htsh / wapsh - User Guide

http://www.exolution.de/wapsh
mailto:wapsh@exolution.de

Draft 1.0, last revised 21 Nov 2000

1. Introduction

htsh is a system which allows users to login in to a Unix shell via a web browser or WAP phone (in the latter incarnation, the *htsh* is also called *wapsh*). *htsh* offers the following features:

- **WML (WAP phone) and HTML (Web browser) interfaces**.
- **Line-oriented interface**. Lines and individual characters can be transmitted to the shell, but no screen mode emulation is provided. Thus it is possible to run most programs, except those requiring a screen interface. (The most notable standard Unix programs that this excludes are screen mode editors such as *emacs(1)* and *vi(1)* – it is nevertheless possible to use *ex(1)*, the line mode of *vi*, and to use the original Unix editor *ed(1)*.)
- **Multiple simultaneous login support**. Any number of users can simultaneously login and operate separate shells using *htsh*.
- **Integration with standard login authentication**. The user logs in by supplying their usual username and password. After authentication and initialisation, *htsh* will launch the user's standard shell.
- **User-specific initialisation file**. Each user can have an initialisation file (residing in their login directory) which tailors the appearance and operation of their browser or phone display. Control statements are provided allowing sections of the initialisation file to be processed dependent on the protocol ("wap" or "http") or user agent (browser/phone type) being employed for the remote login.
- **Shortcuts**. In order to save typing on a WAP phone (or in a web-browser), it is possible to create shortcuts for commonly used shell commands. These shortcuts are displayed as a user-selectable menu.
- **Command history**. Shell inputs are saved in a history list, which can be edited and re-executed
- **Control character input**. Menus and buttons are provided to allow special characters, such as *control-C*, *control-D*, and ESCAPE, to be transmitted to the remote shell.
- **Output scrolling and searching**. Since the display capacity of a WAP phone (and to a much lesser extent a web browser) is limited, the result of commands producing a large amount of output cannot be, and is not, displayed in a single step. *htsh* provides facilities for scrolling forward and backward through the generated output and for searching through it.
- **Environment variables made available to login shell**. *htsh* makes environment variables available to the login shell which identify the protocol ("wap" or "http") and user agent (browser/phone type) being employed. This enables shell startup scripts to be tailored according to these values if desired.
- **Optional separation of *HTTP (web) server* and *Login Host***. Normally the *htsh* (HTML +WAP) *HTTP Server* software and the *htsh* server daemon reside on the same host. It is however possible to separate these two components of *htsh* onto different hosts.
- **Single *HTTP Server* can serve multiple *Login Host*s**. A single *HTTP Server* can be configured to allow users to login on multiple *Login Host*s.
- **Secure transmission**. SSL (*https*) is used to encrypt data transmission between the web browser (client PC) or WAP gateway and the *htsh* web server. If the *htsh* web server and the *htsh* server daemon reside on different machines, the server daemon can be configured only to accept transmissions from designated web servers.

2. Terminology

In the following sections, these terms are used:

- **(Client) browser**: a web browser running on a user workstation, or a WML browser operating on a WAP phone.
- **WAP gateway**: a server which provides connectivity between a mobile telephone network and the Internet.
- **HTML**: *Hypertext Markup Language* – the language used to define the layout and operation of web pages displayed on a web browser.
- **WML**: *Wireless Markup Language* – the language used to define the layout and operation of screen displays on a WAP phone.
- **HTTP**: The *Hypertext Transfer Protocol* used for transmission of hypertext data across the Internet.
- **HTTPS**: The *Secure Hypertext Transfer Protocol* which allows information to be transmitted in encrypted form. Use of HTTPS requires installation of an SSL (Secure Sockets Layer) certificate on an *HTTP Server*.
- ***HTTP Server***: (Also known as a *Web Server*.) Software which handles browser requests submitted across the Internet and sends responses (in the form of HTML or WML) to the browser. *htsh* makes use of Apache, the most widely used web server software.
- **HTTP Server Host**: A computer running the *HTTP Server* software.
- **Socket**: a technique used by applications running on separate (or the same) computers to communicate information with one another.
- ***htsh* Web Application**: A PHP4-scripted web application which provides the browser interface to the *htsh* system.
- **(*htsh*) Login Host**: The computer on which users are allowed to login and operate shells via *htsh*.
- ***htsh* (Server) Daemon** (*htshd*): A process running on the *Login Host* which authenticates login requests sent via the *HTTP Server* and creates a *Client Shell* to serve each login.
- **(*htsh*) Client Shell**: A process running a standard shell on the *Login Host*, which accepts input sent via the *HTTP Server* from the *Client Browser* and sends output back via the same route.

3. Operating Model

Diagram 1 gives an overview of the operation of *htsh*.

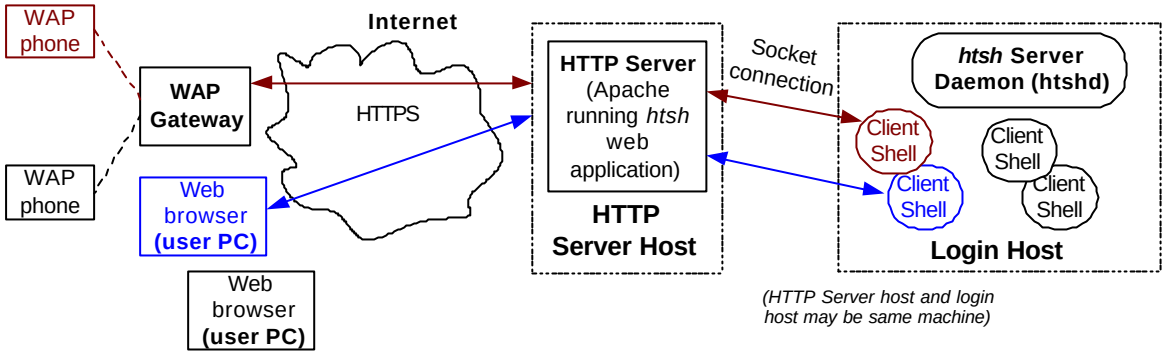


Diagram 1: Schematic overview of the operation of htsh

In order to make use of *htsh*, the following steps must take place:

1. Via a *Client Browser*, the user enters the URL of the *HTTP Server Host* providing access to the desired *Login Host*.

2. The user is presented with a login page, into which they enter their username and password (as defined in the system password database on the *Login Host*) and then submit this information to the *HTTP Server*. If the *HTTP Server* provides access to more than one *Login Host* then the login page will also allow the user to select their desired *Login Host*.

3. The *HTTP Server* transmits the username and password to the *Login Host* which then authenticates the user.

4. On successful authentication, the *htsh* server daemon (*htshd*) running on the *Login Host* creates a *Client Shell* for the user, and returns a success notification to the *HTTP Server* (after which the *HTTP Server* then allows further communication between the *Client Browser* and the *Client Shell*). Included in the success notification message is a randomly generated authorisation key. This key is a secret value shared between the *Client Shell* and the *HTTP Server*. All subsequent transmissions from the *HTTP Server* to the *Client Shell* include a copy of this key and this copy is checked by the *Client Shell* to verify that the transmission has been sent by the logged in user.

5. Using a browser form, the user submits shell input to the *HTTP Server*. The *HTTP Server* transmits this input to the *Client Shell*. The *Client Shell* passes the resulting output back to the *HTTP Server*, which in turn passes it back to the *Client Browser*, where it is displayed to the user.

6. The previous step can be performed repeatedly. Upon completion, the user sends a logout message to the *Client Shell* which then exits.

There are some important points to add to the above diagram and description:

- *Client Shell* output is only ever sent to the *Client Browser* in response to submission of input. (There is no "push" of data from the *HTTP Server* to the *Client Browser*). This means that any output which isn't generated by the *Client Shell* within a short period of time after submission of the current user input will not be returned to the browser in the current request-response cycle. Instead, it will be "piggy-backed" with the output produced in response to the next request from the user's browser. (It is also possible to send a special command ("Check output") which simply requests that any outstanding output from the *Client Shell* is returned to the *Client Browser*.) This mode of operation is required since it is not possible for the *Client Shell* to definitively know whether or not a command has finished producing output. Therefore the *Client Shell* operates a timeout on the output, and if the timeout is exceeded, ceases transmitting output to the HTTP server.

- Some commands to the *Client Shell* may generate large amounts of output. Rather than transmitting all of this output to the *HTTP Server* / *Client Browser*, the *Client Shell* will transmit only a portion to the *HTTP Server*. Further output will be sent to the *HTTP Server* as described in the previous point.

4. *Client Shell* Initialisation

Before executing the user's standard shell, the *Client Shell* performs a number of initialisations based on the contents of the following files:

1. The global initialisation file created by the system administrator and used by all *htsh* logins. (By default this file is expected to be `/etc/htsh_profile`, but a different file name may be specified by the system administrator when starting the *htsh* server daemon.)

2. The user-specific initialisation file `.htshrc` in the user's home directory.

These files are processed in the above order. Either or both of these files may be absent: in this case *htshd* proceeds without error.

Both files have the same format and contents. Each allows the specification of shortcuts to appear in a menu on the *Client Browser*, and also allows the setting of attributes controlling which *htsh* features will appear in the *Client Browser* display and how they will appear. Control statements are provided enabling initialisations to be performed conditionally depending on the communication protocol ("wap" or "http") being employed by the *Client Browser*, and user agent (i.e. *Client Browser* type – each phone and/or web browser model/version generates a unique user agent string).

Note that some of the settings in the initialisation files correspond to values that can also be set globally for users by the system administrator in the *htsh* server daemon (*htshd*) configuration file and on the *htshd* command line (*as described in the htsh Installation and Administration Guide*). Since the initialisation files are processed at the time of user login, settings in these files will override corresponding settings in the *htshd* configuration file and command line.

4.1. Initialisation file content

4.1.1. General formatting notes

The following general rules apply when writing initialisation files:

- Lines whose *first* non-whitespace character is a "#"are treated as comments and ignored.

- Blank lines are ignored

- Lines (other than comment lines) can be continued by placing a backslash at the end of the line and continuing on the next line (the backslash and newline are removed)

- White space indentation can be freely used to make the file layout more readable

4.1.2. Initialisation file commands

The following commands may appear in the initialisation file.

Command format	Description

`sc [-n] name definition`
`sc [-n] definition`
	Create an entry for the browser *shortcut* menu. *Name* specifies the string to appear in the menu list. *Definition* is the corresponding text which will be sent to the *Client Shell* if the user selects this menu item
	
	In the second form, a shortcut is created whose name and definition are the same string.
	
	If the *name* and *definition* strings contain embedded spaces they should be nested in single quotes. Embedded quotes (and backslashes) in either string can be escaped using "\" (backslash).
	
	The *–n* option specifies that a trailing newline should *not* be included as part of the shortcut when it is sent to the *Client Shell*.
	
	By default, if the second argument of this command starts with a hyphen (-), *htsh* assumes this argument specifies options to the command. To create an option whose name starts with a hypen, use the form:
	`sc -- -name definition`
	Some examples:
	`sc 'list files' 'ls –F'`
	`sc ps`
	`sc status 'echo $?'`
	`sc warning 'echo Don\'t do that'`
	`sc -- -demo 'echo hyphen'`

`clearsc`	Clear the list of shortcuts created so far. This command is primarily intended for use (somewhere near the top) in the user-specific initialisation file (`.htshrc`) to remove shortcuts created in the global initialisation file.
`set name value`	Set a value for one of the named variables controlling the operation of *htsh*. The variables which may be set are described below.
`set allowedprotocols 'proto-name... '`	Specify the set of protocols which may be used to login to *htsh*. If more than one protocol name is specified, the names must be specified in single quotes and separated by spaces or tabs. The following protocols can be specified (in lower case): • wap • http By default, the permitted set of login protocols is that specified by the administrator when starting the *htsh* server daemon. (If the administrator has not explicitly specified a list of protocols, then all available protocols can be used to login to *htsh*.) Note that this command can only be used to specify a subset of the protocols specified by the administrator at *htsh* server daemon startup. (In other words, if the administrator specified that only "http" logins were permitted, then this command cannot be used to enable "wap" logins.) Example: `set allowedprotocols 'http wap'`

`set csoutputtimeout` *`nsecs`*	Specify the time for which the *Client Shell* will wait for any further shell output (after receiving shell input) before informing the *HTTP Server* that output is complete. To avoid slow response times, this should be set to some small value (usually less than one second). Example: `set csoutputtimeout 0.5` Default is the value specified (for all users) during startup of the *htsh* server daemon. Attempts to set this value outside the range 0.1 to 15.0 result in an error.
`set csmaxtransfersize` *`nbytes`*	Specify the largest number of bytes that will be transferred in a single block by the *Client Shell* to the *HTTP Server*. This is useful to prevent large outputs from choking the *Client Browser*. Otherwise, the *HTTP Server*/*Client Browser* could be overrun with large amounts of output and the user would be prevented from sending further input (for example a *Control-C* to abort the command generating the output) until all of the output has been completed. This setting defaults to the maximum value specified by the system administrator when starting the *htsh* server daemon, and attempts to specify a value higher than the maximum are silently ignored. Attempts to set this value less than 1000 bytes are silently ignored.
`set historyblocksize` *`nitems`*	Set number of items to be displayed in each block of the history list. By default the web browser interface will display a maximum of 10 commands a time from the history and a WAP phone will display 3 commands. Attempts to set this value to a number lower than 3 are silently ignored.
`set outputbufferlimit` *`nbytes`*	Set an upper limit for the size of the buffer used to record all output during this shell session. This setting defaults to the maximum value specified by the system administrator when starting the *htsh* server daemon, and attempts to specify a value higher than the maximum are silently ignored. Setting this value to zero means that no shell output is buffered (and thus it will only be possible to use the buffer scrolling commands to browse the most recent block of output returned from the shell).
`set outputwindowsize` *`nbytes`*	Set the maximum number of characters to be displayed in each window of output. By default, a web browser displays a maximum of 1000 characters of output at a time, and a WAP phone displays 200 characters at a time. Attempts to set this value lower than 100 bytes are silently ignored.
`set shortcutblocksize` *`nitems`*	Set number of items to be displayed in each block of the shortcut list. This setting only has effect for WAP browsers. By default, the WAP browser interface will display a maximum of 10 shortcuts on one page. Attempts to set this value to a number lower than 3 are silently ignored.
`set shelltimeout` *`num-secs`*	Set the timeout period for shell input. If input is not received within this time, the *Client Shell* automatically terminates. Default is the value specified (for all users) during startup of the *htshd* server daemon.

`set wapbrowserstyle` ***`style`***	Set the browser style to be employed on a WAP phone. (This option only has meaning for WAP phones.) Normally it should never be necessary to set this option: default for this value is automatically determined according to the phone type, as described below. Permissible settings for style are as follows: • *up* – Can be used to force Nokia phones to use the same navigation mode as UP browser phones (i.e. page navigation by in page hyperlinks rather than the softkey menu) Example: `set wapbrowserstyle up`
`set –o` ***`option`*** `set +o` ***`option`***	Enable (-) or disable (+) an option
`set +o allowcontrolchars`	If this option is disabled then the user will not be presented with buttons and input boxes allowing control characters to be sent to the shell session. By default this option is on.
`set +o allowshellcmd`	If this option is disabled, the browser interface will not include a text box for directly entering shell commands. This may be useful if you only want to allow the user to enter input to the client shell using the shortcut menu. By default, this option is enabled.
`set -o allowsilent`	If this option is disabled, then the user is offered the "`> null`" checkbox when entering shell input or editing shell input history. By default this option is off.
`set –o allowtrigraphs`	(This one's for all the old time C programmers out there!) Some WAP phones have a limited set of input keyboard characters available. For example, the Siemens C32 has no means of inputting a backslash character. Trigraphs are three character sequences (all beginning with two question marks) which are interpreted as though they were equivalent single characters. If this option is enabled, then the following trigraphs (as in the C standard) are interpreted: • `??= for #` (hash) • `??( for [` (brackets) • `??) for ]` • `??/ for \` (backslash) • `??' for ^` (caret) • `??< for {` (braces) • `??> for }` • `??! for |` (pipe symbol) • `??- for ~` (tilde) By default this option is off.

`set +o allowuserinit`	If this option is disabled in the global initialisation file, then processing of user-specific initialisation files is prevented. This provides a way for the *htsh* administrator to provide a fixed interface (as defined by configuration file and global initialisation file options) for all users. By default this option is enabled (i.e. user-specific initialisation files are processed).
`set +o displaymenu`	If this option is disabled, the browser interface will not include a menu of shortcut commands. By default, such a menu will appear if at least one entry for the shortcut menu has been created using the *sc* command.
`set –o filteransiesc`	If this option is enabled, then the *Client Shell* attempts to filter any ANSI terminal escape sequences from the output generated by commands. (This may be necessary because certain programs, notably certain options to *ls(1)* on Linux, generate ANSI escape sequences regardless of the setting of the TERM environment variable. By default, this option is off.
`set +o history`	If disabled, the user will not be provided with access to history enabling past shell inputs to be resent. By default this option is on.

`ifprotocol` ***protocol-name***
cmd-list
`fi`
	Allows for sections of the *.htshrc* file to be conditionally processed, depending on what protocol (lowercase "http" or "wap") is being used for communication with the *Client Browser*.
	`ifprotocol wap`
	`sc pwd`
	`fi`

`ifuseragent` ***user-agent…***
cmd-list
`fi`
	Allows for sections of the *.htshrc* file to be conditionally processed, depending on which user agent (browser type) is being employed. Since user agent names often contain spaces, these names should be nested in single quotes. Multiple user agents may be specified on the command line: if any of the specifications match then, the contained *cmd-list* is processed.
	User agent specifications are case sensitive. The user agent specifications can make use of the wildcard characters "*", "?", and "[ ]" with the same meanings as in the shell.
	`ifprotocol wap`
	`ifuseragent '*Nokia*'`
	`set outputblocksize 400`
	`fi`
	`fi`

4.1.3. An example user-specific initialisation file

The following commands show a sample initialisation file which include conditional logic to handle initialisation for Web and WAP logins:

```
# htsh startup file (.htshrc)
#
set csoutputtimeout 1.5
set csmaxtransfersize 5000
```

```
# Make shell timeout 1 hour

set shelltimeout 3600

# Uncomment the following if the "> null" checkbox is desired
#set -o allowsilent

# Special stuff for WAP

ifprotocol wap
    set shortcutblocksize 4
    set outputwindowsize 200
    set historyblocksize 5
    set shelltimeout 600

    set -o allowtrigraphs

    # Create shortcuts to save typing on phone

    sc pwd
    sc who
    sc date
    sc processes 'ps ax -o "pid uid cmd"'

    # Do some extra stuff if this is a Nokia phone

    ifuseragent '*Nokia*'
        sc 'NokiaWap' 'echo This is a Nokia WAP Phone'
        set shortcutblocksize 15

        # Uncomment following if you prefer hyperlinks on Nokia phone
        #set wapbrowserstyle up
    fi
fi

# Some general shortcuts for WAP and Web browser

sc greet 'echo \'hello world\''
sc 'apache stop' '/sbin/init.d/apache stop'
sc 'apache start' '/sbin/init.d/apache start'
sc 'allow core dumps' 'ulimit -c unlimited'
sc 'sleep 60'

# Create a shortcut for a long command

sc 'Large output' 'j=0; while test $j -lt 100; do k=0; \
        while test $k -lt 5; do \
        echo -n "$j-$k aaaaaaaaa "; k=`expr $k + 1`; done; \
        echo ""; j=`expr $j + 1`; done'
```

4.2. Environment Variables

htsh sets a number of environment variables before launching the user's login shell. These include:

`SHELL` Pathname of the user's login shell

`HOME` Pathname of the user's login directory

`TERM` Set to *glasstty* so that programs which abide by the value of this environment variable setting will not attempt to perform screen-mode operations.

`HTSH_PROTOCOL` Set to indicate the communication protocol being employed by the *Client Browser*. Contains either “wap” or “http”

Testing for the existence of this string is the correct way of determining if the login shell is being run over *htsh*.

```
if test ! –z $HTSH_PROTOCOL then
    alias vi='echo "Using vi is not a good idea in htsh"'
fi
```

To test for a specific protocol value, the following can be used

```
if test "X$HTSH_PROTOCOL" = "Xwap" then
    echo "This is a WAP login"
fi
```

`HTSH_USER_AGENT` The user agent identification string as passed in HTTP headers from the *Client Browser* to the *HTTP Server*.

The following sample shows the kinds of tests that can be made in a shell startup file (such as `.bashrc`):

```
if test ! -z "$HTSH_PROTOCOL"; then
        unalias ls         # So ls on Linux doesn’t generate escape sequences
        echo "Looks like an htsh login"
fi
if test "X$HTSH_PROTOCOL" = Xhttp; then
        echo "Ya gotta shell, Web Browser! '$HTSH_USER_AGENT'"
fi
if test "X$HTSH_PROTOCOL" = Xwap; then
        echo "Ya gotta shell, WAP phone! '$HTSH_USER_AGENT'"
        PS1='$ '           # Make prompt short
        if expr "X$HTSH_USER_AGENT" : 'X.*Nokia' >/dev/null ; then
                echo "Looks like this is a Nokia WAP phone"
        fi
fi
```

5. User interface

The user interfaces for web browsers and WAP phone offer essentially the same functionality. The major difference is that all shell command input and output features are provided on a single page display in the web browser interface, while on the WAP phone, these operations are split over several screens and menus. Note that the setting of options in the global and user-specific *htsh* initialisation files will control whether all of the features described in the following sections appear.

6. WAP Browser User Interface

Known under the name *wapsh*, *htsh* currently provides support for two categories of WAP phone:

- Phones running the widely used **UP (Unwired Planet) browser** from *phone.com.* These include phones from Siemens and Motorola (other phones running the UP browser are listed at *http://www.phone.com/*)
- **Nokia** phones.

wapsh determines the type of phone in use and tailors its operation to the features and limitations of each phone type.

The following pages show example screenshots for the two browser types. These screen shots were obtained using:

- the Nokia 7110 Phone Emulator, which provides a close emulation of a real Nokia phone.
- the emulator provided with the Unwired Planet Software Development Kit. This emulator only provides an approximate emulation of a phone display, so that the appearance of *htsh* on UP-browser supplied phones may vary somewhat from the screenshots shown here.

6.1. WAP Browser Navigation

The following WAP-specific terms are used in describing the *wapsh* user interface.

- **Deck**. A deck is approximately the WAP analogue of an HTML web page. Each request to a WAP server returns a deck, which is then displayed by the phone browser. The crucial difference between an WAP deck and an HTML web page is that a deck can be split into a number a of *cards.*
- **Card**. A WAP deck consists of one or more cards. At any time, the phone browser displays one of the cards from a deck. The user can navigate from one card to another using hyperlinks and menu menu options. The advantage of the use of cards is that the user display can be split into logically separate parts, which the user can navigate between, without requiring a (time-consuming) request-response cycle between the phone and the WAP server.

In the description below, the terms *page* and *screen* will often be used synonymously with *deck*.

6.1.1. WAP Browser Navigation Map

Diagram 2 shows the navigation paths between the various pages (decks) in the *wapsh* user interface. Note that the navigation model and style used by *wapsh* will depend on the type of WAP phone in use, as described in the next sections.

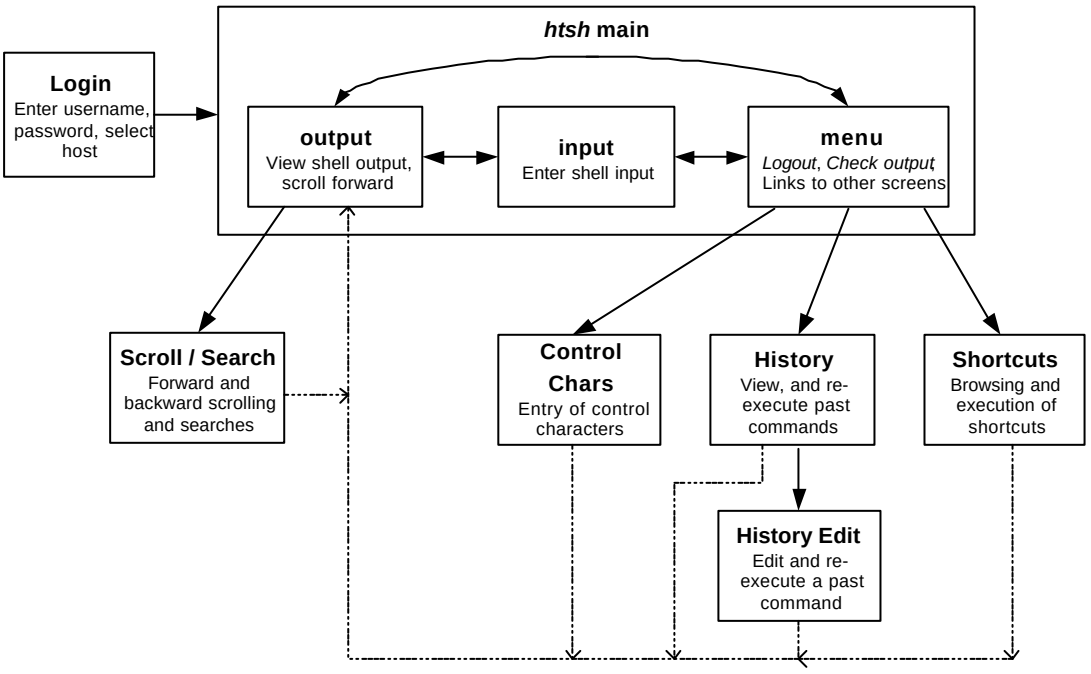


Diagram 2: WAP Browser Navigation Map

6.1.2. UP Phones

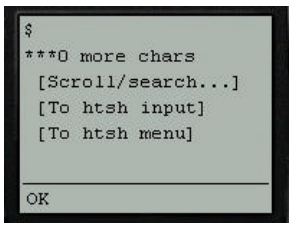
When using UP Phones, the main *htsh* page consists of three parts (cards) labelled *output*, *input*, and *menu*, whose separate functions are shown in Diagram 2. These three cards are transmitted as part of a single WML

deck (page), so that navigating between them does *not* involve a round trip to the WAP gateway and is thus immediate. At the foot of each of the three cards are hyperlinks allowing navigation to each of the other cards on the page.

Navigation to other (sub-)pages (*Scroll/search*, *Control characters*, *History*, *Shortcuts*) is provided via hyperlinks on these cards as indicated in Diagram 2. Each sub-page provides a set of hyperlinks which can be used to navigate back to any of the three main *htsh* cards.

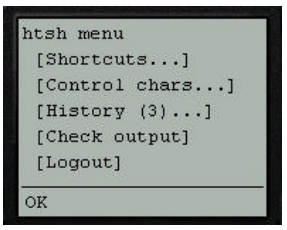
The following displays show the three cards of the main page:

Output card



Input card (partial display)

Menu card (partial display)



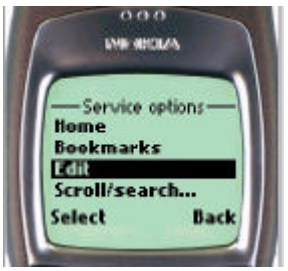
6.1.3. Nokia Phones

Nokia WAP phones run a browser which provides superior handling of forms and input fields, as well as true modeless access to the softkey menus, and *htsh* takes advantage of these features. In particular:

- The main *htsh* page combines all three parts (*output*, *input*, *menu*) onto a single page.
- All navigation between pages is done via options in the (left) softkey menu

The following screen shows the start of the *htsh* main page as displayed on a Nokia phone (the rest of the page text is available by scrolling down, and will be discussed in more detail shortly).

Navigation to other pages is provided via the softkey menu:



(Note that the first three items on the softkey menu, *Home*, *Bookmarks* and *Edit* are defaults with standard meanings provided by the Nokia browser.)

6.2. Login page

The URL of the *wapsh* login page has the form:

`http://`*domain*`/login_wml.php`

This screen allows the user to enter their username and password. If the *HTTP server* permits logging into a range of *Login Hosts*, these will be displayed in a menu on the login page and the user can select the desired *Login Host*. After entering the preceding information, the user clicks the **Login** hyperlink to enter *htsh*

Here is the *Login* page as displayed on a Nokia phone (the UP login procedure is similar, although the display appears different as per the usual mode of the UP browser):

Login page (first half)

Login page (second half)

6.2.1. Specifying a default username and login host (*wapsh* only)

If you regularly login to with the same username on the same login host, you may find it useful to create a bookmark which specifies these defaults. To do this, create a bookmark of the form:

`http://`*domain*`/login_wml.php?u=`*myusername*`&h=`*preferredhost*

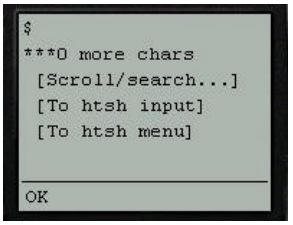
(Note that, for security reasons, it is *not* possible to specify a default for the password value.)

6.3. Shell output

All output so far generated by all commands during a shell session is stored in a buffer on the *HTTP Server*. This output buffer grows continuously up to a maximum limit defined by the system administrator when the *htsh* server daemon is started. (Individual users can also set lower values for the buffer size using the *set outputbufferlimit* initialisation file command). Once this limit is reached, old input is discarded from the start of the buffer.

At any moment, the *Client Browser* displays a section of the output buffer (normally the most recent output). When a command generates a large volume of output, the *Client Browser* displays the first section of output and provides scrolling options to advance forward (or backward) through the output.

After initially logging in, we see a display similar to the following:



The initial "$" character shown here is the prompt from the shell. After we have executed any command, the output of that command is displayed at the top of the *htsh main* screen.

After the shell output comes a line of the form "*** *X* more chars". When a command generates more output than can be displayed in a single WAP browser page, *htsh* will show the first part of the output and use this line to show how much further output is available for viewing. This further output can be viewed using the "Output scrolling" options described below.

6.4. Shell text input

On the Nokia phone, to enter input to the shell, simply scroll down the page until the shell input textbox is visible. For UP browsers, click the **To htsh input** hyperlink.

A textbox can be used to enter text to be sent (by pressing the *send* button) to the *Client Shell*. In addition to supplying the input text, the user can set two further options:

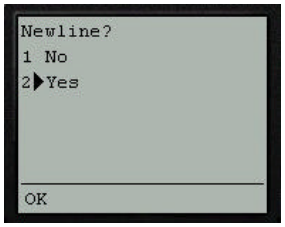
- **"Newline?"** – specifies that a trailing newline is to be appended to the submitted text (This option is enabled by default.)

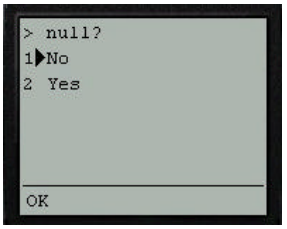
- **"> null"** – This option is useful to discard the output from a command which is expected to generate a large volume of output. It is exactly equivalent to appending the string "`> /dev/null`" to the end of the input text. [**Note**: This option will only be displayed if the *set -o allowsilent* option was specified in an *htsh* initialisation file.]

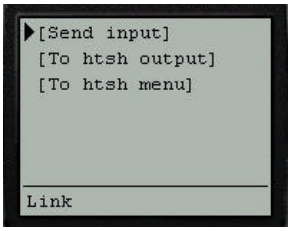
After filling in all of the above information, the shell input can then be sent using the **Send input** hyperlink.

On the WAP phone all of the above information is entered by scrolling through the *htsh main* page:

On the UP browser, the process is similar, although the page display appears somewhat differently:

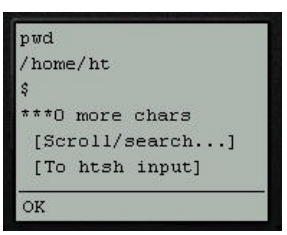






(If, after entering shell input on the UP browser, you decide that you do *not* want to send it to the shell, simply click either of the links to the *output* or *menu* cards, instead of the **Send input** hyperlink.)

After submitting the shell output, *htsh* will then display the resulting output. Here is the output we would see on the two phone types:



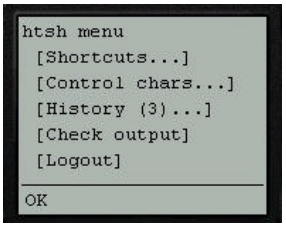
(Note that the shell echoes whatever input we enter, so that we also see the command ("pwd") we just entered.)

6.5. History

htsh maintains a history of past shell inputs (excluding shortcut menu selections and control characters which we describe below). This list can be reviewed and individual items edited and resent. Since the history list may be long, it is displayed in blocks of a fixed number of items, and buttons are provided to navigate forward and backward through the list. (Other than this, navigation through the history list is non-existent. In particular, options are not currently provided to jump to the top or bottom of the list or to search through it.).

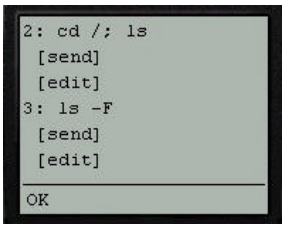
Any item from the history list can be immediately re-submitted by clicking the associated **send** hyperlink, or first edited and then re-submitted.

To access the history list, choose the **History** option from softkey menu (Nokia phone) or go to the *htsh menu* card (UP browser) and select the **History** hyperlink:[1]



On both phones, *htsh* displays a count of how many commands are currently in the history list.

The history list appears as follows:



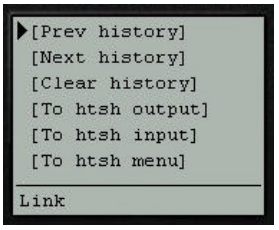
The **send** hyperlink re-submits the corresponding input to the shell without editing, while the **edit** hyperlink allows the input to be edited before resubmission.

Note that when re-submitting an input from the history using the **send** hyperlink, the same "newline?" and "> null" settings will be used as were specified when the input was originally entered. To view or change these settings, the command must be edited before re-submitting.

6.5.1. Scrolling through the history list

If the history list is long, *htsh* allows you to scroll forward or backward through it. Scrolling options, **prev history** and **next history**, as well as the option to return to the *htsh main* page(s), are provided either in the softkey menu (Nokia phone) or at the foot of the history page (UP browser).

[1] Note that history hyperlinks/softkey options are only displayed if at least one shell input has been sent to the client shell (and history access has not been disabled (`set +o history`) in the *htsh* initialisation files).



6.5.2. Clearing the history list

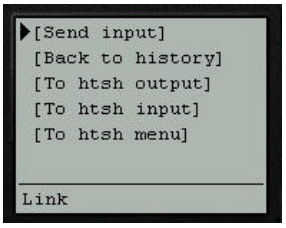
If desired, the entire contents of the history list can be erased. This feature is available under the softkey menu on the Nokia phone, or as a hyperlink at the base of the main *history* page on the UP browser. Clearing the history also automatically returns the user to the *htsh main* (Nokia) or *output* (UP browser) page.

6.5.3. Editing a command from the history

The process of editing a command is similar to entering shell input. When editing an item, in addition to changing the text of the item, it is also possible to modify the settings of the "`newline?`" and "`> null`" checkboxes as desired. The item can then be re-submitted by pressing the corresponding **send** hyperlink.

The following screenshots show this process for the Nokia phone.

Note that hyperlinks / softkey options are provided allowing the user to exit from editing a command and return either to the history list or the *htsh main* page(s).

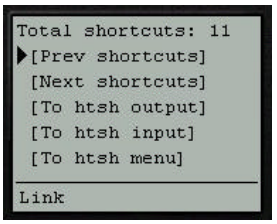


6.6. Shortcuts

The shortcuts defined in the *.htshrc* file with the "sc" command are made available to the user via a set of hyperlinks on the *Shortcuts* page. The shortcuts page appears as follows:

Selecting one of the shortcut items causes the corresponding shell input (which is not displayed as part of the menu list) to be sent to the *Client Shell*.

If the list of shortcuts is long, then it will be displayed in blocks of a fixed number of items (set according to the *set shortcutblocksize* initialisation file command described earlier) with accompanying scrolling options (**Next shortcuts**, **Prev shortcuts**) to move backward and forward through the list of options. These options, as well as options to return to the *htsh main* pages appear under the softkey menu (Nokia phone) or at the foot of the shortcut list (UP Browser).



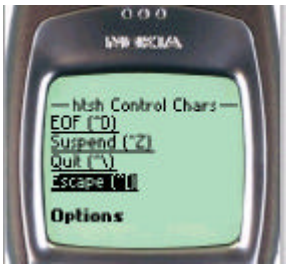
6.7. Control character input

Since the keyboard of a WAP phone is limited, *htsh* provides a special page allowing control characters to be sent to the shell. A textbox may be used to send any standard control character to the *Client Shell*. To send for example, a Control-A, enter the letter "A" (upper or lower case) into the textbox, and click the *send* button. (To be precise, this feature causes the ASCII code whose numeric value is 0x40 (64 decimal) less than the entered character to be sent to the *Client Shell*.)

Alternatively a set of hyperlinks can be used to send specific control characters. The following hyperlinks are provided for specific control characters.

- Control-C (Interrupt)
- Control-D (End of file)
- Control-Z (Suspend)
- Control-\ (Quit)
- Control-[(Escape)

The following screens show the appearance of the control characters page on the Nokia phone (the functionality is similar on the UP browser, although the operation of the browser makes the appearance slightly different):



6.8. Output scrolling

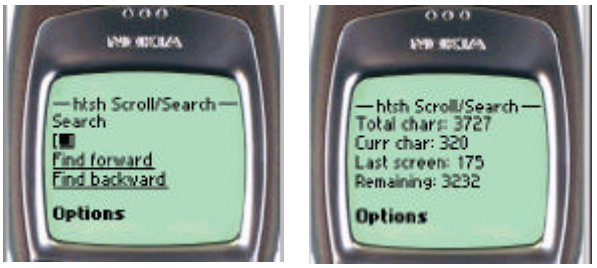
As noted already, if a command generates a large amount of output then *htsh* will only display the first part of the output. Facilities are provided to allow us to scroll though the remaining output (or back to previous output). As an example, suppose we entered the command "*ls –l*" in the root directory (/). Then on the *htsh main/output* page we would see the following display on the Nokia phone (UP browser is similar):

Scrolling down the page, we eventually see the following:

The **Next screen** hyperlink allows us to skip to the next undisplayed block of output. The **Bottom Screen** hyperlink allows us to skip all intervening output and advance to the last page of output.

In addition to these features, the *Scroll/Search* page allows more general searching scrolling through the output generated so far in the login session. As usual this page is reached via the *htsh main* page softkey menu (Nokia phone) or the *htsh menu* page (UP browser).

On the Nokia phone the *Scroll/Search* page appears as follows:

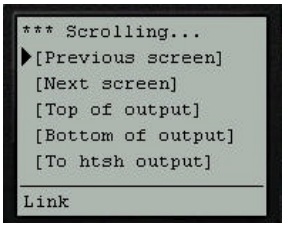


The top half of this page allows for searching the output buffer. The bottom half of the page displays information about the buffer as follows:

- The total number of characters in the buffer.
- The current position in the output (i.e. the location of the first character on the last output screen on the *htsh main/output* page.
- The number of characters displayed in the last output screen on the *htsh main/output* page.
- Number of characters remaining to be displayed beyond the end of the last output screen on the *htsh main/output* page.

6.8.1. Scrolling forward and backward

Options to scroll forward and backward through the output buffer are provided on the *Scroll/Search* page either as softkey options (Nokia phone) or hyperlinks at the foot of the page (UP browser).



Depending on the amount of output so far generated during the login session some of the following navigation options will be provided:

- **Next screen** – Show the next screen of output (this option will not be displayed if there is no output beyond that currently displayed)

- **Bottom of output** – Advance to the last screen of output in the buffer. (This option will not be displayed if there is no more than one further screen of output past the screen currently being viewed – in which case *Next screen* suffices to advance to the remaining output.)

- **Previous screen** – Move to the previous screen of output. (This option will not be displayed if the user is currently viewing the topmost section of the output buffer.)

- **Top of output** – Move to the topmost screen of the output buffer. (This option will not be displayed if there is no more than one previous screen of output above the screen currently being viewed - in which case *Previous screen* suffices to navigate to it.)

Note that when scrolling through the output buffer, *htsh* will attempt to display whole lines of output where possible. This means that *htsh* will sometimes display fewer characters than the permitted maximum (as specified by the *set outputwindowsize* initialisation file command).

6.8.2. Output searching

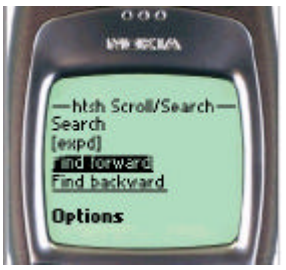
The textbox and **forward / backward search** hyperlinks can be used to enter a string to search for in the output buffer. Searches may be made forward or backward and proceed from the current screen. If a search is successful, *htsh* will attempt to display text from the beginning of the line containing the string. Subsequent searches (if performed without intervening scrolling or other operations) will, however, proceed from the point of the last match, rather than the start of the line.

As an example, suppose that we have just executed the "processes" shortcut described earlier, and that we wish to search the output for a process named "expd". This is the procedure we would follow (on the Nokia browser – the UP browser is similar):

Execute a command with a lot of output

Go to the Scroll/Search page

Enter string "expd", and search forward



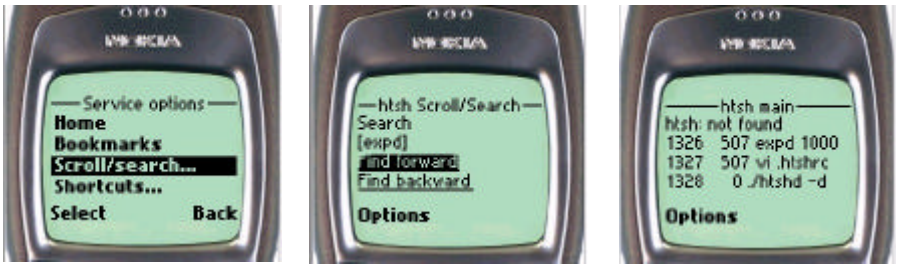
String found at character 3168 in output buffer

After a successful search, *htsh* displays a message string indicating where in the buffer the search string was found, and then the line containing the string (where possible this line will be shown from the beginning). If a search is unsuccessful, *htsh* displays a warning message and leaves our position in the output buffer unchanged, as in the following example, where we unsuccessfully attempt to search for a second occurrence of the string "expd":

Go to the Scroll/Search page

htsh "remembers" last searched for string

Search fails, so we stay at same place in buffer



6.9. Check output

In considering the discussion that follows, you may find it useful to refer back to Diagram 1which shows the relationship between the *Client Browser*, the *HTTP Server* and the *Client Shell*.

Recall that each time a *Client Browser* (WAP phone or web browser) submits input to the shell, this input is passed by the *HTTP Server* to the *Client Shell*. The *Client Shell* then yields output which is passed back to the *HTTP Server* which formats the output appropriately and passes it back to the *Client Browser*. All output produced in a login session and returned to the *HTTP Server* is maintained in a (per-session) buffer which the user may scroll though and search using the procedures described above.

Sometimes a shell command may generate a large volume of output (for example: *ls –lR /usr*) or may take a long time to produce some output (for example: *sleep 10; date*). In both of these cases it is undesirable to have *htsh* wait until all command output has been produced. In the first case this is because the output would take a long time to produce and transmit, and during this time the user would be unable to enter further shell input (such as a Control-C to abort the command). In the second case, there is in general no way for *htsh* to definitively know when the *Client Shell* has finished output and *htsh* should avoid waiting too long for output and instead allow the user to enter further shell input if desired. For these reasons, *htsh* operates two governors on the shell output. The first of these is the limit established by the *set csmaxtransfersize* initialisation file command, which specifies the maximum amount of output that will be transferred back to the *HTTP Server* in response to a shell input (by default 10000 bytes). The second limit is established by the *set csoutputtimeout initialisation* file command which specifies the maximum time that *htsh* will wait without detecting any output before ceasing to wait and instead allowing the user to enter further input. (This timeout value should normally be set to some low value - say 0.5 seconds).

If either of these two governors comes into effect, then it may be that the *Client Shell* still has further output to send to the *Client Browser*. By default, when the *Client Browser* next submits some input, the outstanding output will then be delivered back to the *Client Browser*. Sometimes, however, we would like to check if there is any further output without sending any shell input at all. This is exactly the feature provided by the **Check Output** option, as shown in Diagram 3.

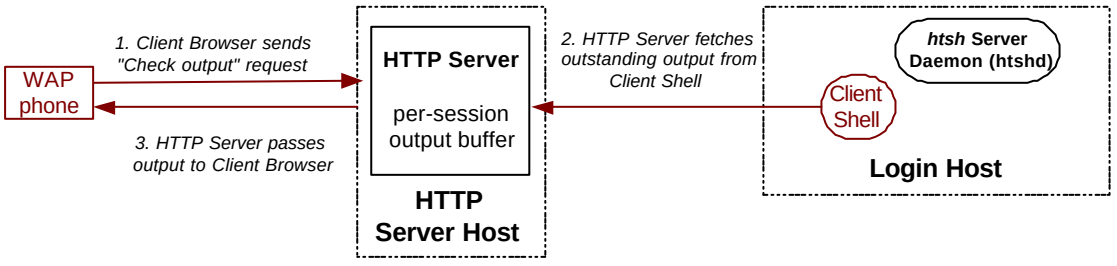
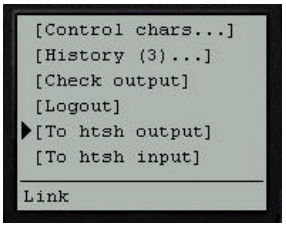


*Diagram 3: Logical operation of **Check Ouput***

Note that it is important to distinguish the purpose of the **Check Output** option from that of the output scrolling and search facilities described earlier. The latter are provided to navigate through the buffer of all *Client Shell*

output, which is *already* on the *HTTP Server*. The **Check Output** button forces a trip to the *Client Shell* itself, to check for further output, which will then be appended to the output buffer maintained on the *HTTP Server*.

The **Check Output** option is provided in the *htsh main* page softkey menu (Nokia phone) or in the *htsh* menu page (UP browser).



6.10. Logout

This option (available on the *htsh main* page softkey menu (Nokia phone) or the *htsh menu* page (UP Browser)) terminates the *htsh* login session and closes the *Client Shell* on the *Login Host*.

7. Web Browser Interface

htsh also provides a web browser interface. One use for this interface is as a training ground to give an idea of the capabilities of the *htsh* interface prior to trying the WAP interface to the system (*wapsh*).

7.1. Web Browser Navigation Map

Diagram 4 shows the navigation paths for the web browser interface of *htsh*.

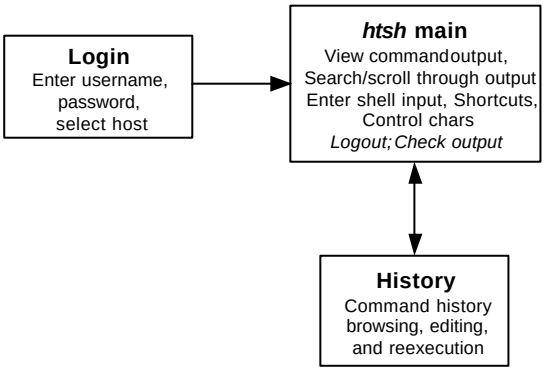
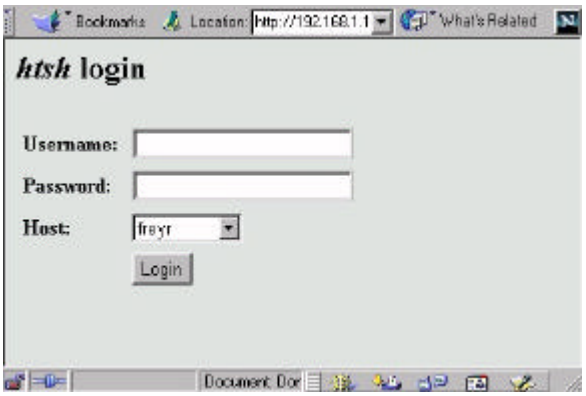


Diagram 4: Web Browser Navigation Map

7.2. Login page

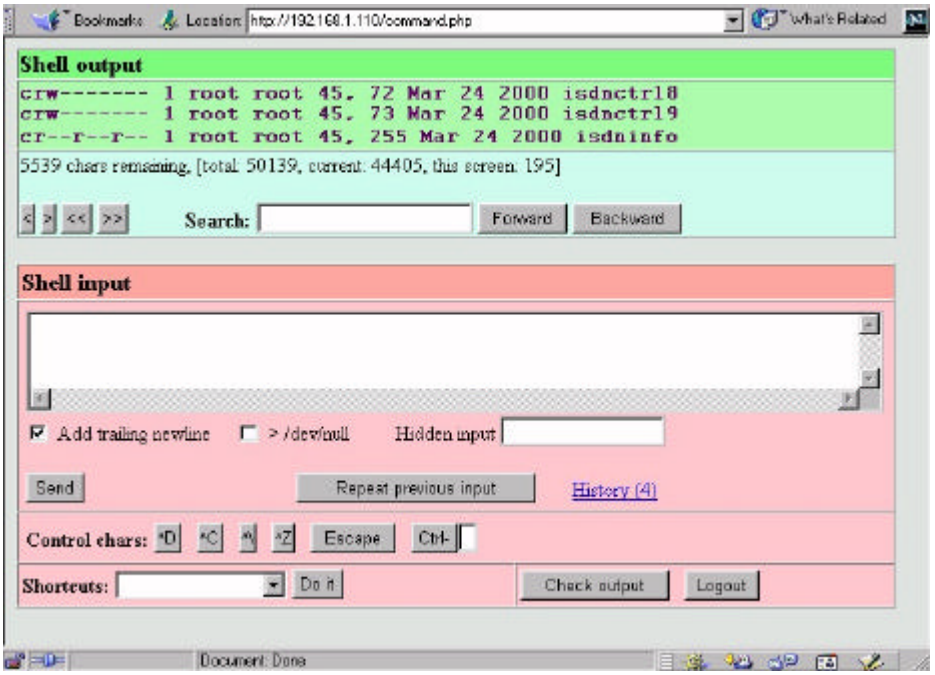
The URL of the *htsh* login page has the form:

```
http://domain/login_html.php
```



7.3. Main page

The web browser main page combines all of the features previously described in the WAP interface (except history browsing and editing).



The following further differences are found in the web browser interface:

- Buttons, rather than hyperlinks, are used to perform most actions.
- The scrolling actions, “Previous screen”, “Next screen”, “Top of output”, and “Bottom of output” are replaced by buttons labelled (respectively) “<”, “>”, “<<”, and “>>”.
- The web browser interface provides an additional **Hidden Input** textbox. This can be used to enter text (such as passwords) which is not supposed to be displayed on the browser. Only one of the “Standard input textbox” or *Hidden Input* textboxes should be filled in when submitting shell input.
- The web browser interface provides an additional **Repeat Previous** button with no counterpart (currently) in the WAP Phone interface. Clicking this button re-submits the most recently submitted shell input.

7.4. History page