

# Command Line

## Tips and Tricks

## Using multiple commands at a time.

The real power in the command line comes into play when you learn how to walk instead of crawling. Crawling is when you enter single commands in order. Often, this method is all you need, but if you know the next three things you're going to type, why press [Enter] between each entry?

You can also use conditionals so that you can make the computer do the thinking, rather than running multiple commands and making the decisions on your own. Computers are supposed to make our lives easier, not more difficult.

## Using multiple commands at a time.

The simplest method of using multiple commands at the prompt is to use a semicolon between commands instead of pressing [Enter] between commands. For example, perhaps you want to create an empty file and then change its permissions so that it's executable by the user. You could

type:

```
touch file
```

```
chmod u+x file
```

or

```
touch file ; chmod u+x file
```

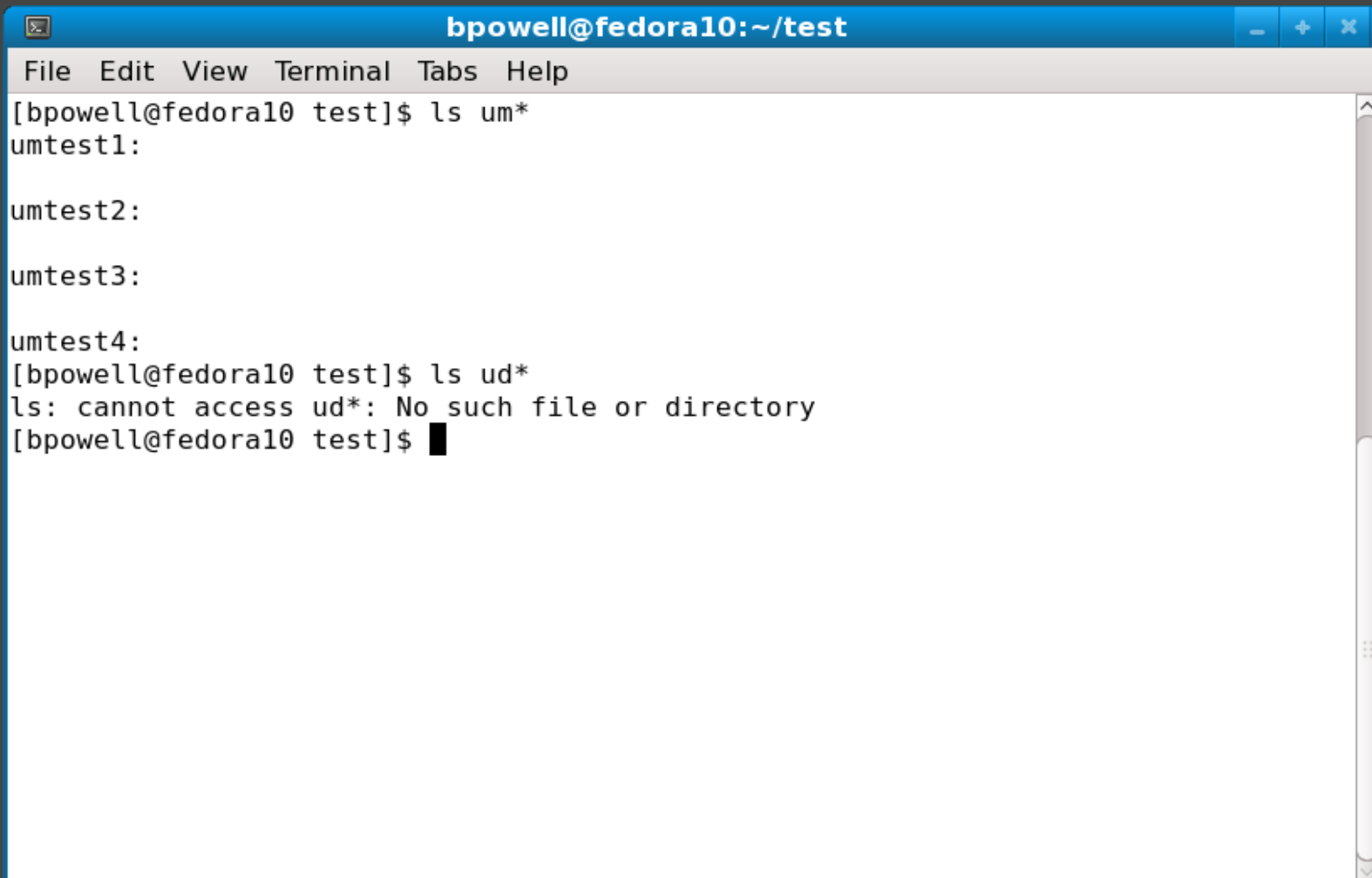
## Using conditionals with multiple commands.

The two conditionals you have at your disposal are `&&` (which means AND) and `||` (which means OR)..

## Using conditionals with multiple commands.

Before proceeding, you need to understand one more term: "exit status". When you run a command, it returns an exit status to tell the system whether it finished without any errors or not. An exit status of 0 typically means that the program completed with no errors. An exit status of 1 is the generic response meaning an error, though some programs offer more numbers.

The exit status of the last command run is stored in the \$? variable.



```
bpowell@fedora10:~/test
File Edit View Terminal Tabs Help
[bpowell@fedora10 test]$ ls um*
umtest1:

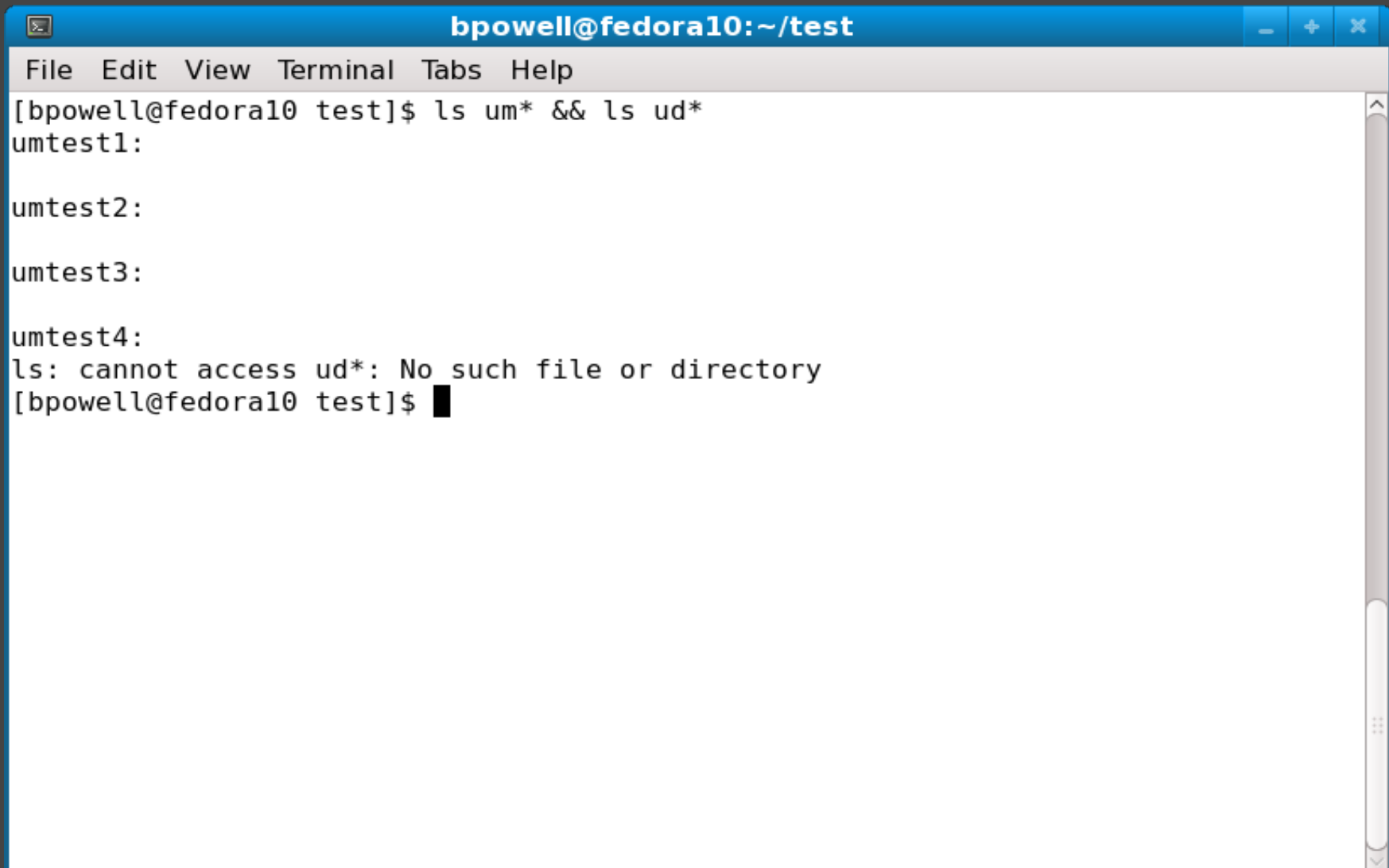
umtest2:

umtest3:

umtest4:
[bpowell@fedora10 test]$ ls ud*
ls: cannot access ud*: No such file or directory
[bpowell@fedora10 test]$
```

## Using conditionals with multiple commands.

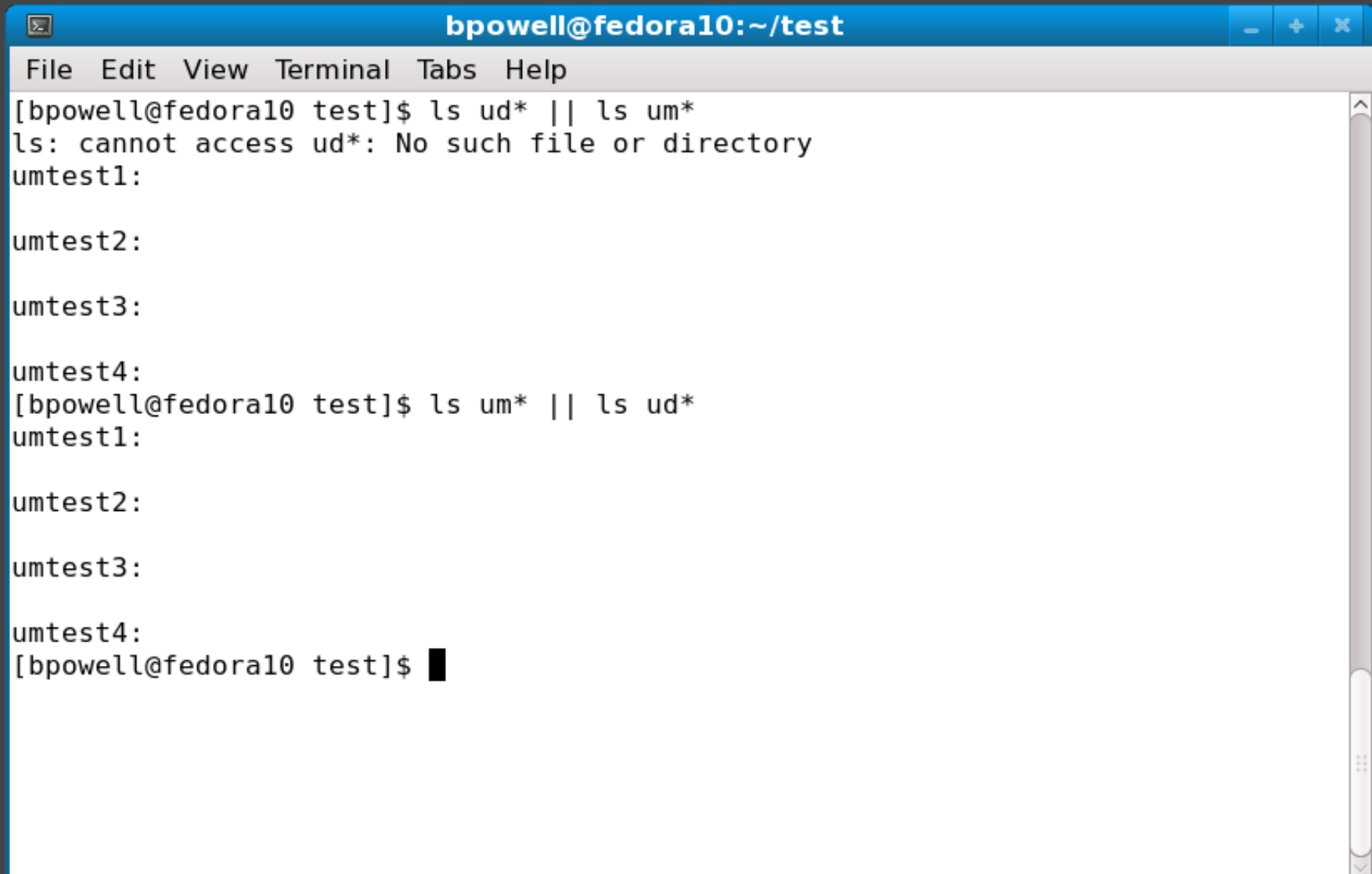
This is called an "AND" because it means "run the first command successfully AND run the second command". That is, only run the second command if the first succeeds.



```
bpowell@fedora10:~/test
File Edit View Terminal Tabs Help
[bpowell@fedora10 test]$ ls um* && ls ud*
umtest1:
umtest2:
umtest3:
umtest4:
ls: cannot access ud*: No such file or directory
[bpowell@fedora10 test]$
```

# Using conditionals with multiple commands.

This is called an "OR" because it means "run the first command successfully OR run the second command". That is only run the second command if the first command fails.



```
bpowell@fedora10:~/test
File Edit View Terminal Tabs Help
[bpowell@fedora10 test]$ ls ud* || ls um*
ls: cannot access ud*: No such file or directory
umtest1:

umtest2:

umtest3:

umtest4:
[bpowell@fedora10 test]$ ls um* || ls ud*
umtest1:

umtest2:

umtest3:

umtest4:
[bpowell@fedora10 test]$
```

## Using lists with multiple commands.

Another type of compound command is called a “list” and is created by putting the command(s) inside curly braces (`{ }`). separated by semi-colons and with a final semi-colon at the end, such as:

```
{ var=1 ; echo $var ; }
```

Using curly braces allows you to treat the commands as a unit, and any variables set inside are still usable in the main shell. They don't need to be exported.

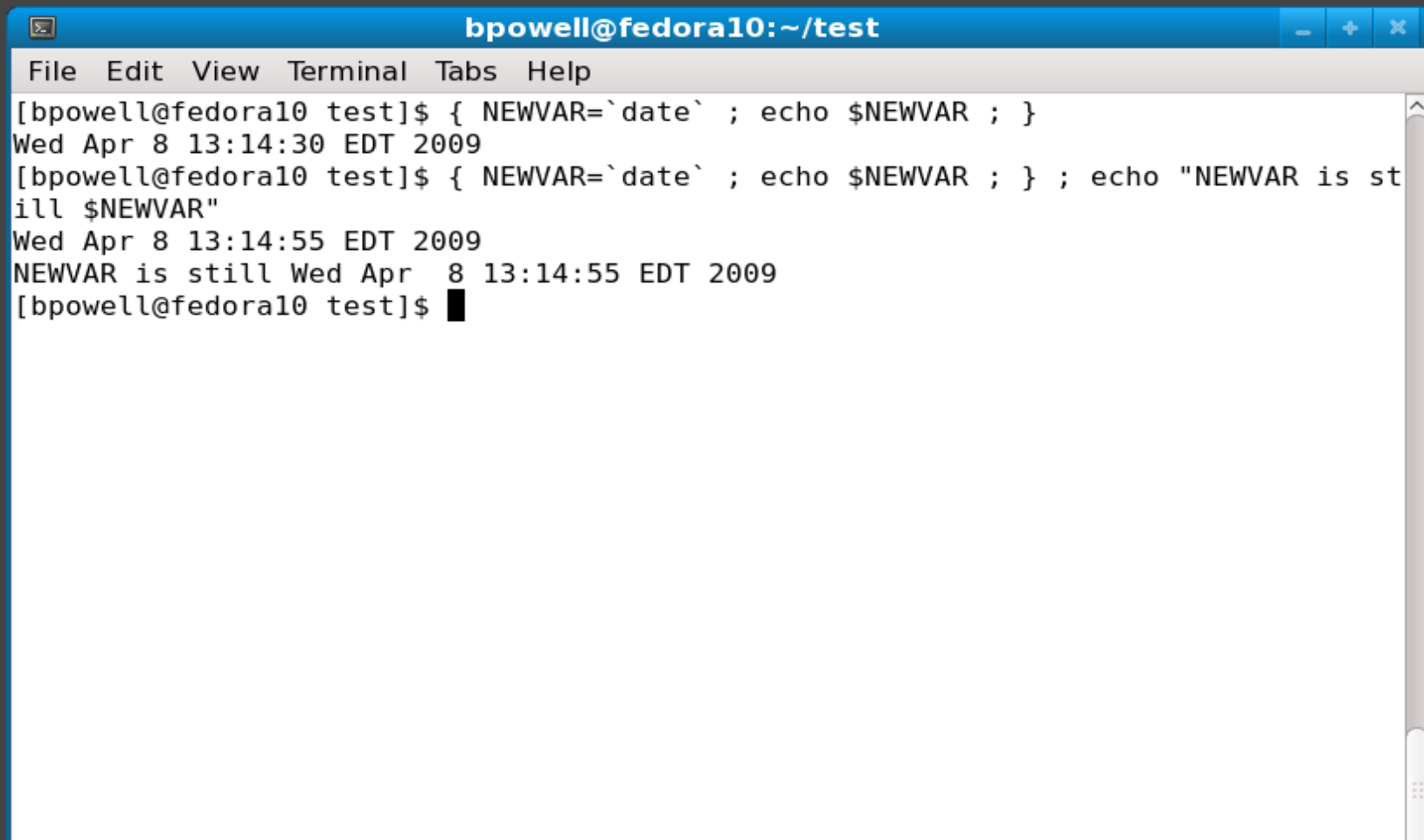
# Using lists with multiple commands.

Must be a space between the braces and the text

Must be semi-colons between each command in the list

There must be a semi-colon at the end of the last command

If you follow the list with other commands you must have a semi-colon

A terminal window titled "bpowell@fedora10:~/test" with a menu bar (File, Edit, View, Terminal, Tabs, Help). The terminal shows three lines of command execution. The first line is a list: { NEWVAR=`date` ; echo \$NEWVAR ; }. The second line is a list followed by another command: { NEWVAR=`date` ; echo \$NEWVAR ; } ; echo "NEWVAR is still \$NEWVAR". The third line is a list followed by another command: { NEWVAR=`date` ; echo \$NEWVAR ; } ; echo "NEWVAR is still Wed Apr 8 13:14:55 EDT 2009".


```
bpowell@fedora10:~/test
File Edit View Terminal Tabs Help
[bpowell@fedora10 test]$ { NEWVAR=`date` ; echo $NEWVAR ; }
Wed Apr 8 13:14:30 EDT 2009
[bpowell@fedora10 test]$ { NEWVAR=`date` ; echo $NEWVAR ; } ; echo "NEWVAR is still $NEWVAR"
Wed Apr 8 13:14:55 EDT 2009
NEWVAR is still Wed Apr 8 13:14:55 EDT 2009
[bpowell@fedora10 test]$
```

## Automating commands with scripting.

Scripting is a method of automating commands that normally would be manually entered.

Scripting, once mastered saves time and keystrokes.

## Automating commands with scripting.

A terminal window titled "bpowell@fedora10:/home/bpowell/test" with a menu bar containing "File", "Edit", "View", "Terminal", "Tabs", and "Help". The terminal shows a shell script being executed: "[root@fedora10 test]# for J in bob matt tom steve dave;do useradd \$J;done". The prompt "[root@fedora10 test]#" is followed by a black cursor block, indicating the script has completed execution. The window has standard Linux window controls (minimize, maximize, close) and a scrollbar on the right side.

```
bpowell@fedora10:/home/bpowell/test
File Edit View Terminal Tabs Help
[root@fedora10 test]# for J in bob matt tom steve dave;do useradd $J;done
[root@fedora10 test]#
```

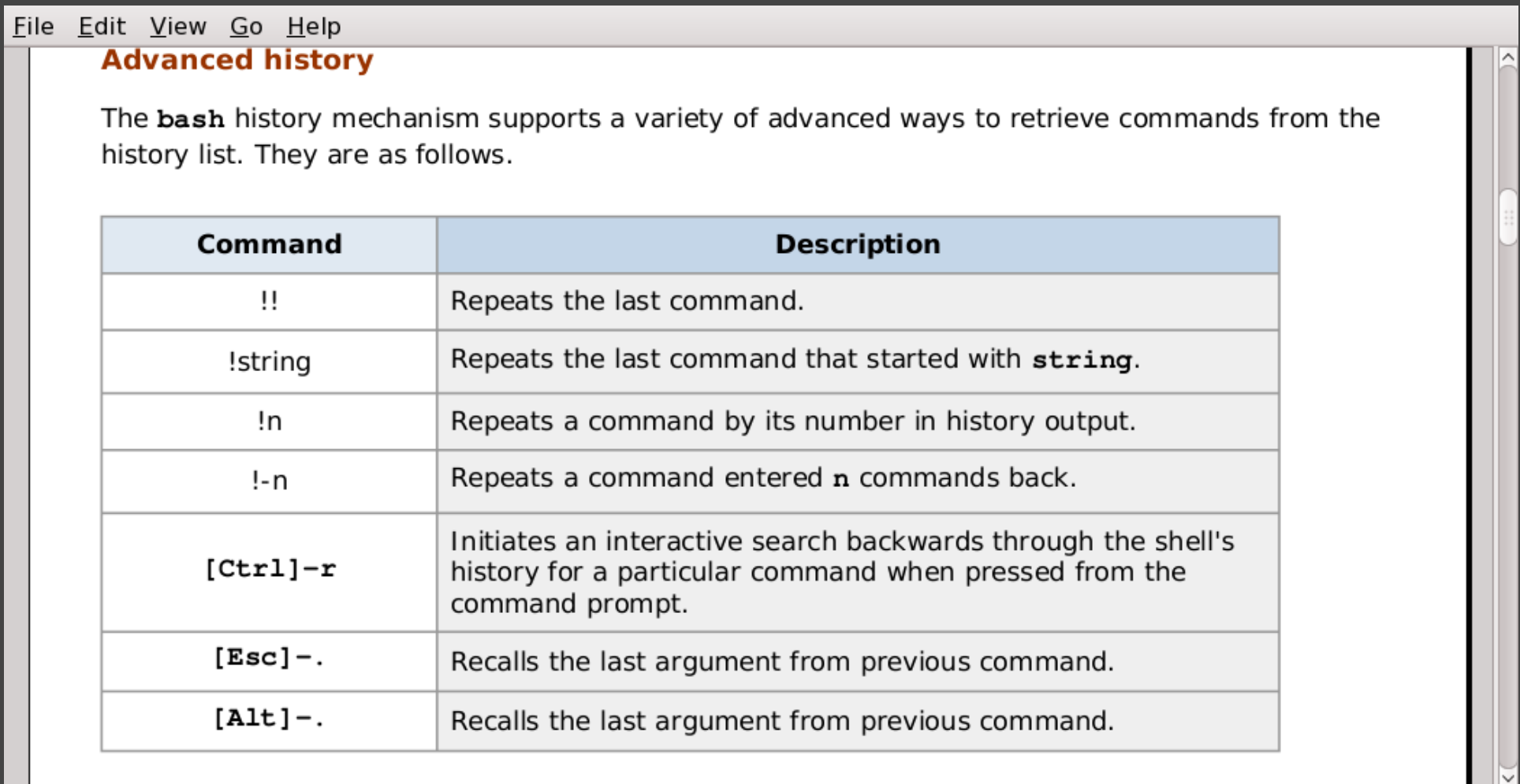
## Using history to recall previous commands.

bash stores a history of commands you have entered so that you can recall them later.

You can recall commands by pressing the up arrow key; your previous command appears on the command line. As you continue to press the up arrow key, you cycle through the commands you typed earlier, from the most recent to the oldest. You can also press the down arrow key to go back down the list if you have gone past the command you wanted to execute. This feature will save you quite a bit of time.

# Using history to recall previous commands.

The bash history mechanism supports a variety of advanced ways to retrieve commands from the history list. They are as follows:



The screenshot shows a terminal window with a menu bar (File, Edit, View, Go, Help) and a title bar. The main content is titled "Advanced history" and explains that the bash history mechanism supports various ways to retrieve commands. Below this is a table with two columns: "Command" and "Description".

Command	Description
!!	Repeats the last command.
!string	Repeats the last command that started with <b>string</b> .
!n	Repeats a command by its number in history output.
!-n	Repeats a command entered <b>n</b> commands back.
[Ctrl]-r	Initiates an interactive search backwards through the shell's history for a particular command when pressed from the command prompt.
[Esc]-.	Recalls the last argument from previous command.
[Alt]-.	Recalls the last argument from previous command.

## Using history to recall previous commands.

Use the syntax `^old^new` to repeat the last command with old changed to new.

For example,

```
$cp filter.c /usr/local/src/project
```

```
$ ^filter^frontend
```

You will get the output:

```
cp frontend.c /usr/local/src/project
```

Note that filter is replaced with frontend.

## Using history to recall previous commands.

More useful sequences:

- !\* which will repeat the last command's arguments (whereas !! does the entire command)
- !\$ to print the final argument
- !:3 to print the third argument of the previous command.

For instance:

```
$ ls /tmp/tmpdir
```

```
ls: cannot access /tmp/tmpdir: No such file or directory
```

```
$ mkdir -p !*
```

```
mkdir -p /tmp/tmpdir
```

```
$ touch file1 file2 file3
```

```
$ cp !:2 /tmp/tmpdir
```

```
cp file2 /tmp/tmpdir
```