

History flags summarized.

Flags for the history command

```
hist -n -- print line numbers
hist -g -- print history for your past sessions (not just the current one)
hist -f filename.py -- writes history lines to filename.py
hist -l 10 -- will limit output to last 10 lines
hist -g timeit -- will filter lines that contain the string "timeit"
hist -u -g timeit -- same as filter, but will only print unique lines
hist -o -n -- also print outputs
```

Note: Typing `history??` in iPython will give you similar info.

Previous iPython sessions

Syntax: `hist -n ~1/`

```
hist -n ~1/20 - means one session back, and 20th input line
hist -n ~2/20 - means two sessions back, and 20th input line
Using hist -n 20 (i.e. without the ~ sign) defaults to the current ipython session.
```

Re-execute a history line

Syntax: `rerun <hi story refere nce>`

```
rerun 20 will execute line 20 of the current session
rerun ~1/20 will execute line 20 of the previous session
rerun 88-90 will execute lines 88,89, 90 of history
```

Recalling input history (for inline editing)

Syntax: `recall <hi story refere nce>`

1. `recall 42` -- will recall line 42, and give you the prompt for editing
2. `recall myfunc` -- will recall the most recent line containing `myfunc`, with the same effect.

Hitting enter after the edit will execute.

Saving history to a file

Syntax: `save filename.py <hi story refere nce>`

1. `save mymodule.py 22-40 25` -- Saves lines 22-40, 25 to file `mymodule.py`
2. `save -a` -- will append

Grepping or filtering history lines

Syntax: `hist -g <reg exp>`

```
hist -g func1 -- will list all history lines containing "func1"
hist -gn func1 -- same thing. adds line numbers
```

Editing history in an editor

Syntax: `edit <hi story refere nce>`

`edit 24 28 47` - loads the lines in order, in your configured ipython editor

Output history editing via the `"_oh"` variable

Summary : iPython allows you to edit **both** your past inputs, as well as past command outputs - so you if you want to avoid retyping this is handy. The built-in list variable `"_oh"` contains all your output history.

Syntax : `"edit _oh[<line-num of output history>]"`

use `"edit _oh[165]"` to open output line 165 in your editor.

Example --

```
`
In [ 165 ] : def myfunc():
print "hello"
....
In [200] : edit 165 # opens line 165 in editor we made changes to
that func w/o saving it to a file
out[200]: def myfunc()\n print "hello" \n
....
...
In [210]: edit _oh[200] # loads the func in editor
`
```

Use `"edit -x"` in case you are editing non-code stuff (to prevent) execution when you leave the editor.

Edit a function that you defined inline

Syntax : `edit <funcname>`

Example - An inline function is defined interactively, and then edited.

```
`
In [1] : def myfunc()
print "hello"
In [2] edit myfunc
`
```

Create a macro from history

Use the `%macro` command to create a macro from multiple history lines.

Example (lines 10 and 11) from history are as follows :

```
10: x=1
11: somefunc(x)
```

You can create a macro as follows :

```
In [20]: %macro my_macro 10-11
```

Now typing `my_macro` will execute those lines.