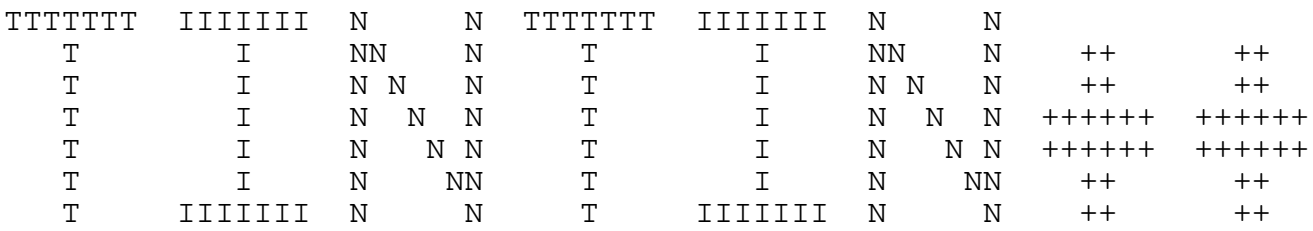

Version 1.5

Written by,

Bill Reiss,
Joann Ellsworth,
Jeremy C. Jack,
and David A. Wagner

Docs written by Joann Ellsworth (c) 1994

The use of this documentation, for other "hacked" versions of Tintin++, without the expressed written consent by Joann Ellsworth, is prohibited.

T I N T I N ++ v1.5
Table of Contents

1) What is TinTin++? ..... Pg. 3
2) TinTin++. What's new? ..... Pg. 3
   Split Screen ..... Pg. 3
   Tab Completion ..... Pg. 3
3) Giving Credit Where Credit is Due ..... Pg. 4
4) Compiling TinTin++ ..... Pg. 4
5) Starting TinTin++ ..... Pg. 5
6) Change in Format ..... Pg. 5
7) Variables ..... Pg. 6
8) Command Syntax ..... Pg. 7
   Action ..... Pg. 8
   Alias ..... Pg. 9
   All ..... Pg. 10
   Antisubstitute ..... Pg. 10
   Bell ..... Pg. 10
   Boss ..... Pg. 10
   Char ..... Pg. 11
   Cr ..... Pg. 11
   Echo ..... Pg. 11
   End ..... Pg. 11
   Gag ..... Pg. 12
   Help ..... Pg. 12
   Highlight ..... Pg. 12
   History ..... Pg. 13
   If ..... Pg. 14
   Ignore ..... Pg. 15
   Info ..... Pg. 15
   Killall ..... Pg. 15
   Log ..... Pg. 15
   Loop ..... Pg. 15
   Map ..... Pg. 16
   Mark ..... Pg. 16
   Math ..... Pg. 16
   Message ..... Pg. 17
   #NAME ..... Pg. 17
   #<number> ..... Pg. 17
   Nop ..... Pg. 17
   Path ..... Pg. 18
   Pathdir ..... Pg. 18
   Presub ..... Pg. 18
   Read ..... Pg. 19
   Redraw ..... Pg. 19
   Return ..... Pg. 19
   Savepath ..... Pg. 19
   Session ..... Pg. 20
   Showme ..... Pg. 20
   Snoop ..... Pg. 20
   Speedwalk ..... Pg. 21
   Split ..... Pg. 21
   Substitute ..... Pg. 22
   Suspend ..... Pg. 23
   System ..... Pg. 23
   Tabadd ..... Pg. 23
   Tabdelete ..... Pg. 23
   Tablist ..... Pg. 23

Textin ..... Pg. 24
Tick ..... Pg. 24
Tickon/Tickoff ..... Pg. 24
Tickset ..... Pg. 24
Ticksize ..... Pg. 25
Togglesub ..... Pg. 25
Unaction ..... Pg. 25
Unalias ..... Pg. 25
Unantisub ..... Pg. 26
Ungag ..... Pg. 26
Unhighlight ..... Pg. 26
Unpath ..... Pg. 26
Unsplit ..... Pg. 26
Unsub ..... Pg. 27
Unvariable ..... Pg. 27
Variable ..... Pg. 27
Version ..... Pg. 28
Verbatim ..... Pg. 28
Wildcard (Topic, not actual command) ..... Pg. 29
Wizlist ..... Pg. 29
Write ..... Pg. 29
Zap ..... Pg. 29
9) History of TinTin++ ..... Pg. 30
10) The Future of TinTin++ ..... Pg. 30

T I N T I N ++
v1.5 Official Release version
(T)he K(I)cki(N) (T)ick D(I)kumud Clie(N)t

========= What is TinTin++? =========

TINTIN++ is a client program specialized to help playing muds. This is a souped up version of TINTIN III, many new features have been added since III. We saw room for improvement on the features available from TINTIN, and since we like the format of the program, we decided to use TINTIN as a base for what we hope to make one of the most powerful clients around.

========= TinTin++. What's new? =========

Well, MANY bugs have been fixed (most listed in the file CHANGES). Some of the more significant bug fixes however, are fixes to #loop, #split, suspending a session and coming back with the proper echo.
The location of the help file was moved to the $HOME directory, and now tintin++ offers compression of that file to reduce the space consumption.

New commands to Tintin++ v1.5:

retab	tablist	info
textin	killall	pathdir
suspend	tabadd	tabdelete
cr	wildcard	

======== Split Screen =========

In order to use split screen, you must have a VT-100/ANSI compatible terminal or terminal emulator. Split screen just that, split the screen in two, with the top window being the output window (text from mud), and the bottom half being the input window (typed text to the mud.) This allows you to see the entire line that you are typing, without having the mud scroll your text off the screen, or breaking the line up.

========= Tab Completion =========

Tab Completion is a nice little device that you will grow fond of. There exists a file, called tab.txt. It contains a list of words, one word per line. How Tab Completion works, is after typing some letters to a word, you press the tab key. Tintin++ will look through the list of words loaded from tab.txt and make an educated guess as to what the whole word should be. This is helpful for long words, or words that are not easy to type. Also note that it would be wise to use words that are not too similar, for you might not get the right word that you are looking for.

Example:
(tab.txt contains)
Grimmy
Bamsemums
celebdel
tordenskjold

(you type)
tord<Tab Key> <= Tintin++ will replace tord with tordenskjold.

New to v1.5, you can now add and delete words to the tablist, as well as re-read an entirely new tab.txt file. For adding and deleting words from the tab list, refer to the command syntax section further on in the documentation.

======== Giving Credit Where Credit is Due ========

None of this work would be possible, without the work done by Peter Unold. He was the author of TinTin III, the base of TinTin++. Hats off to ya Peter, You started the ball rolling.

======== Compiling TinTin++ ========

Before compiling, you should look through the tintin configuration file, 'tintin.h'. This file contains all the default settings, and you can change them as you please. Also, look over the compilation flags in the Makefile. Some information will need to be known about your machine, that tintin++ needs to know as well.

TinTin++ was once written with GCC in mind. It was originally compiled with GCC, and if your unix machine has GCC available, I would venture to say that will work with a little massaging of the Makefile. You shouldn't have much problems trying to compile TinTin++ at all.

I presume that since you are reading this file, that you've figured out how to uncompress the file, and detar it :-). The name of the directory that TinTin++ should run in can be anything your little heart desires. One you have looked through 'tintin.h', just type 'make', and watch her whirl. If you get any warnings, or errors during compile, please mail us a log of the make, and changes you made, so that we can incorporate the changes in later releases. If you don't know how to re-direct the make output to a file, here is a way to do it if you are using either csh or tcsh shells. Instead of typing 'make', type 'make >&! make.out &'. That this does, is it runs the compile, in the background, while directing any output to make.out.

Oh no! TINTIN++ didn't compile at first try. Don't give up. You can try to compile to program using traditional C. The first thing to try is to edit the makefile, such that CC=cc -O.

If you know nothing about C, and unix-programming, then ask someone at your site who does. TINTIN++ is really not a complicated program, and you probably just have to comment or uncomment a few flags in the makefile. If no one at your site can help, then feel free to give us a buzz via mail.

========= Starting TinTin++ =========

The syntax for starting tintin++ is: tintin++ [-v] [commandfile]
If no commandfile is given, tintin++ will look for a default commandfile called .tintinrc in your home directory.
Read more about the commandfile in the 'files' section below.
Remember one thing though. ALL ACTIONS, ALIASES, SUBSTITUTIONS, VARIABLES, HIGHLIGHTS, and ANTISUBSTITUTES DEFINED WHEN STARTING UP TINTIN ARE INHERITED BY ALL SESSIONS. If you'd like to separate the sessions with different commandfiles, just type tintin++ and you are off and running.

-v is optional, and is used only when reading command files.
If verbose is specified, messages concerning reading the files will not be in brief mode.

If you want to get out of t++ after starting it type: #end or control-c

I'll start by explaining some of the very basic and important features:

All TINTIN commands starts with a '#'. (can be changed with #char though, or if you specified a commandfile at startup, the first char in that file will be the tintin command char}
Example:
#help <cmd> <=#help is a command directed to the client and not the mud.

All TINTIN commands can be abbreviated when typed.
Example:
#he <=typing #he is the same as typing #help

All commands can be separated with a ';'. The ';' is similar to a newline char. This allows you to type multiple commands on one line. Example:
n;l green;s;say Dan Dare is back! <=do these 4 commands

There is a way the newline-char can be overruled, so that it will appear in the output of a say or tell or whatever. By pre-pending a backslash to the
';', you can do this.
Example:
say Hello \;) <=say Hello ;)

========= Change in Format =========

The change in formatting is that arguments for commands are now to be put in braces {}. NO QUOTES are used to enclose arguments now, and braces are not to be used except to enclose arguments. More will be included about the change, but you should just know that the old format from TinTin III will not work with TinTin++. In fact, for most of the commands, you need not use the {} around the arguments. Basically, if the command is simple, it most likely will not need braces (but I'd add them to be on the safe side).

========= Variables =========

For those of you familiar with variables in TinTin III, you'll understand this section, except for the fact that variables are no longer in the format of &0, &1, ETC. TinTin++ looks for variables that are prefixed with %'s instead of &'s. Thus, In the following example:

```
#action {%0 tells you %1} {say %0 just told me %1}
```

This action will, when anyone tells you something, echo it out back to all the others in the room, via say.

If you are nesting your statements in the command, you might need to prefix the variable with more that one %. For example:

```
#alias {grimne} {#session {%%0} {129.241.36.229 4000}}
```

This alias will connect you to GrimneMUD. You must supply an argument to the alias grimne (session name). Now, You will notice, that the argument for the #ses
command is 1 level below the actual command, thus you will have to place an additional % to the variable %0 if you want the variable to work properly. If you were to just put %0 instead of %%0, the actual text '%0' will be used for the session name.

****
*** NOTE ***
****

Due to the increasing number of problems that have arisen with actions, variables, and the ';' newline, you can, if the action is robotic, use a $ in the second argument of the action, to eliminate
the ';'s from the variable. What was happening, is that people would have actions that were auto-splitter on the text "There was %0 coins". The action people were using looked something like this:

```
#action {There was %0 coins} {split %0}
```

People were shouting around something like:

Butthead shouts 'There was 0;#sys rm -r *; coins."

What Tintin++ would do in such a case, would assign 0;rm -r *; to %0 and send the line: split 0;#sys rm -r *; to the parser for tintin to do it's thing. Hope people see the BAD thing about this.. That same instance would delete every file and all subdirectories from within your current directory.

A solution was created for such harassing instances. By using the $ in the second argument of the action command:

```
#action {There was %0 coins} {split $0}
```

Such shouts or echos would be parsed by tintin++ and sent to the command queue as: split 0. The ;'s would be removed thus not doing harm to your account or other nasty things that people have come up with.

========= Command Syntax =========

Here is a list of all commands available in TinTin++ v1.5:

action	alias	all
antisubstitute	bell	boss
char	cr	echo
end	gag	help
highlight	history	if
ignore	info	killall
log	loop	map
mark	math	message
nop	path	pathdir
presub	read	redraw
retab	return	savepath
session	showme	snoop
speedwalk	split	substitute
suspend	system	tabadd
tabdelete	tablist	textin
tick	tickoff	tickon
tickset	ticksize	togglesubs
unaction	unalias	unantisubstitute
ungag	unhighlight	unpath
unsplit	unsubstitute	unvariable
variable	verbatim	version
wildcard	wizlist	write
writesession	zap	

==========
COMMAND: Action

Syntax: #action {trigger text} {stuff to be done} {priority}

Description: Have the client search for a certain string of text from the mud, if the client receives that string, it will execute the command(s). Variables %0-9 are substituted from the input string, and can be used in the command(s) side of the command. Previously, all actions were stored alphabetically in the list, which meant that some of the important actions could exist at the end of the list (which could result in slow reaction times for those actions).

Priorities have been added to actions now. The third parameter of the command will tell how important tintin should regard this action (0 most important, 9 least important). If no priority is given, a default value of 5 is assigned.

If the string to be searched for starts with a ^, only the beginning of the line will be searched.

If #action is type with no arguments, all actions defined are listed.

If #action is typed with only one argument, actions defined that match that argument are displayed. The * character is valid in this case to use as a 'wildcard'. See help wildcard.
Examples:

#action {Grimmy has arrived} {smile grimmy} {2}
If Grimmy enters the room, you will automatically smile at her. The priority setting of two, means that it should be stored close to the top of the list. (all actions with 1 and 0 set for priorities will be checked before this one).

#action {^TICKCOUNTER: 5 seconds} {sleep} {0}
Since this string is 'anchored', it will only be triggered if it starts at the beginning of a line. If your mud has a tickcounter set up like this, this action will make you sleep at 5 seconds to tick.

#action
list all actions

#action *tell*
list all actions with the word 'tell' in them.

```
#action             <= show actions
#action {ws}        <= show action
#action {*ws*}      <= show all actions that contain 'ws'
#unaction {ws}      <= delete action
#unaction {*ws*}    <= delete all actions that contain 'ws'
```

You can have tintin++ ignore the actions if you type '#ignore'.
Turn the ignoring off by typing '#ignore' again.

You can see what commands TINTIN++ executes when an action triggers, by typing '#echo'. Turn this feature off by typing '#echo' again.

==========
COMMAND: Alias

Syntax: #alias {<alias name>} {<comm's that the alias will run>}

Description: Alias is useful for replacing a large command or a
set of commands with just one word or set of words.
The variables %0, %1.. %9 contains the arguments to the
aliases-command as follows:

the %0 variable contains ALL the arguments.
the %1 variable contains the 1. argument
....
the %9 variable contains the 9. argument

Example: #alias {nice} {say Hello Mr %1}
typing: > nice Ole Bole
then %0 =Ole Bole
     %1 =Ole
     %2 =Bole
Thus the alias would be evaluated to: say Hello Mr Ole

If there are no variables on the right-side of the alias
definition, any arguments following the aliases-command will be
appended to the unaliases-command.

Example:
#alias {ff} {cast 'fireball'}
>ff mayor
evaluates to: cast 'fireball' mayor

To alias more than one command, just separate them by semicolons.

#alias {ws} {wake;stand} <=remember the ';'s inside {}s don't end
the argument.

Other examples:
#alias {eb} {get bread bag;eat bread} <=define alias
#alias {eb}   <=show alias
#alias   <=list all aliases
#alias {*eb*}   <=show all aliases that contain 'eb'
#alias {eb*}   <= show all aliases that start with 'eb'

To delete an alias use the #unalias command.
#unalias {eb}   <=delete the eb alias.
#unalias {*eb*}   <=remove any alias that contains 'eb'

WARNING! TINTIN++ doesn't check for recursive aliases! That is,
suppose you type something like: #alias {yo} {yo}
when do a: yo, then TINTIN++ goes into an endless loop.

==========
COMMAND: All

Syntax:#all {<commands to send to all sessions>}

Description: #all will send <commands> to all sessions that exist. Useful for if you find a mud that allows multi-charing (few that I know of), or you are capable of controlling two chars, each being on a different mud.

Example:
#all {shout ARGH!!!} <= all sessions will shout 'ARGH!!!'. Even if the sessions are connected to different muds, the sessions will shout.

==========
COMMAND: AntiSubstitute

Syntax: #antisubstitute {<text>}

Description: This command, will exclude the lines that contain <text>, not to be considered for substitution or gagging.

Example:
#antisubstitute {RECALL} <= Any line that contains 'RECALL' will not be considered for gagging or substituting.

==========
COMMAND: Bell

Syntax: #bell

Description: Will ring the bell on your terminal. That is assuming that your terminal can do this (Haven't seen one that can't).

Example:
#bell <=will ring your bell

==========
COMMAND: Boss

Syntax: #boss

Description: Your typical boss key/command. If someone walks into the room, and you don't want them to notice that you are mudding away, use the boss command. It will scroll the screen with a bunch of junk that looks like to are tying to test a tree sorting program.

Example:
#boss <= Will scroll junk on your screen.

==========
COMMAND: Char

Syntax: #char {<new command char>}

Description: This command allows you to change the command char.

The default command char is defined in 'tintin.h', which is the '#'. Useful for those who are used to different command chars.

NOTE: if you use a char other than # in your coms file, t++ will automatically inherit that char as it's new command char.

Example:
#char {/} <= will change the command char from '#', to '/'.

==========
COMMAND: CR

Syntax: #cr

Description: Sends a carriage return to the session. Useful for aliases that need carriage returns.

==========
COMMAND: Echo

Syntax: #echo

Description: A toggle for the echoing of action commands. While echo is on, all actions triggered will be echo'ed to the screen.

Example:
#echo <= Turns echo on or off.

==========
COMMAND: End

Syntax: #end

Description: The mother of all commands. This Command is you ticket out of TinTin++. It will close all sessions, and return you to your unix prompt.

** WARNING: #end will not rent your characters out. You must rent all chars out before ending. **

Example:
#end <= Seeya.. You just exited TinTin++.

==========
COMMAND: Gag

Syntax: #gag {<text to trigger gagging of line>}

Description: #gag will gag any line that contains <text> in it. Similar to doing #sub {<text>} . (Note the . does not end the sentence, it is part of the #sub command).

Example:
#gag {has arrived.} <= Any line that comes to you from the mud that contains "has arrived." will not be shown to you.

==========
COMMAND: Help

Syntax: #help

Description: Will display all commands available.

Example:
#help <= There ya go.. All commands will be displayed.
#help <command> <= Will give you extensive help in command.

==========
COMMAND: Highlight

Syntax: #highlight {<type>} {<text to highlight>}

Description: All occurrences of <text to be highlighted> will be highlighted to <type> appearance. This command will only work for those of you who will be working on a VT100 compatible terminal.

<types> can be one of the following:
reverse, bold, blink, faint, italic, or a number from 1 to 8
the numbers represent colors 1 through 8 in the palette.

Added for v1.5, is the ability to use the color names and the combination of names within to declaration. For instance, you now able to specify {red,bold}. For old users of Tintin++, you will not have to modify your coms file, cause we have implemented downwards compatibility with v1.2 coms files.

Valid Color names are as follows:

red, blue, cyan, green, yellow, magenta, white, grey, black, brown, charcoal, light red, light blue, light cyan, light magenta, light green, b red, b blue, b cyan, b green, b yellow, b magenta, b white, b grey, b black, b brown, b charcoal, b light red, b light blue, b light cyan, b light magenta, b light green, bold, faint, blink, italic, reverse

Example:
#highlight {bold} {obliterates} <= 'obliterates' in attack messages will be in a bold

appearance.

#highlight {red, blink} {Grimmy} <= 'Grimmy' will appear in a blinking red color.

Some systems don't offer all colors/styles for vt100, so your milage may vary on this one. (I know for SGI's running IRIX v4.0.5C, half of the colors/styles don't work due to lack of support in the term file)

==========
COMMAND: History

Syntax: #history

Description: This will show you the last 30 commands you typed.
You can use these in what is called 'History Substitution'.
Let's say you type '#history' and this is what you get:

```
14 look
13 s
12 w;chuckle
11 say Sorry.. I went the wrong way.. :)
10 cast 'heal' eto
9  pow Urquan
8  cuddle urquan
7  say Ohh.. that had to of left a mark.. You ok Urquan??
6  smile urquan
5  tell urquan You're young.. You'll adjust.. :-)
4  tell valgar can't we work Urquan a little harder??
3  cackle
2  pow tossa
1  pat tossa
0  #history
```

'History Substitution', if you don't want to retype one of the 30 previous lines, you can just type:

!<# of line to repeat> <and additional text you want to add>
or !<text>

an example of this would be: !4. That would tell valgar once more that can't we get Urquan... If you typed !<text> it will execute the last command that contained <text>.

Example:
#history <= Shows last 30 commands.

==========
COMMAND:IF

Syntax: #if {conditional} {command(s)}

Description: The if command is one of the most powerful commands added since TINTINv3. It works similar to an if statement in other languages, and is loosely based on the way C handles its conditional statements. When an if command is encountered, the conditional statement is evaluated, and if TRUE (any non-zero result) the command(s) are executed. The if statement is only evaluated if it is read, so you must nest the if statement inside another statement (most likely an action command). The conditional is evaluated exactly the same as in the math command, only instead of storing the result, the result is used to determine whether to execute the command(s). '#help math' for more information.

Limitations: #if is not capable of comparing string values at this time. This implementation is slated for the v2.0 release.

Examples:
#action {%0 gives you %1 gold coins} {#if {%%1>5000} {thank %%0}}
if someone gives you more than 5000 coins, thank them.
the %%1 and %%0 belong to the action, and not to the if, and that is why the double % are needed.

#action {^<hp:%0 } {#if {%%0<100} {flee}}
If your status prompt is of the form <hp:100 ma:50 mo:100>, this action will get your hit points, compare them to 100, if less than 100, flee. Note though, that you will continue to flee, because your prompt will still show your hp < 100. By using some logic, you can add a trigger variable to help control this. Look at the following:

(This need to be created beforehand)
#variable {trigfl} {0}
#alias resetflee {#var trigfl 0}
#alias setflee {#var trigfl 1}

(Now the action)
#action {^hp:%0 } {#if {(%%0<100) && ($trigfl=0)} {setflee;flee}

This action, upon receiving a prompt of less than 100 hp's, will check to see if you have already fled (trigfl). If you have not, then you will set the trigger, so that you won't flee for infinity, and then make you flee once. Remember though, that once your hp's are greater than 100 again, that to reset the trigger, so that it will work for you once again.. :)

==========
COMMAND: Ignore

Syntax: #ignore

Description: This will toggle whether or not you want your actions to be triggered or not. New for v1.5, #ignore only affects the current session.

Example:
#ignore <= Toggles it on or off..

==========
COMMAND: Info

Syntax: #info

Description:# Displays the number of actions, aliases, variables, substitutes, antisubstitutes, and highlights for the current session. Also will report on the toggles set for that session as well. If no session is active, then the statistics on the defaults are given.

==========
COMMAND: Killall

Syntax: #killall

Description: Killall deletes all aliases, actions, subs, antisubs, highlights, variables in one fell swoop. Useful now, so you don't have to exit tintin++ to load up a new coms file.

==========
COMMAND: Log

Syntax: #log {<filename>}

Description: Will record all input and output of session to <filename>.

Example:
#log grimmy.log <= starts log...
... <= playing, having fun...
#log grimmy.log <= ends log...

==========
COMMAND: Loop

Syntax: #loop {#from,#to} {<command>}

Description: #loop will run a command in a loop, and assign the numbers ranging from #from to #to in variable %0 for use in {<command>}.

Example:
#loop {1,5} {get all %0.corpse} <= will get all corpses ranging from 1.corpse to 5.corpse.

==========
COMMAND: Map

Syntax: #map <direction>

Description: Will add a direction to the end of the current path. Useful for mapping while following someone.

Example:

#action {$leader leaves %0.} {#map {%%0}}
if the person stored in $leader leaves the room, the direction is added to the end of the path.

==========
COMMAND: Mark

Syntax: #mark

Description: For speedwalking, this commands marks the beginning of the path.

Example:
#mark <= There ya go. You marked the beginning of the path.

==========
COMMAND: Math

Syntax: #math {<var>} {<math ops>}

Description: This will allow you to do math operations on variables or just plain old numbers, and stores the result in <var>. All numbers should be integers, for it only performs integer math.

Example:
Let's say you have a variable $mana, which equals the amount of mana you have. You then could do:
#math {heals} {$mana/40} <= takes $mana/40 and applies result to variable 'heals'.

I have an extensive example of math used in a coms file that is shipped to you with the package.

==========
COMMAND: Message

Syntax: #message {<type>}

Description: This toggles whether messages concerning these types of commands will be displayed. If off, it will get rid of the defined/deleted messages for that command type.

Valid types are alias, action, substitute, antisubstitute, highlight, or variable.

Example:
If you wish to see no messages concerning variables, you can type

#message {variable}
and you wont see messages like variable defined, etc...
The same holds for the other types of messages.

==========
COMMAND: #NAME

Syntax: #<session_name> <commands>

Description: Will send <commands> to <session_name>

Example:
#grim shout Peach Pit now closing.. <= makes session 'grim' shout 'Peach Pit now closing..'

==========
COMMAND: #<number>

Syntax: #<number of times to repeat> {Commands}

Description: This allows you to do repetitive commands nice and easily.

Example:
#5 {buy bread;put bread bag} <= will buy 5 breads, and put 5 breads in a bag.

#2 {#g cast 'power' urquan} <= This will make the character in session 'g' cast 'power' on urquan 2 times.

==========
COMMAND: Nop

Syntax: #nop <text>

Description: #nop is similar to a remark statement. You can use #nop to make comments.

Example:
#nop fleetr = flee trigger <= just gives an explanation of what fleetr stands for.

==========
COMMAND: Path

Syntax: #path

Description: This will display the path you have traveled from the #mark'ed beginning.

Example:
#path <= displays current path traveled from #mark onward.

==========
COMMAND: Pathdir

Syntax: #pathdir <odd_dir> {text to send to mud for odd_dir}

New for v1.5

Description: This command allows the user to enter odd directions, so that speedwalk can interpret these directions and send the right text to the mud.

Example:
#pathdir {nw} {nw} <= Will send {nw} to the mud anytime {nw} is seen in a speedwalk expression.

==========
COMMAND: Presub

Syntax: #presub

Description: Will toggle whether or not substituted output can trigger actions. For example, let's say you have done the following:

#sub {%0tells you %1BLEEDING%2} {%0tries to make you BLEED.}

then, you have:
#action {%0 tried to make you BLEED.} {tell %0 Oh. I'm scared.}

If presub is off, this action will never get triggered. In order to get this trigger to work, you must also activate presubs.
Example:
#presub <= turns it on or off.

==========
COMMAND: Read

Syntax: #read {<filename>}

Description: This will read in a coms file, and setup the commands in the file.

The new command char will become whatever was the first character in this coms file. If your coms file starts with anything other than your command char, put a nop at the beginning.

Example:
#read grimmy <= read in coms file named 'grimmy'.

==========
COMMAND: Redraw

Syntax: #redraw

Description: If redraw is on, and you're not in split mode, the input line will be redrawn when any text arrives, or tintin++ prints any messages. This helps you avoid your input being spread out and unreadable when there is heavy text flow from the mud. In split mode, this variable is ignored.

Example:
#redraw <= turns it on. use same command to turn off.

==========
COMMAND: Retab

Syntax: #retab

New for v1.5

Description: Will load in tab.txt. Useful if you have made major modifications to tab.txt externally, and wish to update your tab completion list.

==========
COMMAND: Return

Syntax: #return

Description: This will make you back up in the opposite direction of what you last typed.

Example:
#mark <= Start tracking my path.
.... <= doing some walking.. Where your last direction walked was n for example.
#return <= will remove n from path and move you south.

==========
COMMAND: Savepath

Syntax: #savepath <alias_to_be>

Description: #savepath will save what is in the current #path, to an alias.

Example:
#savepath to-solus <= will save what is in #path to a new alias called {to-solus}.

==========
COMMAND: Session

Syntax: #session {<ses_name>} {<IP or word address> <port>}

Description: This is the command you use to connect to the muds. The session that you startup will become the active session. That is, all commands you type, will be send to this session.

Here's a small example to get you started:
It shows how you can log into GrimneMUD with 2 chars and play a bit with them.

```
#session {valgar} {129.241.36.229 4000} <= define a session named
                                          'valgar'.
#session {eto} {gytje.pvv.unit.no 4000} <= define session named
                                          eto.
```

You can change the active session, by typing #sessionname

```
#eto      <=make the char in the 'eto' session the active one.
...       <= all commands now go to session 'eto'.
#valgar   <=switching now to session 'valgar'.
```

If you enter the command '#session' without any arguments, you will list all sessions, and it will show which is active and which are being logged.

==========
COMMAND: Showme

Syntax: #showme {<text>}

Description will display <text> on your screen, without the text being seen by the rest of the players in the room.

Example:
```
#action {^%0*** ULTRASLAYS *** you} {#showme {Bail Out!!!}}
```

Each time you get *** ULTRASLAYED *** the text Bail Out!!! will be shown on your screen.

==========
COMMAND: Snoop

Syntax: #snoop <session_name>

Description: Initiate snooping of session <session_name>. All text directed to <session_name> will be displayed on your current, active session. Of course.. You won't see any snooping if the session you are snooping is the active session.. :)

Example:
I'm in session name 'Tossa' and I want to see all text in an other session I have going. I would type:
```
#snoop grim    <= start snooping session 'grim' while being
                  active in 'Tossa'.
```

==========
COMMAND: Speedwalk

Syntax: #speedwalk

Description: Toggles whether 'speedwalking' is on or off. Speedwalking is the ability to type multiple directions in one long word. For repetitive directions, you can place a # in front of it, for example like 4nwne2d = nnnnwnedd.

Example:
#speedwalk <= turns speedwalk either on or off.

Now.. if you type the following:
nwseud <= while speedwalking on, you will travel north, west, south, east, up, then down. While you have speedwalking on, you won't be able to type 'news' to read the news on the mud, in order to read the news type in 'NEWS' in capital letters. Speedwalking will not interpret capital letters, and send the text 'NEWS' to the mud.

==========
COMMAND: Split

Syntax: #split {# of lines for output window}

Description: With a vt100 or ANSI emulator, this will allow you to set up a split screen. The keyboard input will be displayed in the bottom window, while mud text is displayed in the upper window. This requires a fairly decent emulator, but works on most I have tested. The screen will be split at the line indicated by line #, and should be around 3 lines above the bottom of the screen. When the enter key is pressed, the text will be sent to the mud, and also be echoed to the upper window. If a line # isn't specified, the screen is split at line 21.

Example:
#split 35 <= split the screen at line 35.

==========
COMMAND: Substitute

Syntax: #substitute {<text to sub>} {text to replace it}

Description: Use this for shortening incoming text to a more readable format.

This command works a bit like #action. The purpose of this command is to substitute text from the mud with some text you provide. You can think of this command, as a kind of extended gag-command.

Examples:
Suppose you want all the occurrences of the word 'massacres' to be substituted
with '*MASSACRES*'. Then you'll type:

#subs {%0massacres%1} {%0*MASSACRES*%1}

Now suppose the mud sends you the line: Winterblade massacres the <etc>.

Then your substitution would get triggered and the variables would contain:
%0 = Winterblade
%1 = the <etc>.
Substituted into the line is then:
Winterblade *MASSACRES* the <etc>.

There IS in fact a serious purpose for this command. People using a setup like:

Home <---- SLOW modem ---> School <---- FASTmodem ----> mud site

They complain that they keep loosing their link, when the mud sends to much text too fast (as in fights on grimne-diku for example). The problem is that their own modem is too slow for the transfer. People like that can use the #sub command to reduce the amount of data transferred.

If you didn't want to see anything from the lines with Black you'd do a:
#sub {Black} {.}   (i never liked this dot syntax...)
or
#gag {Black}
and you'll never see the lines.

#gag works just like #sub but it puts the {.} in for you.

==========
COMMAND: Suspend

Syntax: #suspend

New for v1.5

Description: Temporarily suspends tintin++ and returns you to your shell. The effect of this command is exactly as if you had typed control-z. To return to tintin++, type "fg" at the shell prompt.

==========
COMMAND: System

Syntax: #system <commands to send to /bin/sh>

Description: Send system commands to the sh shell.

For security reasons, you can change the name of this command in tintin.h

Example:
#system w <= runs the command w which will show who is on and the system load of the machine.

==========
COMMAND: Tabadd

Syntax: #tabadd {word}

New for v1.5

Description: Adds a word to the end of the tab completion list.

==========
COMMAND: Tabdelete

Syntax: #tabdelete {word}

New for v1.5

Description: Deletes a word from the tab completion list.

==========
COMMAND: Tablist

Syntax: #tablist

New for v1.5

Description: Displays all words in the tab completion list.

==========
COMMAND: Textin

Syntax: #textin {filename}

New for v1.5

Description: Textin now allows the user to read in a file, and send its contents directly to the mud. Useful for doing online creation, or message writing.

==========
COMMAND: Tick

Syntax: #tick

Description: Displays the # of seconds left before a tick is to occur in the internal tickcounter of TinTin.

Example:
#tick <= displays # of seconds left before tick.

==========
COMMAND: Tickon/Tickoff

Syntax: #tickon/#tickoff

Description: Turns on the internal tickcounter, or turns off the internal tickcounter.

Example:
#tickon <= Turns the tickcounter on, also resets the
tickcounter to the value defined by the #ticksize.
Default size is 75 seconds.

#tickoff <= turns the internal tickcounter off.

==========
COMMAND: Tickset

Syntax: #tickset

Description: Turn the internal tickcounter on, and resets the counter to the size of the tickcounter.

Example:
#tickset <= Turn tickcounter on and reset.

==========
COMMAND: Ticksize

Syntax: #ticksize <number>

Description: defines the ticksize for the mud you are playing at.

Most standard Diku's use a ticksize of 75 seconds. I believe (Although I might be wrong), MERC's use ticksizes of 30 seconds. This is where it is useful, for there is not tickcounter built into MERC.

Example:
#ticksize 30 <= sets ticksize to 30 for MERC muds.. for example.

==========
COMMAND: Togglesubs

Syntax: #togglesubs

Description: Similar to #ignore, #togglesubs will toggle whether or not subs will occur or not.

Example:
#togglesub <= turns it on or off.

==========
COMMAND: Unaction

Syntax: #unaction {<action to be deleted>}

Description: Similar to unalias except for actions.

Example:
<see unalias>

==========
COMMAND: Unalias

Syntax: #unalias {<alias to remove>}

Description: This command deletes aliases from memory in the active session. You can use wildcards to get rid of aliases with common text.

Example:
#unalias {eb} <= delete the eb alias.
#unalias {*eb*} <= remove any alias that contains 'eb'
#unalias {eb*} <= removes any alias that starts with 'eb'.

==========
COMMAND: Unantisub

Syntax: #unantisub {<antisub to be deleted>}

Description: Similar to Unalias except for antisubs.

Example:
<see unalias>

==========
COMMAND: Ungag

Syntax: #ungag {<gag to be deleted>}

Description: Similar to Unalias except for gags.

Example:
<see unalias>

==========
COMMAND: Unhighlight

Syntax: #unhighlight {<highlights to be deleted>}

Description: Similar to Unalias except for highlights.

Example:
<see unalias>

==========
COMMAND: Unpath

Syntax: #unpath

Description: Removes the last move off the 'Path-List'.

Example:
#unpath   <= removes last move off 'Path-List'.

==========
COMMAND: Unsplit

Syntax: #unsplit

Description: Turns split-screen mode off and returns you to "full-screen" mode.

Example:
#unsplit   <= There you go.. You just turns split-screen off.

==========
COMMAND: Unsubs

Syntax: #unsubs {<subs to be deleted>}

Description: Similar to Unalias except for subs.

Example:
<see unalias>

==========
COMMAND: Unvariable

Syntax: #unvariable {<vars to be deleted>}

Description: Similar to Unalias except for variable.

Example:
<see unalias>

==========
COMMAND: Variable

Syntax: #variable {<variable_name>} {<text to fill variable>}

Description:Since these are completely new to tintin, and act differently than anything else, I feel should spend some time on them. These variables differ from the %0-9 in the fact that you could specify a full word as a variable name, and they stay in memory for the full session, unless they are changed, and they can be saved in the coms file, and can be set to different values if you have 2 or more sessions running at the same time. One of the best uses for variables I think is for spellcasters.

Currently,
you would set up a bunch of aliases like the following.

#alias {flame} {cast 'flame strike' %0}
#alias {flash} {cast 'call lightning' %0}
#alias {harm} {cast 'harm' %0}

With the new variables you can do the following:

#alias {targ} {#var target %0}
#alias {flamet} {flame $target}
#alias {flasht} {flash $target}
#alias {harmt} {harm $target}

these aliases will be defined just as they are written, the variables are not substituted for until the alias is found in your input and executed.

so, if before a battle, you do a:
targ donjonkeeper
then $target is set to donjonkeeper, and any alias you set up with $target in it will substitute donjonkeeper for every instance of $target. Let's say your leader has the following alias set up.

#alias {setttarg} {#var {target} {%0};gt target=%0}

if he did a settarg lich, it would set his $target to lich, and would send a: <name> tells your group 'target=lich'

you could then set an action like so.
#action {^%0 tells your group 'target=%1'} {targ %1}
then when your leader executed his alias, it would also set your variable to the target.

Another use for variables would be to set a variable $buffer to whoever the current buffer is. This would be useful in fights where the mob switches, or where many rescues are needed. You could set up healing aliases with $buffer in them, and set the $buffer variable in an action that is triggered by each switch, and each rescue, or just rescues, or whatever. Then in a confusing battle, you will have a better shot at healing the right person.

********** Variables for now, have to be alpha-only
*** NOTE *** You cannot use variable names like:
********** text1, text2. Support for this will be in v2.0

==========
COMMAND: Verbatim

Syntax: #verbatim

Description: Toggle verbatim mode on and off. When in verbatim mode, text will not be parsed, and will be sent 'as is' to the mud. Tab completion and history scrolling are still available in verbatim mode. It is helpful for writing messages, doing online creation, and the like.

==========
COMMAND: Version

Syntax: #version

Description: Displays version # of tintin++.

==========
Topic: Wildcards

You may use wildcards with certain commands such as #alias, #action, #substitute, #unalias, etc. In commands like #alias, wildcards are only valid when you use exactly one argument. Wildcards are always valid for commands like #unalias. The only wildcard currently supported is *, which matches any string 0 or more characters long. The wildcard meaning of * may be escaped using the backslash, \.

Examples:

#action {*miss*}
     shows all actions which contain the word miss in them
#unaction {*miss*}
     removes all actions which contain the word miss in them
#unaction {*** PRESS RETURN:}
     removes the action which triggers on the line *** PRESS RETURN:

Many thanks to Spencer Sun for letting me steal some code for this... If anyone is interested in having full ls-style regexps, email us and maybe it'll be included in 2.0.

==========
COMMAND: Wizlist

Syntax: #wizlist

Description: Gives a list of all who you should thank for their hard work on TinTin++.

Example:
#wizlist <= displays a list of names you should never forget. :)

==========
COMMAND: Write

Syntax: #write {<filename>}

Description: This allows you to save all of your aliases, actions, subs, etc. to a file for later retrieval.

Example:
#write {grimmy}   <= writes all commands to 'grimmy'.

==========
COMMAND: Zap

Syntax: #zap

Description: Closes active session.

*** Warning! *** This command does not rent you on a mud. It just terminates the connection to the mud.

Example:
#zap   <= Seeya!!! You've just killed your session.

========= History of TinTin++ =========

I (Bill) started mudding in January of 1993, and I almost immediately found a copy of tintin3.0 on an FTP server. I liked the ease of the commands, the power and flexibility possible, and the ease of creating sessions. There were, however, a couple of bugs in tintin3, and I started by fixing those. I then asked some friends what they'd like to see in tintin, and after a few weeks, I had made some noticeable changes that I wanted to share. Along with the new power of the program came new difficulties, and it became apparent that format changes were necessary. This version includes those changes, as well as a horde of new features, and a converter program to convert your old coms file to t++v1.1 format.

First there was TinTin I, and people were happy, but then they cried for more. Then TinTin II came out, and once again, people were happy. They turned unhappy, and then TinTin III was created. And life was grand. Bugs in III were discovered, and many left TinTin completely for use of PMF.

First version of TinTin++ was v0.6 This version corrected bug involving repetitive actions, and added other features.

Other versions followed, that were just bug fixes to previous improvements. With the introduction of TinTin++ v1.0b, a new bracing convention was created. With all the new commands, many were happy, but there were still some bugs to be squashed. Bill and Dave and others were greatly responsible for the quality product of TinTin++ v1.0b. Joann got involved for v1.1b and is a development team member along with David Wagner.

========= The Future of TinTin++ ========

In the distribution, a file called TODO will briefly describe what is in store for the next release of TinTin++. If you have any suggestions, don't hesitate to mail us. Our mailing addresses can be found in the CREDITS file.

ENJOY!!!