How do I save terminal output to a file?

Asked 7 years, 7 months ago Active 1 year ago Viewed 2.8m times

1130

How do I save the output of a command to a file?

Is there a way without using any software? I would like to know how.

command-line

729

Share Improve this question Follow

edited Feb 14 '14 at 21:38
kiri
25.4k ● 16 ● 71 ● 113

asked Feb 14 '14 at 19:49
led-Zepp
11.6k ● 5 ● 13 ● 12

8 Answers

Active | Oldest | Votes

1331

Yes it is possible, just redirect the output (AKA `stdout`) to a file:

```
SomeCommand > SomeFile.txt
```

Or if you want to append data:

```
SomeCommand >> SomeFile.txt
```

If you want `stderr` as well use this:

```
SomeCommand &> SomeFile.txt
```

or this to append:

```
SomeCommand &>> SomeFile.txt
```

if you want to have both `stderr` and output *displayed on the console* ***and*** *in a file* use this:

```
SomeCommand 2>&1 | tee SomeFile.txt
```

(If you want the output only, drop the `2` above)

Share Improve this answer Follow

edited Sep 27 '20 at 8:48

Pablo Bianchi
9,048 ● 3 ● 46 ● 88

answered Feb 14 '14 at 19:52

Seth
51.6k ● 41 ● 141 ● 193

28 Note that `someCommand 2> someFile.txt` and `someCommand 2>> someFile.txt` also redirects `stterr` to someFile.txt – Slothworks Aug 29 '15 at 13:32

I'm trying to do this with gcc command but it doesn't work. It works with other commands, but not this one. It simply creates the output file with nothing inside it. – KeyC0de Sep 29 '16 at 13:40

@Nik-Lz Often this is because the command is sending all its output on stderr. If gcc is generating error messages, this seems likely. See Slothworks comment for how to capture stderr instead of stdout. – Jonathan Hartley Sep 14 '17 at 13:29

1 NB: to get the output of the `make` command into a file it requires this syntax instead: `make > someFile.txt 2>&1` (source: linuxquestions.org/questions/linux-newbie-8/...) – Gabriel Staples Jan 15 '18 at 0:49

1 I have a problem that it stops writing when the file reaches about 8MB. Is this a known limit? – relG Oct 27 '18 at 8:02

1320

To write the output of a command to a file, there are basically 10 commonly used ways.

Overview:

> *Please note that the `n.e.` in the syntax column means "not existing".*
>
> There is a way, but it's too complicated to fit into the column. You can find a helpful link in the List section about it.

```
          || visible in terminal ||   visible in file   || existing
  Syntax  ||  StdOut  |  StdErr  ||  StdOut  |  StdErr  ||   file
==========++==========+==========++==========+==========++==========
    >     ||    no    |   yes    ||   yes    |    no    || overwrite
    >>    ||    no    |   yes    ||   yes    |    no    ||  append
          ||          |          ||          |          ||
   2>     ||   yes    |    no    ||    no    |   yes    || overwrite
   2>>    ||   yes    |    no    ||    no    |   yes    ||  append
          ||          |          ||          |          ||
   &>     ||    no    |    no    ||   yes    |   yes    || overwrite
   &>>    ||    no    |    no    ||   yes    |   yes    ||  append
          ||          |          ||          |          ||
  | tee   ||   yes    |   yes    ||   yes    |    no    || overwrite
  | tee -a ||   yes    |   yes    ||   yes    |    no    ||  append
          ||          |          ||          |          ||
 n.e. (*) ||   yes    |   yes    ||    no    |   yes    || overwrite
 n.e. (*) ||   yes    |   yes    ||    no    |   yes    ||  append
          ||          |          ||          |          ||
|& tee    ||   yes    |   yes    ||   yes    |   yes    || overwrite
|& tee -a ||   yes    |   yes    ||   yes    |   yes    ||  append
```

List:

- `command > output.txt`

  The standard output stream will be redirected to the file only, it will not be visible in the terminal. If the file already exists, it gets overwritten.

- `command >> output.txt`

  The standard output stream will be redirected to the file only, it will not be visible in the terminal. If the file already exists, the new data will get appended to the end of the file.

- `command 2> output.txt`

  The standard error stream will be redirected to the file only, it will not be visible in the terminal. If the file already exists, it gets overwritten.

- `command 2>> output.txt`

  The standard error stream will be redirected to the file only, it will not be visible in the terminal. If the file already exists, the new data will get appended to the end of the file.

- `command &> output.txt`

  Both the standard output and standard error stream will be redirected to the file only, nothing will be visible in the terminal. If the file already exists, it gets overwritten.

- `command &>> output.txt`

  Both the standard output and standard error stream will be redirected to the file only, nothing will be visible in the terminal. If the file already exists, the new data will get appended to the end of the file..

- `command | tee output.txt`

  The standard output stream will be copied to the file, it will still be visible in the terminal. If the file already exists, it gets overwritten.

- `command | tee -a output.txt`

  The standard output stream will be copied to the file, it will still be visible in the terminal. If the file already exists, the new data will get appended to the end of the file.

- **(*)**

  Bash has no shorthand syntax that allows piping only StdErr to a second command, which would be needed here in combination with `tee` again to complete the table. If you really need something like that, please look at "How to pipe stderr, and not stdout?" on Stack Overflow for some ways how this can be done e.g. by swapping streams or using process substitution.

- `command |& tee output.txt`

  Both the standard output and standard error streams will be copied to the file while still being visible in the terminal. If the file already exists, it gets overwritten.

- `command |& tee -a output.txt`

  Both the standard output and standard error streams will be copied to the file while still being visible in the terminal. If the file already exists, the new data will get appended to the end of the file.

edited May 23 '17 at 12:39
Community Bot
1

answered Feb 8 '16 at 14:52

Byte Commander ♦
96.2k ● 40 ● 254 ● 393

93 Thanks for the table, it's excellent! This should be top answer – DevShark Aug 15 '16 at 16:24

3 @karthick87 This is not really related to the question about redirecting output to a file, because it just redirects one stream to another. `2>&1` redirects STDERR to STDOUT, `1>&2` redirects STDOUT to STDERR and `3>&1` would redirect stream 3 to STDERR. – Byte Commander ♦ Sep 19 '16 at 16:42

23 Just a note that '|&' wasn't working for me on macOS. This is due to it having an older version of bash (I think). The less elegant '2>&1 |' works fine though – Danny Parker May 9 '17 at 10:01

2 @ByteCommander I get the error: `sh: 1: Syntax error: "&" unexpected` when I use `|& tee` from a Python script in a c9.io server. It seems a different shell is being used. `echo $SHELL` shows `/bin/bash` and `$SHELL --version` shows version 4.3.11(1)-release. I tried `#!/bin/bash` in my python script but I still get `sh: 1: Syntax error`. I got what I needed so I'm giving up on sorting the weirdness between `sh` and `bash` on my server. Thanks. – samkhan13 Jan 28 '18 at 3:09

1 @samkhan13 looks like you are running `sh` and not `bash` (or maybe `bash` in `sh` mode...). You can check what exactly your current shell process is using `ps -p $$ -o cmd=`, because `echo $SHELL` is unreliable and will show you your login shell, ignoring whether you might have started a different subshell. – Byte Commander ♦ Jan 28 '18 at 12:35

125

You can also use `tee` to send the output to a file:

```
command | tee ~/outputfile.txt
```

A slight modification will catch stderr as well:

```
command 2>&1 | tee ~/outputfile.txt
```

or slightly shorter and less complicated:

```
command |& tee ~/outputfile.txt
```

`tee` is useful if you want to be able to **capture command output while also viewing it live**.

Share Improve this answer Follow

edited Oct 26 '16 at 10:20

David Foerster
33.8k ● 54 ● 85 ● 137

answered Jun 20 '14 at 4:45

Aaron
6,375 ● 5 ● 29 ● 45

It says that the & is unexpected, and doesn't write the log at the same time as the command runs. I am using this in a bash file however, does that make any difference? – tim687 Apr 6 '16 at 13:51

@tim687 I have removed that edit. Sorry about that...wasn't a part of my original answer. – Aaron Apr 6 '16 at 14:11

1 how do I interpret meaning of `2>&1`? – Maha Jul 15 '16 at 7:47

2 @Mahesha999 2 is the file descriptor for STDERR, and 1 is for STDOUT. So that 2>&1 sends STDERR to STDOUT. This SO question explains it pretty well: stackoverflow.com/questions/818255/... – Aaron Jul 15 '16 at 7:53

1 Instead of `2>&1 |`, one can also simply use `|&`. – Byte Commander ♦ Oct 7 '16 at 17:47

20

You can redirect the command output to a file:

```
your_command >/path/to/file
```

To append the command output to a file instead of overwriting it, use:

```
your_command >>/path/to/file
```

Share Improve this answer Follow

edited Sep 3 '17 at 14:38

Eliah Kagan

110k ● 51 ● 301 ● 461

answered Feb 14 '14 at 19:52

chaos

24.7k ● 10 ● 69 ● 72

Thanks a lot ! is there any Limits ? like the max size of the file ? – led-Zepp Feb 14 '14 at 19:55

3 The max file size is just limited through the file system – chaos Feb 14 '14 at 19:59

This answer will not save stderr. Use &> , see stackoverflow.com/questions/637827/... and tldp.org/LDP/abs/html/io-redirection.html – Panther Feb 14 '14 at 20:16

3 The OP never asked to save stderr – chaos Feb 14 '14 at 20:18

It says "No such file or directory". Is it possible to also create the directories automatically? – Qwerty May 22 '19 at 9:27

An enhancement to consider -

19

Various scripts will inject color codes into the output which you may not want cluttering up your log file.

To fix this, you can use the program sed to strip out those codes. Example:

```
command 2>&1 | sed -r 's/'$(echo -e "\033")'\[[0-9]{1,2}(;([0-9]{1,2})?)?[mK]//g' | tee
~/outputfile.txt
```

Share Improve this answer Follow

answered Jul 8 '14 at 20:57

Sean Huber

291 ● 4 ● 6

1 How to save the output in a way that colours are conserved ? I would like to import the result of a command in libreoffice and keep the colours. – madrang May 12 '15 at 6:36

1 @madrang: I only read your comment now but you may find this answer useful. – Sylvain Pineau Sep 21 '15 at 10:41

Oh, almost exactly what I am looking for. How to print also on screen the output? – Sigur Dec 23 '16 at 22:10

1 Note that many commands that produce colorized output, such as `ls` and `grep`, support `--color=auto`, which outputs color codes only if standard output is a terminal. – Eliah Kagan Sep 3 '17 at 14:37

10

+50

`some_command | tee command.log` and `some_command > command.log` have the issue that they do not save the command output to the `command.log` file in real-time.

To avoid that issue and save the command output in real-time, you may append `unbuffer`, which comes with the `expect` package.

Example:

```
sudo apt-get install expect
unbuffer some_command | tee command.log
```

```
unbuffer some_command > command.log
```

Assuming `log.py` contains:

```
import time
print('testing')
time.sleep(100) # sleeping for 100 seconds
```

you can run `unbuffer python log.py | tee command.log` or `unbuffer python log.py > command.log`

More information: How can I save a command output to a file in real-time?

Share Improve this answer Follow

answered Jul 4 '18 at 20:54

Franck Dernoncourt
2,942 ● 6 ● 33 ● 52

1 They do save the output as they receive it, the problem is that python turns on buffering when the output is not to a TTY. Other options for disabling this in Python: stackoverflow.com/q/107705/2072269 – muru Jul 5 '18 at 2:22

1 Thanks! Spent so long looking for this, and this is exactly what I needed. Works with the stdout from the Google Assistant scripts. – anonymous2 Dec 26 '19 at 2:23

You don't need to install expect, stdbuf does the same thing and is generally already there in your distribution. You can "line buffer", which still has the advantage of some buffering and saves time, or completely "unbuffer". – Zakhar Aug 23 '20 at 18:06

10

For `cron` jobs etc you want to avoid the Bash extensions. The equivalent POSIX `sh` redirection operators are

```
Bash          POSIX
------------  --------------
foo &> bar    foo >bar 2>&1
foo &>> bar   foo >>bar 2>&1
foo |& bar    foo 2>&1 | bar
```

You'll notice that the POSIX facility is in some sense simpler and more straightforward. The `&>` syntax was borrowed from `csh` which should already convince you that it's a bad idea.

Share Improve this answer Follow

edited May 1 '19 at 11:09

answered Apr 11 '18 at 12:25

tripleee
1,189 ● 2 ● 13 ● 19

10

There are two different questions here. The first is in the title:

> How do I save terminal output to a file?

The second question is in the body:

> How do I save the output of a command to a file?

All the answers posted here address the second question but none address the first question which has a

great answer in **Unix & Linux**:

- Save all the terminal output to a file

This answer uses a little known command called `script` which saves all your shell's output to a text file until you type `exit`. The command output still appears on your screen but **also appears in the text file**.

The process is simple. Use:

```
$ script ~/outputfile.txt
Script started, file is /home/rick/outputfile.txt
$ command1
$ command2
$ command3
$ exit
exit
Script done, file is /home/rick/outputfile.txt
```

Then look at your recorded output of commands 1, 2 & 3 with:

```
cat ~/outputfile.txt
```

This is similar to earlier answer of:

```
command |& tee ~/outputfile.txt
```

- But you don't have to use `|& tee ~/outputfile.txt` after each `commnd`.
- The `script` command has added benefit (or disadvantage) of reloading `~/.bashrc` when it starts.
- The `script` command shows the command prompt (`$PS1`) followed by the command(s) you entered.
- The `script` command records all the details in full color.

Share Improve this answer Follow

answered Dec 27 '19 at 1:38

WinEunuuchs2Unix

87.7k • 27 • 181 • 361

1 +1 :-) I use `script` in the following context: Wake me up when a slow command line process wants my attention? – sudodus Dec 31 '19 at 0:09

1 @sudodus +1 on your linked answer there. I would never have dreamed of using `script` for such an application :) – WinEunuuchs2Unix Dec 31 '19 at 0:16

`script` is brilliant, thank you so much! – Stromael Dec 14 '20 at 14:30

This looks promising but didn't work for me with Cygwin. `Script started on 2021-08-20 11:14:49+10:00 [TERM="xterm" TTY="/dev/cons0" COLUMNS="80" LINES="24"]` *(script is empty)* `Script done on 2021-08-20 11:14:49+10:00 [COMMAND_EXIT_CODE="0"]` – Greg Aug 20 at 9:04