントリが無いとき、 *screen* はおかしな警告を出す。
- モデムラインのデバイスドライバが HANGUP シグナルを送るように設定されていないと、このラインがハングアップした時に *screen* は自動的なデタッチ (あるいは終了) を行えないことがある。 *screen* セッションをデタッチするにはコマンドラインオプション -D または -d を用いること。
- パスワードを設定しているときでも、コマンドラインオプション -D および -d によって、パスワードを尋かれることなしにデタッチができてしまう。
- ''breaktype'' と ''defbreaktype'' の両方ともが、全端末デバイスで用いられるブレークの生成方法を変更してしまう。前者はそのウィンドウの設定だけを変更すべきで、後者は以降新規に生成されるウィンドウの設定だけを変更すべきである。
- マルチユーザセッションにアタッチしようとすると、そのユーザの .screenrc ファイルは評価されない。各ユーザの個人設定は、そのセッションがブートされるときの .screenrc ファイルに書くか、あるいは手動で変更しなければならない。
- 全機能を最大限に生かすためには、おかしな想像力が非常に有用となる。

- バグ報告、修正、改良、T シャツ、お金、ビール、ピザ等は **screen@uni-erlangen.de** まで送ってほしい。