Screen shots for yume3 (yume) easy custom menus program

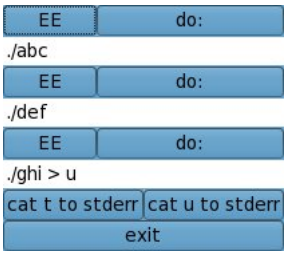
The yume distribution provides an **examples/** directory that contains a few dozen examples of yume usage. This web page lists most of those examples, via screen shots, brief explanations, and links to example sources and explanatory files.

If you create useful or exemplary yume scripts yourself, please feel free to contribute them for inclusion in the **examples/** directory. Send email to yume at pat7.com. Subject must begin with the word **yume**.

The following links take you to the indicated sections within this document.

Example screen shots and brief explanations

- **abc-def-ghi : Redirection example**



Link to code
abc-def-ghi contains a yume command of the form "yume ... > t", which causes standard output from yume and its abc and def subprocesses to be directed into file t. (Standard output from ghi is directed into file u.) abc, def, and ghi are shell scripts with commands like "`echo This is def at `date``". Each time abc or def executes, the output is appended to file t. File t can be displayed (on the console where abc-def-ghi was started) via the "cat t to stderr" button. Each time ghi executes, the output overwrites file u. File u can be displayed via the "cat u to stderr" button.

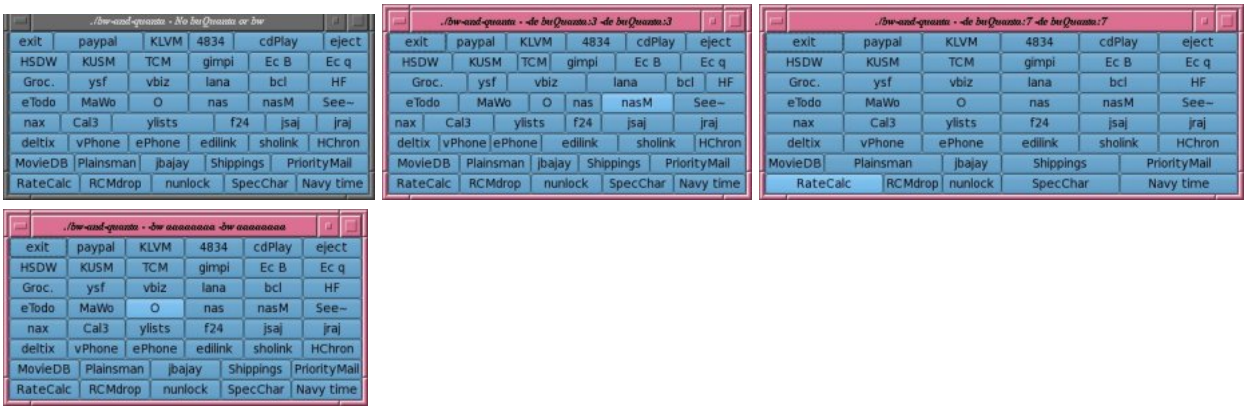
Return to top

- **button-widths : Control of relative-widths example**

Link to code
button-widths contains three yume commands illustrating the -bw (button widths) switch without a -at switch. The generated menus are shown above (at 2/3 scale), within frames that show their window titles. (Frame appearance is window-manager-dependent.) Menus 1 and 2 each use one -bw switch, with parameter "dc", which in a line of two buttons gives the first button 4/7 of the line width, and other button 3/7 of the line. (In the absence of a -at switch, overall menu width is governed by the length of the longest line, which depends on its number of buttons, the font, and the text content.) Menu 3 has a "-bw dc" switch near its beginning, and a "-bw bbba" just before the last line. In this line of four buttons, the first three each get 2/7 of the line width, and the other button gets 1/7.

Return to top

- **bw-and-quanta : -at, -bw, -de, and -in examples**



Link to code
Link to explanatory note

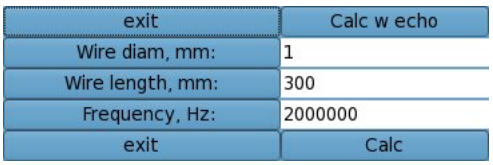
bw-and-quanta contains four yume commands to produce four menus (shown above at 2/3 scale), demonstrating effects of different sets of button-size controls. All the menus have the same buttons (as listed in the include-file bw-and-quanta.in), but each uses a different combination of controls:

- The 1st menu has: no buxMargin or buQuanta or -bw settings.
- The 2nd menu has: -de buxMargin:50 -de buQuanta:50
- The 3rd menu has: -de buxMargin:10 -de buQuanta:1000
- The 4th menu has: -bw aaaaaaaa

The title of each menu-window shows which button-size controls it uses.
Please see the explanatory note for further information about this example.

Return to top

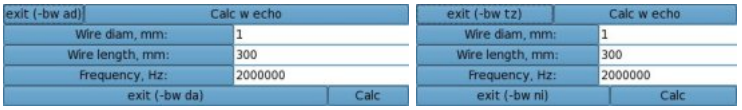
- **calc-wrap : -iv example**



Link to code
Link to explanatory note
calc-wrap contains a yume command illustrating use of -iv fields to set environment variables. It makes a gui wrapper for a command-line-style calculation program, with edit-boxes for parameter values and a click button to run the program.
Please see the explanatory note for further information about this example.

Return to top

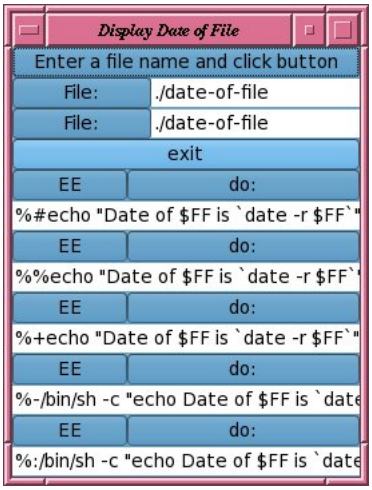
- **calc-wrap2 : -at and -bw examples**



Link to code
Link to explanatory note
calc-wrap2 demonstrates several -bw (button-width) settings. calc-wrap2 creates 8 menus (two of which are shown above at 2/3 scale), all having the same sets of buttons, but with various ratios of sizes and absolute sizes. It also displays a few comments and a list of -at and -bw parameter values for the menus it created.
Please see the explanatory note for further information about this example.

Return to top

- **date-of-file : Meta-mode-controls example**



Link to code
Link to explanatory note

date-of-file contains commands illustrating yume's meta-mode controls. A meta-mode control is a percent sign and a code character at the beginning of a command, by which yume's usual method of command invocation can be altered. As noted in yume(1), a command from a yume menu is executed by a shell via a forked copy of yume, unless the command begins with a percent sign and a meta-mode code character, in which case the invocation method depends on that code character.

For details, please see explanatory note and yume-meta(1) [text] or yume-meta(1) [pdf]

Other meta-mode examples include test-y-2, test-y-4, and xterms-w-sh.

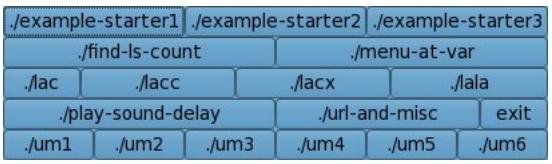
Return to top

- ## errors-demo : Error-messages example

[This example produces console output, no images.]
Link to code
Link to explanatory note
errors-demo displays examples of error codes that occur when yume is invoked incorrectly. The explanatory note has a few more comments and shows the output as of March, 2009.

Return to top

- ## example-starter1



Link to code
example-starter1 was an early version of a yume menu to start yume examples. It has a fixed list of buttons. It is an example of a small menu with several rows of buttons.

Return to top

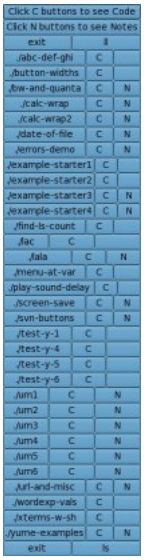
- **example-starter2**

[Image shown half-scale and truncated for brevity]
Link to code
example-starter2 was the second version of a yume menu to start yume examples. It makes a menu with one button for each executable file in the current directory -- several buttons in the menu shown above are for executables that are not yume menu scripts (abc, in-test-1, latex-2err, etc).

example-starter2 and later example-starter versions provide examples of menus created on-the-fly from strings in environment variables.

Return to top

- **example-starter3**



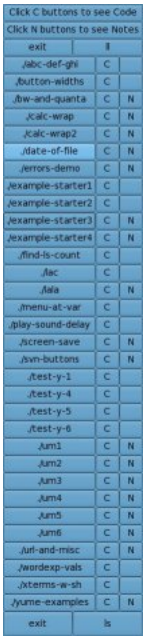
Link to code
Link to explanatory note
example-starter3 was the third version of a yume menu (shown half-scale) to start yume examples. It uses strings in environment variables to create its menu dynamically, making one line of buttons for each qualified file in the current directory. A file is qualified if it's executable and contains "yume example" anywhere in it, or "yume" first on a line of the file. The left-hand button in a menu line executes an example; the middle button displays the code of the example; and the right-hand button displays an explanatory note (if available) or is a no-op.

example-starter3 uses button margins and quanta sizes in an attempt to make buttons of uniform size. Note that **example-starter4**, the next example, uses -bw options rather than margins and quanta, to generate buttons that are pleasingly more uniform.

Please see the explanatory note for comments on using **echo** and **xargs** for proper quoting treatment of dynamic menu strings.

Return to top

- ## example-starter4



Link to code
Link to explanatory note
example-starter4 is the current version of a yume menu (shown half-scale) to start yume examples. It differs from **example-starter3** in using -bw options rather than margins and quanta, to generate buttons of more-uniform width.

Return to top

- ## find-ls-count : xclip-with-yume example

Link to code
find-ls-count contains a yume command with an example of using xclip (a unix utility) to communicate with yume. The general idea is that you mouse-select some text and drop it on a yume button, whose command uses xclip to get the text and use it in a command. In this example, the 'Find filename' button will list all the filenames in the current directory that include the selected text.

Here are a few comments about elements of this script: The

```
-at 100x400+77+7
```

switch tells yume to locate a 100x400-pixel window's top left corner at (77,7), which will be quite near the top of your display, not far from the left edge. The switch sequence

```
-bu -la 'Find filename' \
"V=\`xclip -o\`; echo \"Finding \$V in $D\"; find . | grep \$V"
```

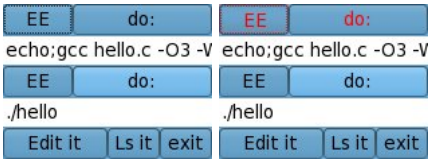
starts a button row with a button labeled "Find filename" that runs the double-quoted command shown. Note that two double-quote characters within the command are backslash-quoted so they apply when yume executes the command, rather than when yume starts. Likewise, the dollar sign (in the `\$V` instances) is backslash-quoted to delay `$V` evaluation.

The value of `` `xclip -o` `` (that is, left-quoted `xclip -o`) is current clipboard contents (ie, whatever text was most-recently selected). When yume processes the whole command, the shell runs `xclip -o` and assigns the value to V. The `echo` command displays the selected text and the current-directory name. The `find` command finds all the filenames in and beneath the current directory and pipes them to `grep`, which selects those that contain the selected text.

Note, additional examples using `xclip` appear in **url-and-misc**. (The same examples appear individually in um4, um5, and um6.)

Return to top

- ## lac : C compilation aid



Link to code
lac contains a yume command that creates a menu with several commands to compile, run, or edit a C program. By default, **lac** targets the most-recently-modified .c file, but you can give **lac** any [.c] file name when you start it. Note, if *fn* stands for a filename, both of "**lac** *fn*" and "**lac** *fn*.c" act the same, using *fn* as the base name in the menu.

If you click the EE button at the left end of the top row, the text of the EE and `do:` buttons turns red, as shown in the second screen shot. This shows that "entry execution" is turned on and that the compilation command in the text-box will execute whenever the mouse cursor crosses the `do:` button, rather than only when you click the `do:` button. (Clicking the EE button toggles EE state from on to off or from off to on.) (The color change mentioned above assumes that a default version of file ~/.yume-gtkrc has been installed; if not, colors will differ.)

The `echo` command before the gcc command types a blank line, to tell the user the compile has begun; the `echo` command after gcc should produce a beep and blank line when the compile finishes. Compile errors, if any, will be listed between the two blank lines. [Note, in the `echo -e '\a'` part of the command, `\a` stands for bell. Depending on your shell, a bell may or may not sound.]

Before it makes a menu, **lac** tests the existence of half-a-dozen variables that specify window title, compiler name, compiler options, editor name, source file extension, and `make` setting. If these variables (YTITL, YGCC, YOPTS, YEDIT, YEXT, and YMAKE) have not been set, **lac** assigns them default values: lac, gcc, -O3 -Wall, $EDITOR, .c, and nil.

As just noted, when YEDIT isn't set, YEDIT defaults to $EDITOR. If in turn EDITOR isn't set, YEDIT then defaults to emacs. If you ordinarily use some other text editor, set EDITOR appropriately. For example, with vim and bash, put the first of the following lines in file ~/.bashrc; or, with vim and csh, put the second line in ~/.cshrc :

```
export EDITOR=vim
setenv EDITOR vim
```

In addition to the YOPTS option settings (which are `-O3 -Wall` by default) **lac** may add a `-lm` switch to the compilation command that it generates. This switch (referred to as MATHLIB in **lac** code) is set if grep finds "`<math.h>`" or any of certain math functions (sqrt, exp, log, etc) within the .c source file. If you add math library references to your code after invoking **lac**, you can either run **lac** again, or type `-lm` within the **do:-** item's text box. That is, in case of special circumstances, you can manually edit any of the text boxes as you like. But if you find yourself frequently having to do manual edits, then either change **lac** or create a script to change it. The next two paragraphs describe two possible customizations of **lac** via a script. Both of these scripts appear in **yume**'s **examples/** directory.

First, note that the **lacc** utility exports `YTITL=$0, YGCC="g++", YEXT=".cc", and YMAKE=make`. Then it invokes **lac**. This creates a C++ compilation aid instead of a C compilation aid. The created menu has an additional `do:` group with a `make` command, as shown in red in the **lacc** screen shot below.

Secondly, the **lacx** utility exports `YTITL=$0, YEDIT=x2, YOPTS="-O3 -Wall -lX11 -lforms -lXpm", and YMAKE=make`. Then it invokes **lac**. This creates a C compilation aid suitable for compiling **yume** and other xforms-based programs. (Remove or edit the `YEDIT=x2` line unless x2 is an alias for your text editor.)

Return to top

- ## lacc : C++ compilation aid

Link to code
lacc uses **lac** to create a menu with several commands to compile, make, run, or edit a C++ program. By default, **lacc** targets the most-recently-modified .cc file, but you can give **lacc** any [.cc] file name when you start it. For example, suppose you want to work with xyz.cc and it isn't the most-recently-modified .cc file. Instead of starting **lacc** with the command "`lacc`", start it with "`lacc xyz`" or with "`lacc xyz.cc`".

To locate the most-recently-modified .cc file, **lacc** uses shell code like

```
FILE=`ls -t *.cc | head -1`
```

(Note, other scripts like **lac** and **lala** use similar shell code, but with ".c" or ".tex' rather than ".cc".)

The `ls -t *.cc` portion of this code makes a list of current-directory filenames that end with .cc, sorted by modification time; the `head -1` portion selects the first element from that list. If the current directory has no filenames that end with .cc, error messages like

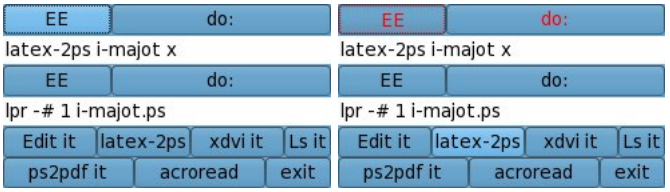
```
ls: cannot access *.cc: No such file or directory
egrep: .cc.cc: No such file or directory
```

will appear, and the resulting menu will refer to "filenames" like .cc and .cc.cc.

Also see discussion of **lac**.

Return to top

- **lala : Latest-Latex-file Latex aid**



Link to code
Link to explanatory note
lala contains a yume command that creates a menu with several commands for processing latex files. By default, **lala** targets the most-recently-modified .tex file, but you can give **lala** any file name when you start it. For discussion of "most-recently-modified" files, please see topic **lacc**.

Clicking the EE button at the left end of the top row toggles the text color of the EE and `do:` buttons from black to red (or vice versa) as shown in the second screen shot. The red color means that "entry execution" is turned on and that the command in the text-box will execute whenever the mouse cursor crosses the `do:` button.

If you are using `xdvi` to view your `latex` output, and have entry execution turned on for the `latex-2ps` command, you can make and save a change to your .tex file, then move the mouse cursor across the red `do:` button, and after you hear a beep [indicating that `latex`, `dvi2ps`, and font generation (if needed) are done], move the mouse cursor into the `xdvi` window. `xdvi` will detect that the .dvi file it is displaying has changed, and will display the new form of your document. (Whether beep works is shell dependent.)

Note, the `latex-2ps` script uses `latex` to process a .tex file. latex-2ps runs latex invisibly unless an error occurs. It beeps when done if `latex` finishes without error; but if a `latex` error occurs, it uses script `latex-2err` to present selected latex error log messages in an xterm window. In other words, if `latex` and `dvi2ps` processing goes ok, `latex-2ps` beeps and is done; but if that processing has errors, `latex-2ps` invokes the `latex-2err` script to display some of the logged error messages in an xterm. `latex-2err` uses grep to suppress some `latex` error messages. For example, it squeezes out messages containing "No pages" and "Overfull", among others.

Script `latex-2ps` is used to `latex` the specified tex file if it's newer than its dvi file, or if an extra parameter is given after the file name. The "x" at the end of each `latex-2ps` command just before the blue-line text-cursor mark in each screen shot is just such an extra parameter, used to force `latex` and `dvi2ps` processing regardless of file dates. You may remove the x if you wish.

Return to top

- **menu-at-var : Variable-location-of-menu example**

Link to code
menu-at-var contains a yume command that creates a one-button menu, near the top right corner of the screen. The exact location of the button depends on the hostname, as described in text in menu-at-var code.

The button pictured above was created on qili, and when it is clicked it opens an xterm that runs on qili and is displayed on the current display.

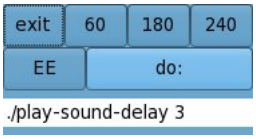
Note, `menu-at-var` is mostly just a simple example of placing a menu at different locations for different cases, rather than a useful menu. In practice, one would use a slightly longer `xterm` command, for example like:

```
yume -bu -la yuli "xterm -e ssh yuli&" &
```

which when run on qili would open a yuli xterm and display it on the current display.

Return to top

- **play-sound-delay : Script example that plays a sound after a delay**



Link to code
play-sound-delay contains a yume command to create a menu as shown above. When you click one of the numbered buttons, a sound will be played after a delay of that many seconds. If you click the `do:` button (with the command as shown) a sound will be played after a delay of 3 seconds.

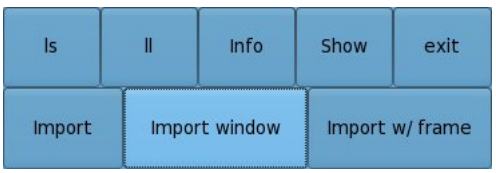
Further discussion of **play-sound-delay** appears in its code file. But very briefly, the script structure is like:

```
[ -z $1 ] && {
   <yume command>
   exit;
}
<command to sleep $1 seconds, then play sound via play-sound-delay>
```

The `[ -z $1 ] && { ... }` sequence says to test if parameter 1 has zero length, ie, is missing. If it's missing, the `yume` command gets executed -- creating a menu that contains recursive calls to **play-sound-delay**, as shown above -- and the script exits. But if parameter 1 is present, it is treated as a number of seconds to sleep, after which a sound is played.

Return to top

- **screen-save : Save screen-shots utility**



Link to code
Link to explanatory note

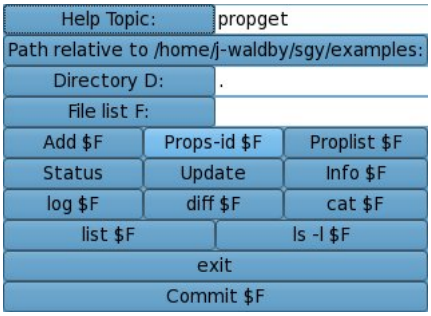
screen-save contains a yume command to create a menu with buttons labeled as shown above [that is, ls, ll, Show, exit, Import, Import window, and Import w/ frame] for saving screen-shots sequentially in a screen-saver directory. The explanatory note contains a page of information about using the screen-save menu and adapting it to your own preferences.

Note, some useful adaptions can be made simply by supplying parameters to the **screen-save** invocation. See comments near the beginning of its code. For example, the pictures for this web page have titles like `eg-ss-10.jpg, eg-ss-11.jpg,` and so forth. These pictures were saved using a **screen-save** menu that was started by the command, "`screen-save . eg-ss- 1`", which saves pictures in current directory, gives them a prefix of "eg-ss-", and numbers them with an interval of 1.

Note, the **screen-save** menu uses the **screen-save** script recursively to do different imports, supplying a fourth parameter to control what the script does. For a simpler example and brief explanation of this technique, please see the closing paragraphs of the discussion of **play-sound-delay**.

Return to top

- ## svn-buttons : SVN utility aid



Link to code
Link to explanatory note
svn-buttons makes a yume menu with buttons to execute various svn commands.

Commands for the Subversion version control system typically have the form "`svn V F`" where V is a verb and F is a filelist. Examples: "`svn log myfile.c`" and "`svn cat myfile.c`"

Several commands have the form "`svn V`"; for example, "`svn status`" and "`svn update`".

To tell the menu where your svn-versioned files are located, type the directory name into the "Directory D:" textbox. (This will set environment variable D within yume's environment.) The path you type into the textbox should be relative to the path shown on the line above it. (The screen-shot path has been blurred out.)

To tell the menu the filelist that commands should apply to, type the filenames into the "File list F:" textbox.

To execute an svn command with the verb add, proplist, status, update, info, log, diff, cat, list, or commit, click the correspondingly-named button. Then svn will apply the verb to your specified filelist.

As you can see in the **svn-buttons** code, the "`Props-id $F`" button will run the command sequence

```
cd $D; svn propset svn:keywords "Date Id" $F
```

which will tell svn to enable expansion of $Date$ and Id sequences within the specified files.

The "`ls -l $F`" button will run the command sequence

```
cd $D; ls -l $F
```

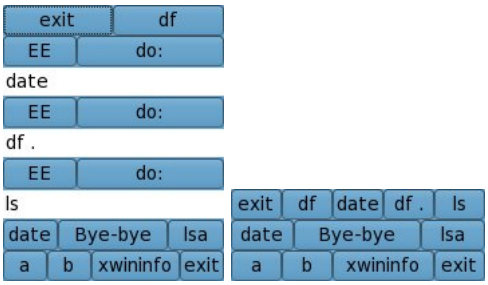
which will show directory details of the files in the filelist.

The Subversion system comes with help files that can be accessed by commands of the form "`svn help V`", where V is an svn verb or an svn help topic. To display help text on the console where you ran **svn-buttons**, put the topic name in the "Help Topic:" textbox and click the "Help Topic:" button.

See explanatory note for further comments, if you don't want `EDIT=emacs` when an svn commit is done. (EDIT controls the editor that svn invokes for the user to record comments upon commit.)

Return to top

- ## test-y-1 : Otherly-labeled "exit" button



Link to code
test-y-1 contains two yume commands to create two nearly-equivalent menus as illustrated above. These menus are nothing special but are part of an informal yume regression-test suite.

Note, however, sequences like "`-la 'Bye-bye' -ex`" to apply a label other than "exit" to an exit button.

Return to top

- ## test-y-2 : %+ metacommand

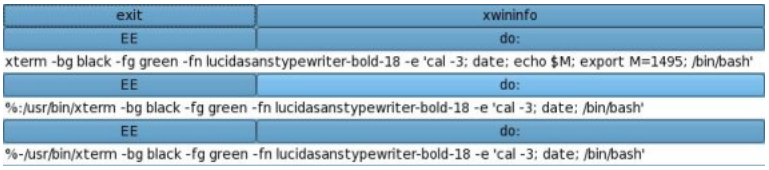
Link to code
test-y-2 contains a yume command to create a menu as illustrated above. The command string associated with the "**d+du**" button is "**%+date;du .**", which first exits yume, then displays the date on the console, then displays the result of "**du .**" on the console. The command string behind the other button is similar, with "**df .**" in place of "**du .**".

Note, the menu goes away if either button is used.

Other meta-mode examples include date-of-file, test-y-4, and xterms-w-sh.

Return to top

- ## test-y-4 : Special-purpose xterms; %: and %- metacommands



Link to code
test-y-4 contains a yume command to create a menu as illustrated above (at 3/4 scale). The command in the top textbox creates an xterm, displays a three-month calendar and current date and variable M in it, then sets M and starts a shell.

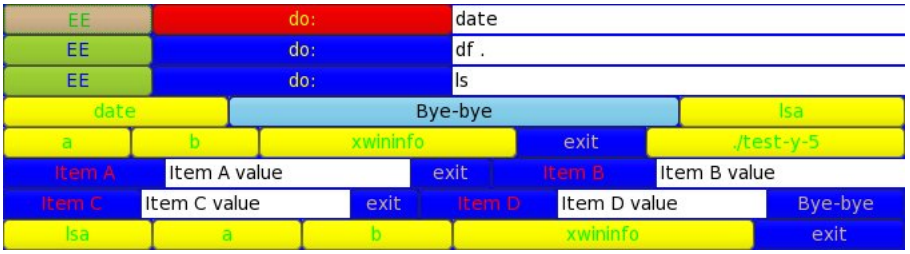
In like manner, one can create xterms set up for particular purposes.

The other textboxes create equivalent xterms, aside from not showing or setting M. The command beginning with **%:** is executed by invoking /usr/bin/xterm (within a forked copy of yume) directly rather than invoking /bin/sh (within a forked copy of yume) to invoke xterm. The command beginning with **%-** invokes /usr/bin/xterm directly from yume, without forking. Hence the yume menu disappears when the xterm opens in response to that command.

Other meta-mode examples include date-of-file, test-y-2, and xterms-w-sh.

Return to top

- **test-y-5 : Different-colored buttons example**



Link to code
test-y-5 contains a yume command illustrating use of the `-de gtkrc` option switch to select a particular gtkrc file, and in that file, setting the colors that will be used for buttons in a yume menu. The commands within this menu are not particularly useful or interesting, but merely serve to populate a menu so that button colors can be observed.

A particular gtkrc file is selected by a yume option of the following form: `-de gtkrc:nameoffile` . For example, test-y-5 says:

```
yume ...  -de gtkrc:./yume3-gtkrc-5A ...
```

Note, if your home directory contains file `.yume-gtkrc` , settings from that file normally are processed by gtk_init() before settings from a -de specified gtkrc file are processed. You can suppress processing of `~/.yume-gtkrc` via: `-de nogtkrc` . The nogtkrc flag does not suppress files mentioned by `-de gtkrc:` , however.

In the gtkrc file, foreground and background colors can be specified for various widgets in various states. For example, the EE buttons' normal font color and background color in this example are given by:

```
style "y3ee" = "y3m" { fg[NORMAL] = "blue"
                       bg[NORMAL] = "YellowGreen" }
```

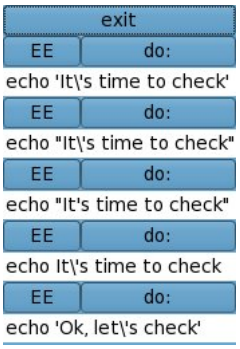
Style y3m, found at the beginning of yume3-gtkrc-5A, defines a base-level color scheme for yume buttons. Styles for specific buttons can override whatever the user wants to be different from the base-level scheme, in this case normal-status text color and button background color.

In this example, the topmost EE button has been clicked. Thus, that EE button is in Active state (for which, bg[ACTIVE] is "tan", from y3m style), rather than Normal. When an EE button is Active, its associated do: button is named "y3dr" (for which, bg[NORMAL] is "red2") rather than "y3dp" (for which, bg[NORMAL] is "blue", from y3m style). When the screenshot was taken, the mouse was over the large 'Bye-bye' button, which is sky-blue because y3m specified bg[PRELIGHT]="SkyBlue".

Note, the top three rows in this menu are -do groups in -dl form. The middle two lines and the last line are ordinary -bu buttons. The "Item A"..."Item D" elements are from -iv shell-variable items.

Return to top

- **test-y-6 : Single-, Double-, and Backslash Quoting examples**



Link to code
test-y-6 contains a yume command illustrating some elementary usage of double quotes, single quotes, and backslash quoting.

The first and last commands in the menu each produce the same two error messages:

```
/bin/bash: -c: line 0: unexpected EOF while looking for matching `''
/bin/bash: -c: line 1: syntax error: unexpected end of file
```

(on a Linux system with /bin/sh linked to /bin/bash).

These error messages are mentioned in the "Dealing with errors in shell commands" subsection of the "DIAGNOSTICS, WARNINGS, CAVEATS" section of yume(1). The "file" referred to in the error messages is just the command string that yume passed to /bin/sh.

In unquoted shell strings, backslash is an escape character that quotes the character after it. When it appears within a quoted string, backslash most commonly is treated as a literal character: always literal within a string quoted by single-quotes, else literal unless next character is "$", "\", or "`" and sometimes "!". See sh(1).) Thus, in the two commands with errors, the first single-quote matches the one just after the backslash. The text from there to end of line isn't quoted text. The single-quote at end of line begins a new section of quoted text, so sh begins looking for a matching single-quote; but as nothing follows, errors occur.

Return to top

- **um1 : Open URL in Netscape example**

Link to code
Link to explanatory note
um1 illustrates yume menu buttons that open specific URL's in Netscape.

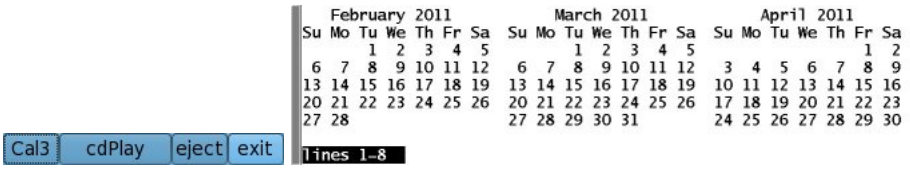
Return to top

- **um2 : Open URL in Firefox example**

Link to code
Link to explanatory note
um2 illustrates yume menu buttons that open specific URL's in Firefox.

Return to top

- **um3 : xterm cal -3, cdplay, eject examples**



Link to code
Link to explanatory note
um3 illustrates yume menu buttons that display a three-month calendar in large print (the screenshot is shown half-scale), and that start cdplay or eject disk from /dev/cdrom.

Note, for further xterm examples, please see **test-y-4**.

Return to top

- **um4 : xclip + firefox examples**

Link to code
Link to explanatory note
um4 illustrates use of xclip for opening clipped URL's or auction numbers. That is, if you drop a URL string on the middle button, the button's command sequence will get the string using xclip; make the URL into a single line, via tr; and then open the resulting URL in Firefox. If you drop an ebay auction number on the right button, the button's command sequence will form a URL to open that auction's webpage in Firefox.

In the above, "drop string on button" means: Left click the button after highlighting or selecting a string of text in some window.

Return to top

- **um5 : H:M:S, dc, and factor examples**

Link to code
Link to explanatory note
um5 illustrates use of xclip together with dc and factor for simple computations on selected numbers.

Per note: "um5's HMS button gets a number (of seconds) from the clipboard, via "xclip -o". It displays the number of minutes, hours, or days that the number equals. um5's Factor button gets a number (or numbers) from the clipboard, via "xclip -o". It displays the factors of the number(s)."

For example, if you select text 12000 and click HMS, the following will display:

```
12000 seconds = 200.0 minutes = 3.33 hours = .13 days
```

or if you click Factor,

```
12000: 2 2 2 2 2 3 5 5 5
```

The note shows some more examples.

dc is a reverse-polish arbitrary precision calculator. The H:M:S script embeds four dc command sequences within

```
echo "$S seconds = `$D"60 1$E"` minutes = `$D"3600 2$E"` hours = `$D"86400 2$E"`
```

where E="k${S}r/p" and D="/usr/bin/dc -e ". In the example with S=12000, the first dc calculation

```
`$D"60 1$E"`
```

expands to

```
/usr/bin/dc -e "60 1k12000r/p
```

which pushes 60; pushes 1; k pops 1 and sets precision to 1 digit; pushes 12000; r reverses 60 and 12000; / divides 12000 by 60; p prints result, 200.0. And so forth.

Return to top

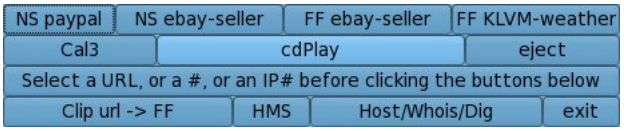
- **um6 : xclip + host, whois, dig example**

Link to code
Link to explanatory note
um6 illustrates use of xclip together with host, whois, and dig.

Per note: "When you drop an IP number or internet machine name on the Host/Whois/Dig button, the script puts the IP number into shell variable IP, then in turn invokes host, whois, and dig to get data about the IP number. Output from each command is filtered by grep -v commands, to suppress some uninteresting, repetitive stuff."

Return to top

- **url-and-misc : um1-um6 composite example**

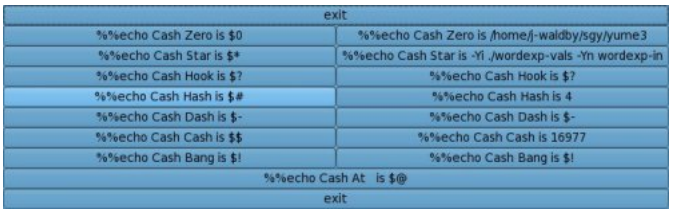


Link to code
Link to explanatory note
Per note: "**url-and-misc** contains several commands, to illustrate that a wide variety of operations can be included in a single menu. For ease of viewing, the buttons of this menu are also broken out into separate files, um1 ... um6, with a Note file for each."
um1: 'NS paypal' and 'NS ebay-seller' buttons -- open web pages in netscape
um2: 'FF ebay-seller' and 'FF KLVM-weather' -- open web pages in firefox
um3: 'Cal3', cdplay, eject -- use right size of xterm for display
um4: 'Clip url -> FF', 'Ebay# -> FF' -- Drop URL or Ebay# on a button
um5: 'HMS' and 'Factor' buttons -- Drop number on button, get results
um6: 'Host/Whois/Dig' button -- Drop IP number on button, get results

---

Return to top

- **wordexp-vals : Shell Special Parameter expressions example**



Link to code

wordexp-vals illustrates use of shell Special Parameter expressions $@, $*, $#, $?, $-, $$, $!, and $0 with yume. If your menus do not use any of these expressions, this example will not be relevant.

Some parts of the following discussion require familiarity with the "Special Parameters" subsection of the "PARAMETERS" section of sh(1).

wordexp-vals contains a yume command like `yume -in wordexp-in`, where `wordexp-in` is a file with yume-parameter entries to make a menu as above (shown 2/3 scale). For each of the seven shell special parameter expressions $*, $#, $?, $-, $$, $!, and $0, `wordexp-in` contains a variant of the sequence

```
-bu '%%echo Cash Zero is $0' "%%echo Cash Zero is $0"
```

with appropriate substitutions for each special parameter other than $0.

Thus, the menu contains seven lines with a pair of buttons on each line and each button label is the same as the command string it executes. For each left button, the button's command is single-quoted (so the special parameter expands at button run time rather than yume run time), while for each righthand button, the button's command is double-quoted (so the special parameter expands only at yume run time). The menu also has a one-button line for $@ and an exit button.

For $0, the left button will display [if /bin/sh is linked to /bin/bash]

```
Y3:  echo Cash Zero is $0
Cash Zero is /bin/bash
```

while the righthand button will display

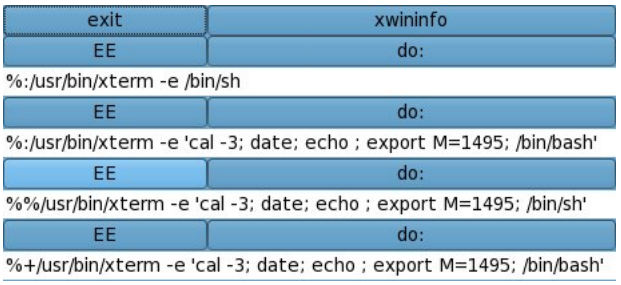
```
Y4: echo Cash Zero is yume
Cash Zero is yume
```

As noted in yume(1) and yume-meta(1), the **%%** meta-mode tells **yume** to print button's command on stdout each time before the command is executed. The number between "Y" and ":" is the internal data structures index number of the button's command.

This example is less complete than it should be, as it doesn't illustrate button-run-time differences between single- and double-quoted instances of $* and $@, nor does it illustrate values when **%-** or **%:** meta-mode is used. In these non-shell meta-mode cases, wordexp [see wordexp(3)] is used to expand shell variables before command invocation.

Return to top

- ## xterms-w-sh : More meta-mode examples



Link to code

The menu presented by **xterms-w-sh** contains `do:` blocks with commands like those shown in test-y-4. Some differences between these examples are:

- **xterms-w-sh** turns on ShoMode = ShoAddObj+ShoWExp, which causes menu objects to be listed on stdout as they are added to internal data structures, and causes parameter expansions to be listed whenever wordexp [see wordexp(3)] is called, as will happen for the **%:** meta-mode commands.
- Xterms opened by **xterms-w-sh** use default fonts, colors, and sizes, rather than specified fonts and colors as in test-y-4.

Note, command strings that begin with meta-mode **%-** or **%+** *terminate the yume menu* when they execute.

Other meta-mode examples include date-of-file, test-y-2, and test-y-4.

Return to top

- ## yume-examples : Small menu to start example-starter4

Link to code
Link to explanatory note
yume-examples makes a yume menu with an exit button and a button labeled "Show list of yume examples" -- a button that runs example-starter4.

A primary reason for **yume-examples** existing is to allow running examples without needing to cd to **yume**'s **examples/** before starting an example-starter. Start **yume-examples** via a command like

```
yume-examples yume/examples/
```

-- that is, use a parameter to specify the location of **examples/**.

Return to top