Command	Syntax or Option	Explanation	Example
lt		List dumped tensors.	lt
	-n <name_pattern>	List dumped tensors with names matching given regular-expression pattern.	lt -n Softmax.*
	-t <op_pattern>	List dumped tensors with op types matching given regular-expression pattern.	lt -t MatMul
	s <sort_key>	Sort the output by given sort_key, whose possible values are timestamp (default), dump_size, op_type and tensor_name.	lt -s dump_size
	-r	Sort in reverse order.	lt -r -s dump_size
pt		Print value of a dumped tensor.	
	pt <tensor>	Print tensor value.	pt hidden/Relu:0
	pt <tensor>[slicing]	Print a subarray of tensor, using numpy-style array slicing.	pt hidden/Relu:0[0:50,:]
	-a	Print the entirety of a large tensor, without using ellipses. (May take a long time for large tensors.)	pt -a hidden/Relu:0[0:50,:]
	-r <range>	Highlight elements falling into specified numerical range. Multiple ranges can be used in conjunction.	pt hidden/Relu:0 -a -r [[-inf,-1],[1,inf]]
	-s	Include a summary of the numeric values of the tensor (applicable only to non-empty tensors with Boolean and numeric types such as int* and float*.)	pt -s hidden/Relu:0[0:50,:]
@[coordinates]		Navigate to specified element in pt output.	@[10,0] or @10,0
/regex		less-style search for given regular expression.	/inf
/		Scroll to the next line with matches to the searched regex (if any).	/
pf		Print a value in the feed_dict to Session.run.	

	pf <feed_tensor_name>	Print the value of the feed. Also note that the pfcommand has the -a, -r and -s flags (not listed below), which have the same syntax and semantics as the identically-named flags of pt.	pf input_xs:0
eval		Evaluate arbitrary Python and numpy expression.	
	eval <expression>	Evaluate a Python / numpy expression, with numpy available as np and debug tensor names enclosed in backticks.	eval "np.matmul((`output/Identity:0` / `Softmax:0`).T,`Softmax:0`)"
	-a	Print a large-sized evaluation result in its entirety, i.e., without using ellipses.	eval -a 'np.sum(`Softmax:0`,axis=1)'
ni		Display node information.	
	-a	Include node attributes in the output.	ni -a hidden/Relu
	-d	List the debug dumps available from the node.	ni -d hidden/Relu
	-t	Display the Python stack trace of the node's creation.	ni -t hidden/Relu
li		List inputs to node	
	-r	List the inputs to node, recursively (the input tree.)	li -r hidden/Relu:0
	-d <max_depth>	Limit recursion depth under the -r mode.	li -r -d 3 hidden/Relu:0
	-c	Include control inputs.	li -c -r hidden/Relu:0
lo		List output recipients of node	
	-r	List the output recipients of node, recursively (the output tree.)	lo -r hidden/Relu:0
	-d <max_depth>	Limit recursion depth under the -r mode.	lo -r -d 3 hidden/Relu:0
	-c	Include recipients via control edges.	lo -c -r hidden/Relu:0
ls		List Python source files involved in node creation.	
	-p <path_pattern>	Limit output to source files matching given regular-expression path pattern.	ls -p .*debug_mnist.*
	-n	Limit output to node names matching given regular-expression pattern.	ls -n Softmax.*

ps		Print Python source file.	
	ps <file_path>	Print given Python source file source.py, with the lines annotated with the nodes created at each of them (if any).	ps /path/to/source.py
	-t	Perform annotation with respect to Tensors, instead of the default, nodes.	ps -t /path/to/source.py
	-b <line_number>	Annotate source.py beginning at given line.	ps -b 30 /path/to/source.py
	-m <max_elements>	Limit the number of elements in the annotation for each line.	ps -m 100 /path/to/source.py
run		Proceed to the next Session.run()	run
	-n	Execute through the next Session.run without debugging, and drop to CLI right before the run after that.	run -n
	-t <T>	Execute Session.run T - 1 times without debugging, followed by a run with debugging. Then drop to CLI right after the debugged run.	run -t 10
	-f <filter_name>	Continue executing Session.run until any intermediate tensor triggers the specified Tensor filter (causes the filter to return True).	run -f has_inf_or_nan
	--node_name_filter <pattern>	Execute the next Session.run, watching only nodes with names matching the given regular-expression pattern.	run --node_name_filter Softmax.*
	--op_type_filter <pattern>	Execute the next Session.run, watching only nodes with op types matching the given regular-expression pattern.	run --op_type_filter Variable.*
	--tensor_dtype_filter <pattern>	Execute the next Session.run, dumping only Tensors with data types (dtypes) matching the given regular-expression pattern.	run --tensor_dtype_filter int.*
	-p	Execute the next Session.run call in profiling mode.	run -p
ri		Display information about the run the current run, including fetches and feeds.	ri
help		Print general help information	help
	help <command>	Print help for given command.	help lt