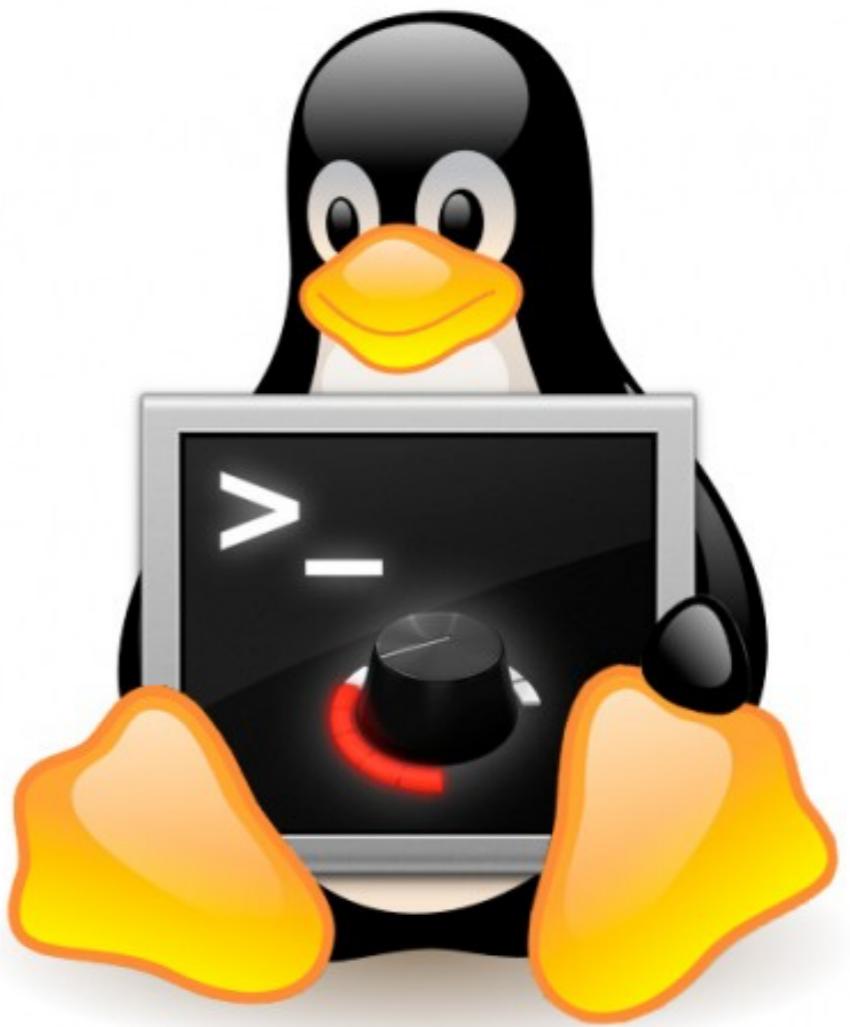


BIO 294: Bioinformatics for comparative and evolutionary genomics



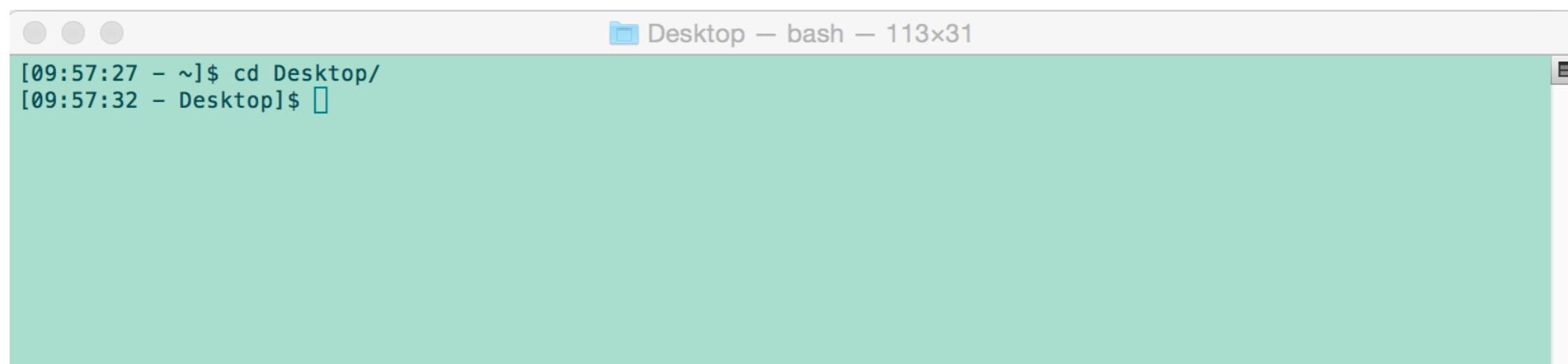
University of
Zurich^{UZH}

How to use a terminal ?



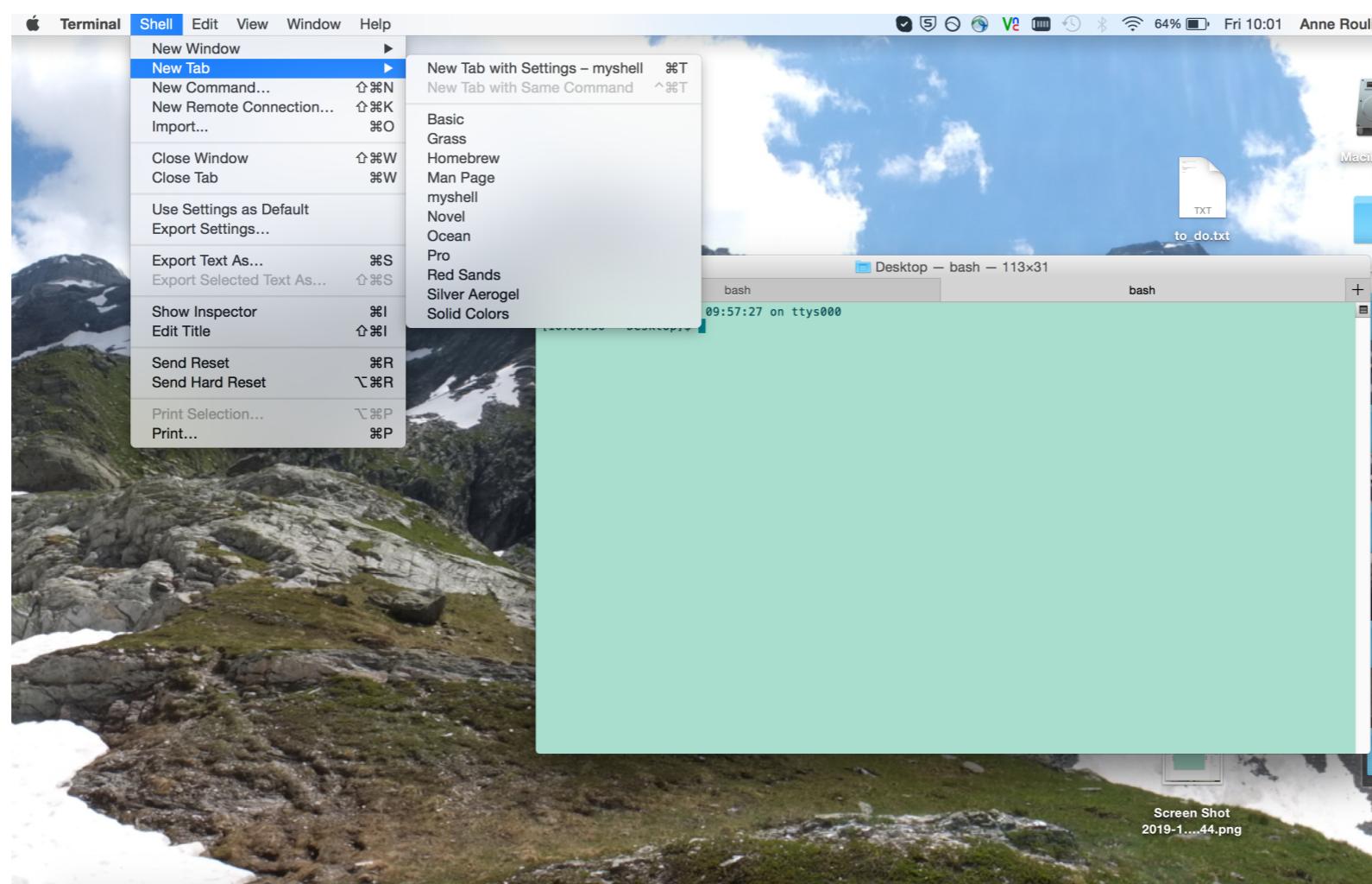
The **shell** is a program which processes commands and returns output. Most shells also manage foreground and background processes and command line editing. These features are standard in bash, the most common shell in modern linux systems.

A **terminal** refers to a wrapper program which runs a shell.

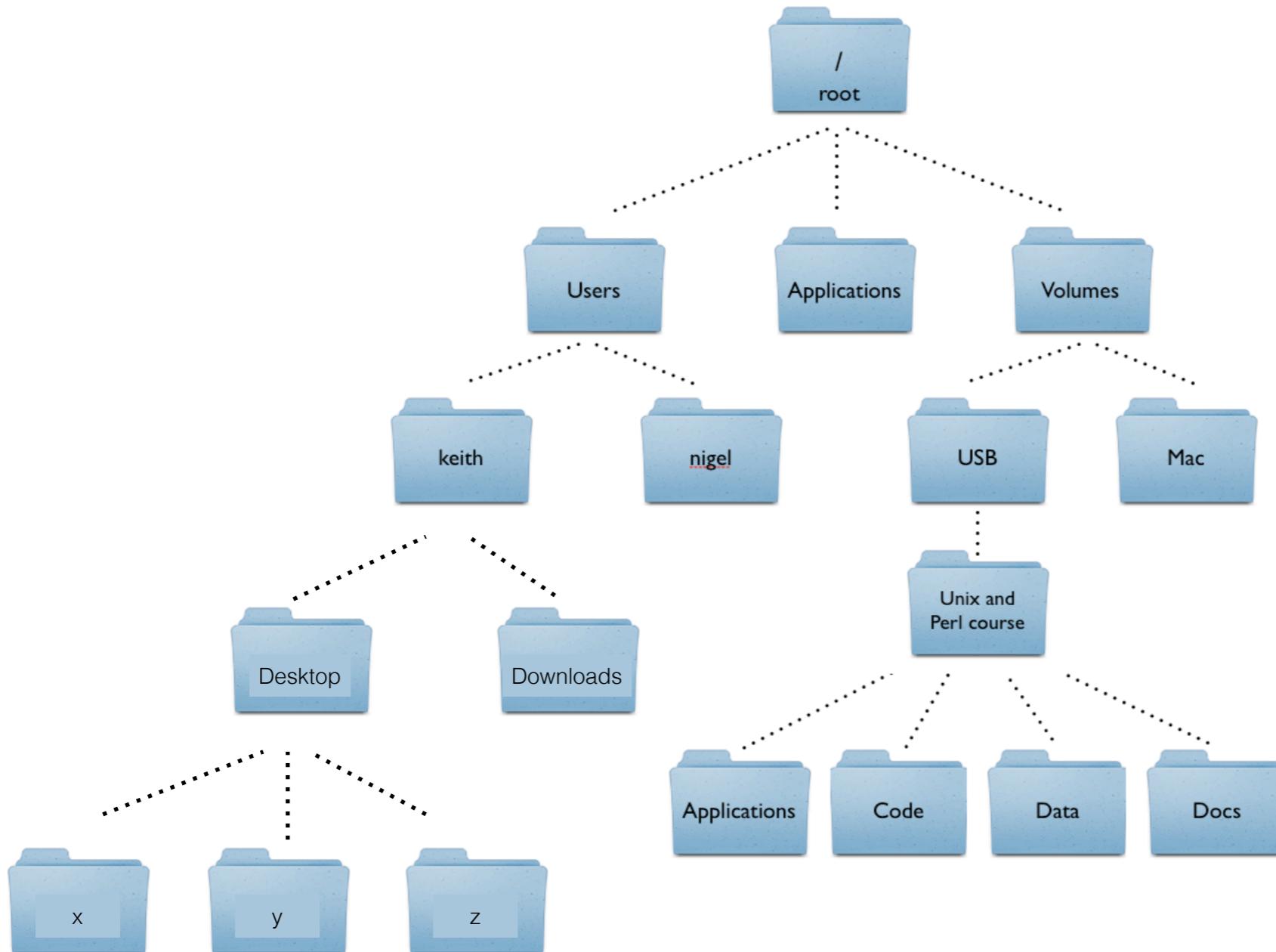


You can have:

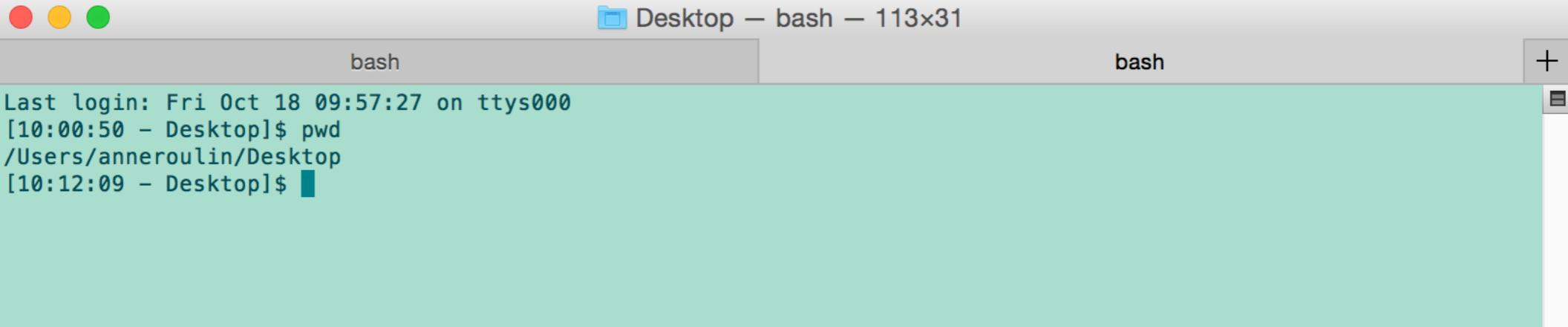
- have multiple terminal windows on screen (see ‘Shell’ menu)
- have multiple tabs open within each window. There will be many situations where it will be useful to have multiple terminals.



Unix keeps files arranged in a hierarchical structure

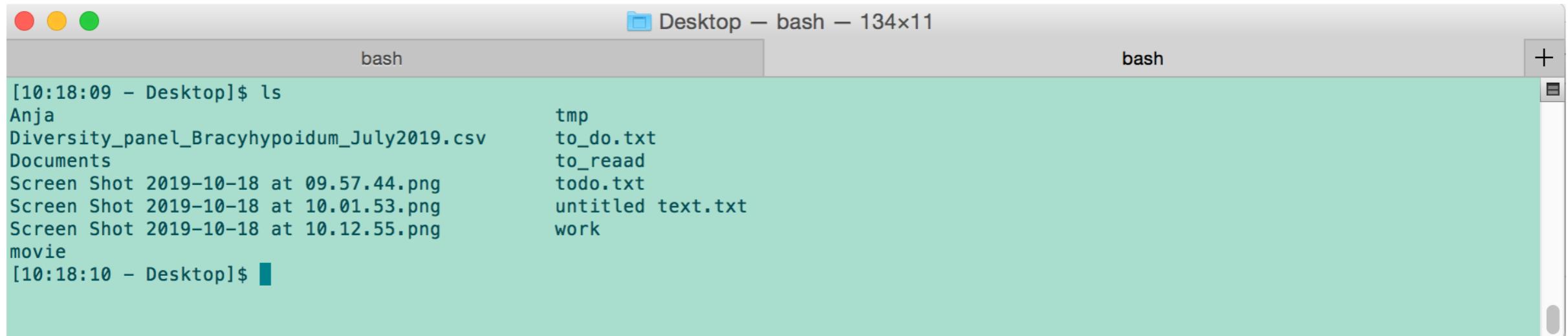


Finding out where you are: pwd (print working directory)



```
Last login: Fri Oct 18 09:57:27 on ttys000
[10:00:50 - Desktop]$ pwd
/Users/anneroulin/Desktop
[10:12:09 - Desktop]$
```

List the content of your directory: ls

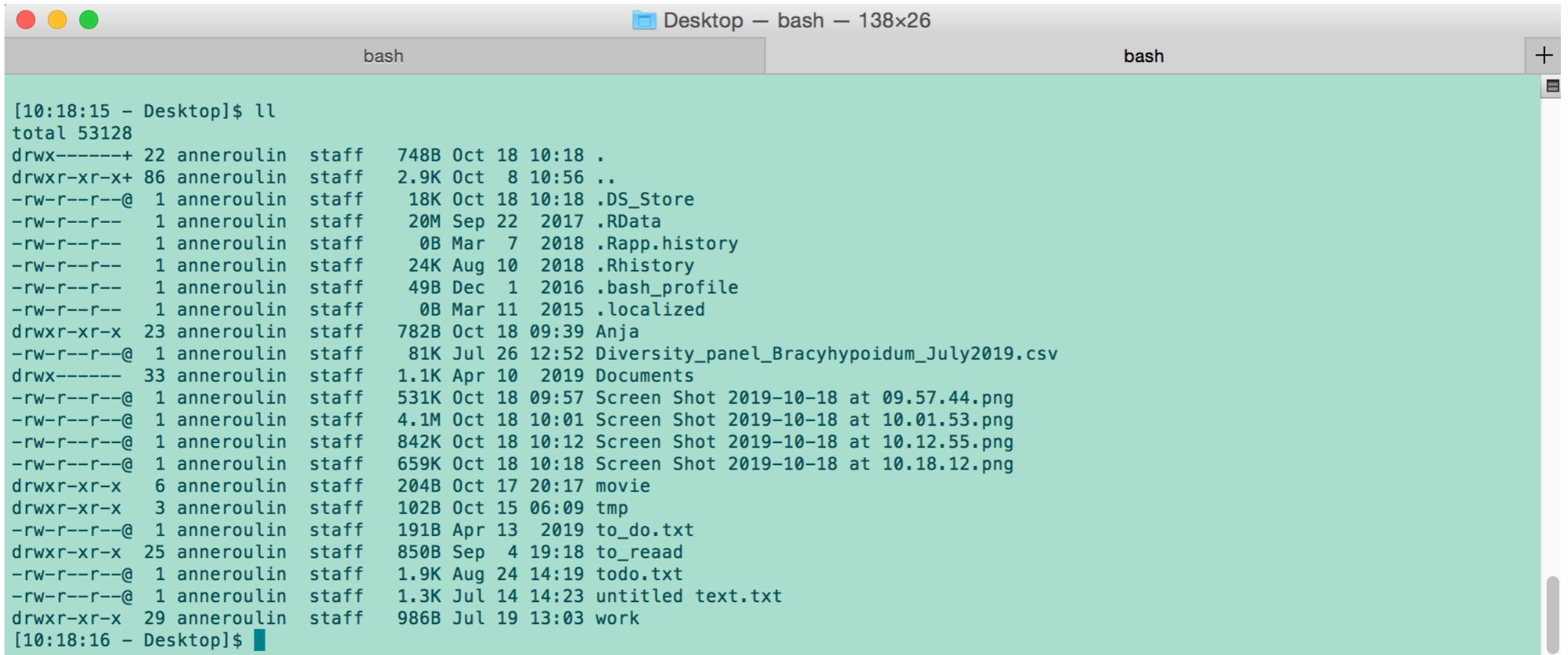


The screenshot shows a single terminal window titled "Desktop – bash – 134x11". The window has three tabs, all labeled "bash". The leftmost tab is active and displays the output of the "ls" command. The output lists several files and directories:

```
[10:18:09 - Desktop]$ ls
Anja
Diversity_panel_Brachypodium_July2019.csv
Documents
Screen Shot 2019-10-18 at 09.57.44.png
Screen Shot 2019-10-18 at 10.01.53.png
Screen Shot 2019-10-18 at 10.12.55.png
movie
[10:18:10 - Desktop]$
```

On the right side of the terminal window, there are two other tabs, both also labeled "bash". The top one is titled "Desktop – bash – 134x11" and the bottom one is partially visible.

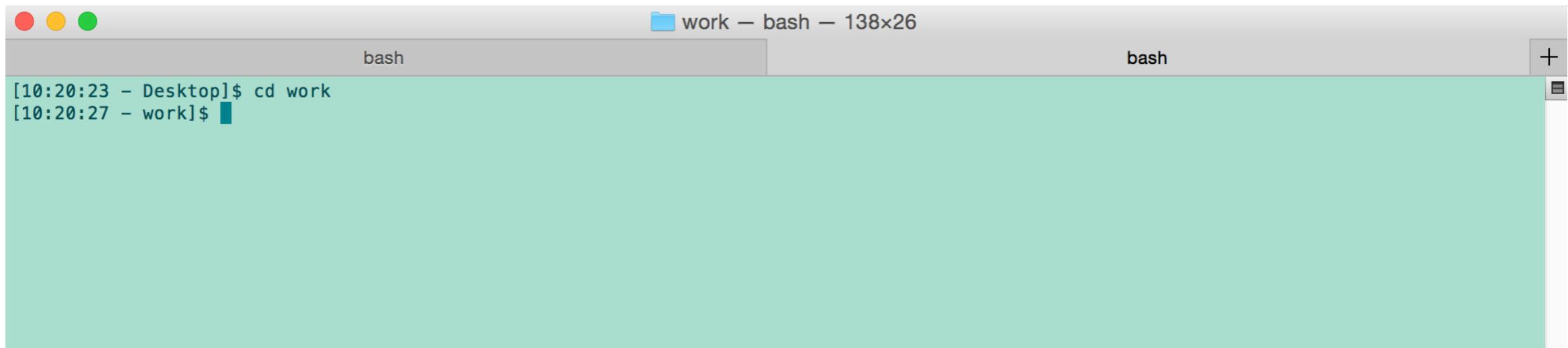
List the content of your directory: ll



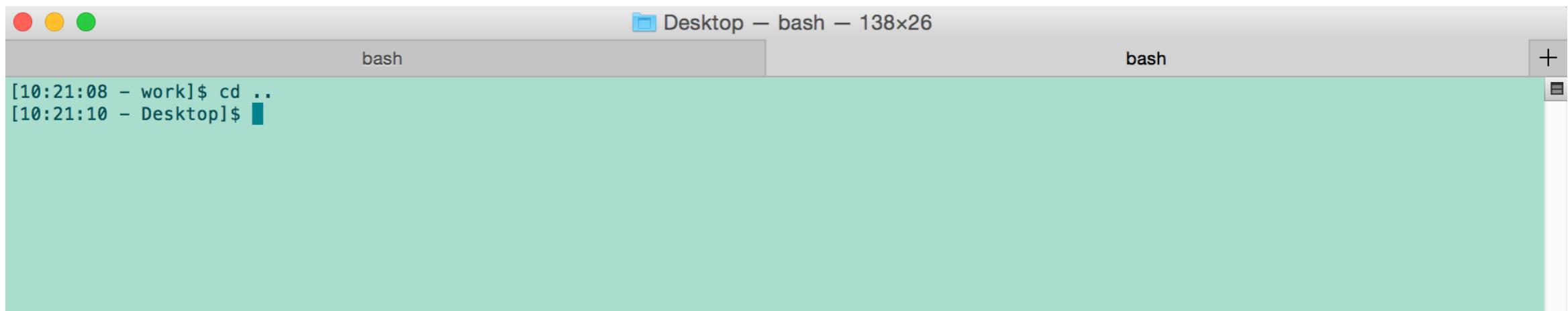
The screenshot shows a macOS terminal window titled "Desktop – bash – 138x26". The window contains the output of the "ll" command, listing the contents of the current directory. The output is as follows:

```
[10:18:15 - Desktop]$ ll
total 53128
drwx-----+ 22 anneroulin  staff  748B Oct 18 10:18 .
drwxr-xr-x+ 86 anneroulin  staff  2.9K Oct  8 10:56 ..
-rw-r--r--@  1 anneroulin  staff   18K Oct 18 10:18 .DS_Store
-rw-r--r--   1 anneroulin  staff   20M Sep 22 2017 .RData
-rw-r--r--   1 anneroulin  staff    0B Mar  7 2018 .Rapp.history
-rw-r--r--   1 anneroulin  staff   24K Aug 10 2018 .Rhistory
-rw-r--r--   1 anneroulin  staff   49B Dec  1 2016 .bash_profile
-rw-r--r--   1 anneroulin  staff    0B Mar 11 2015 .localized
drwxr-xr-x  23 anneroulin  staff  782B Oct 18 09:39 Anja
-rw-r--r--@  1 anneroulin  staff   81K Jul 26 12:52 Diversity_panel_Bracyhypoidum_July2019.csv
drwx----- 33 anneroulin  staff  1.1K Apr 10 2019 Documents
-rw-r--r--@  1 anneroulin  staff   531K Oct 18 09:57 Screen Shot 2019-10-18 at 09.57.44.png
-rw-r--r--@  1 anneroulin  staff   4.1M Oct 18 10:01 Screen Shot 2019-10-18 at 10.01.53.png
-rw-r--r--@  1 anneroulin  staff   842K Oct 18 10:12 Screen Shot 2019-10-18 at 10.12.55.png
-rw-r--r--@  1 anneroulin  staff   659K Oct 18 10:18 Screen Shot 2019-10-18 at 10.18.12.png
drwxr-xr-x  6 anneroulin  staff  204B Oct 17 20:17 movie
drwxr-xr-x  3 anneroulin  staff  102B Oct 15 06:09 tmp
-rw-r--r--@  1 anneroulin  staff   191B Apr 13 2019 to_do.txt
drwxr-xr-x 25 anneroulin  staff  850B Sep  4 19:18 to_reaad
-rw-r--r--@  1 anneroulin  staff   1.9K Aug 24 14:19 todo.txt
-rw-r--r--@  1 anneroulin  staff   1.3K Jul 14 14:23 untitled text.txt
drwxr-xr-x 29 anneroulin  staff  986B Jul 19 13:03 work
[10:18:16 - Desktop]$
```

Changing directory: cd

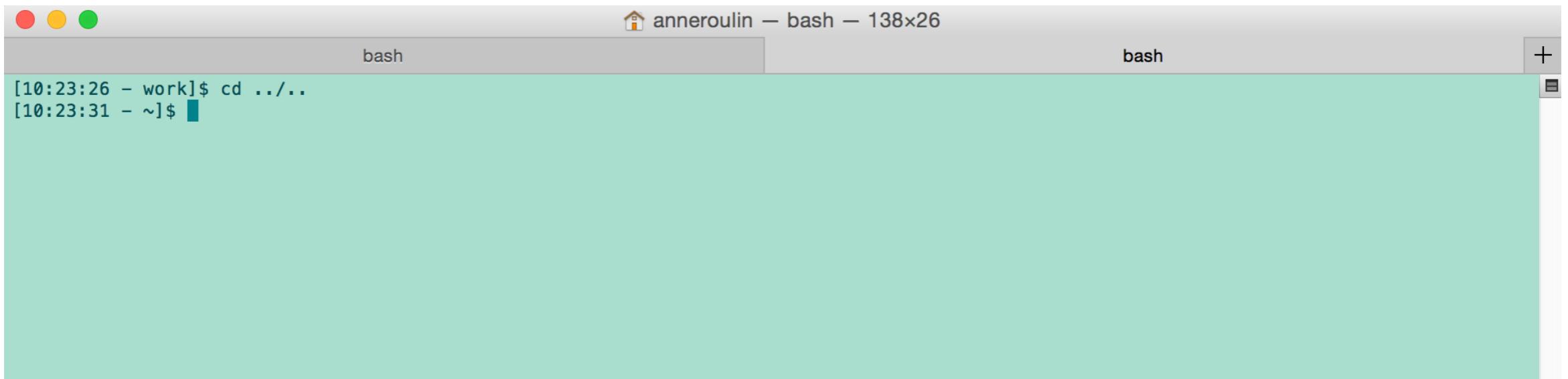


```
[10:20:23 - Desktop]$ cd work
[10:20:27 - work]$
```



```
[10:21:08 - work]$ cd ..
[10:21:10 - Desktop]$
```

Changing directory: cd



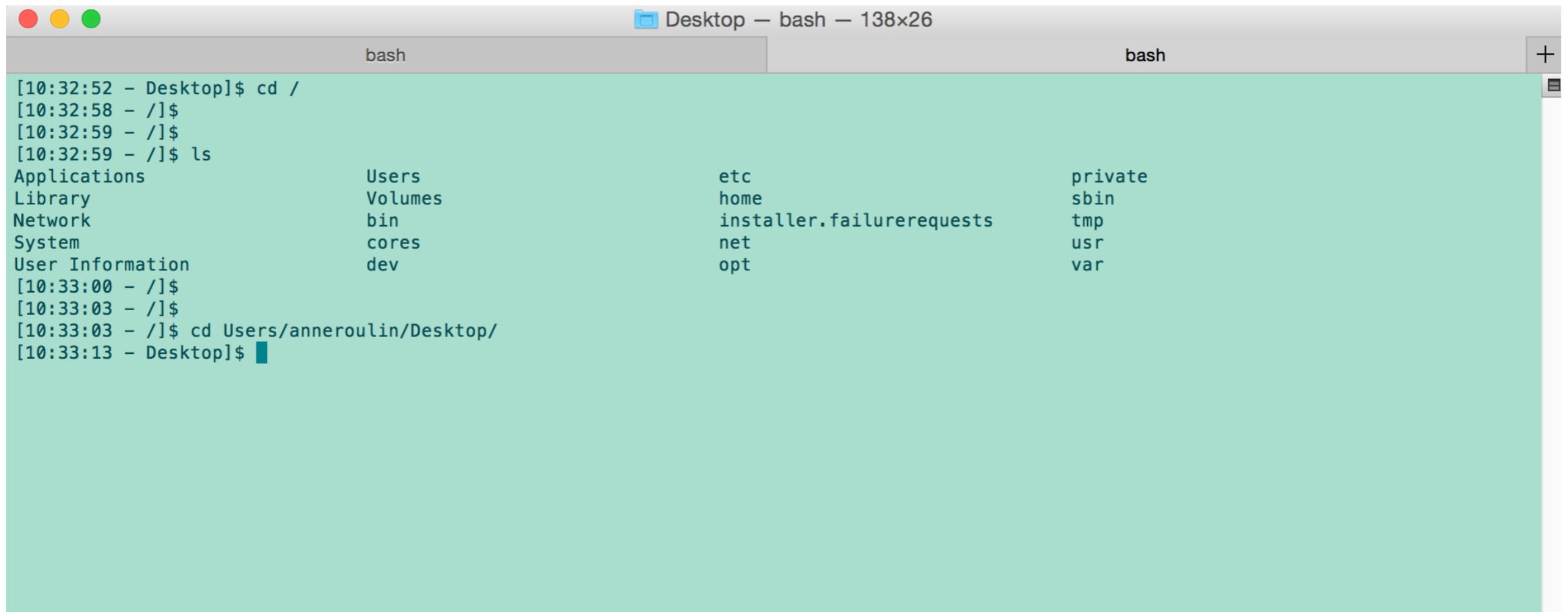
A screenshot of a macOS terminal window titled "anneroulin — bash — 138x26". The window contains two tabs, both labeled "bash". The left tab shows the command "[10:23:26 - work]\$ cd ../../" being typed. The right tab shows the result "[10:23:31 - ~]\$". The terminal has its characteristic teal background and white text.

Nb: use



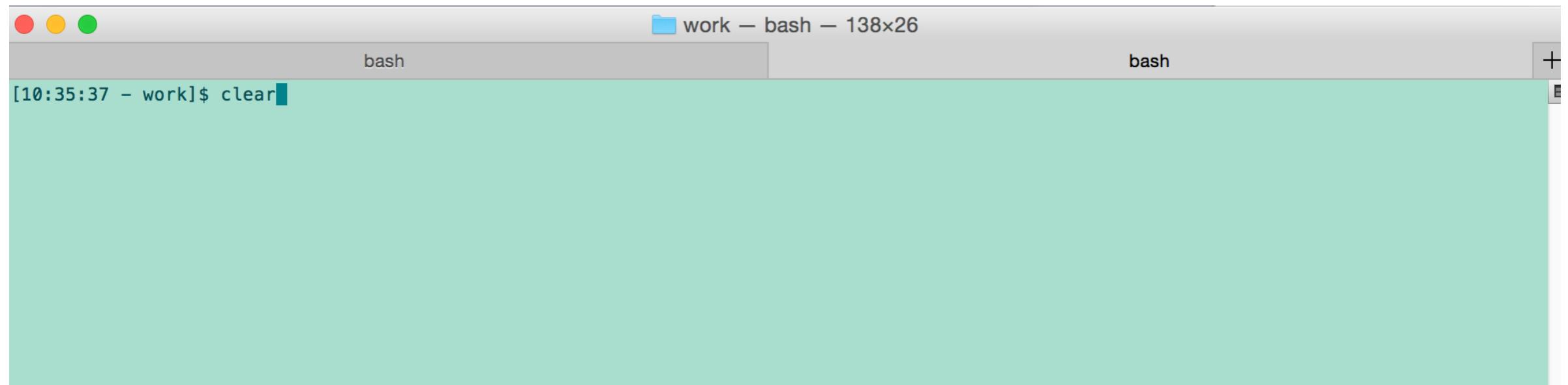
to complete the path you are looking for

Back to the root: cd /

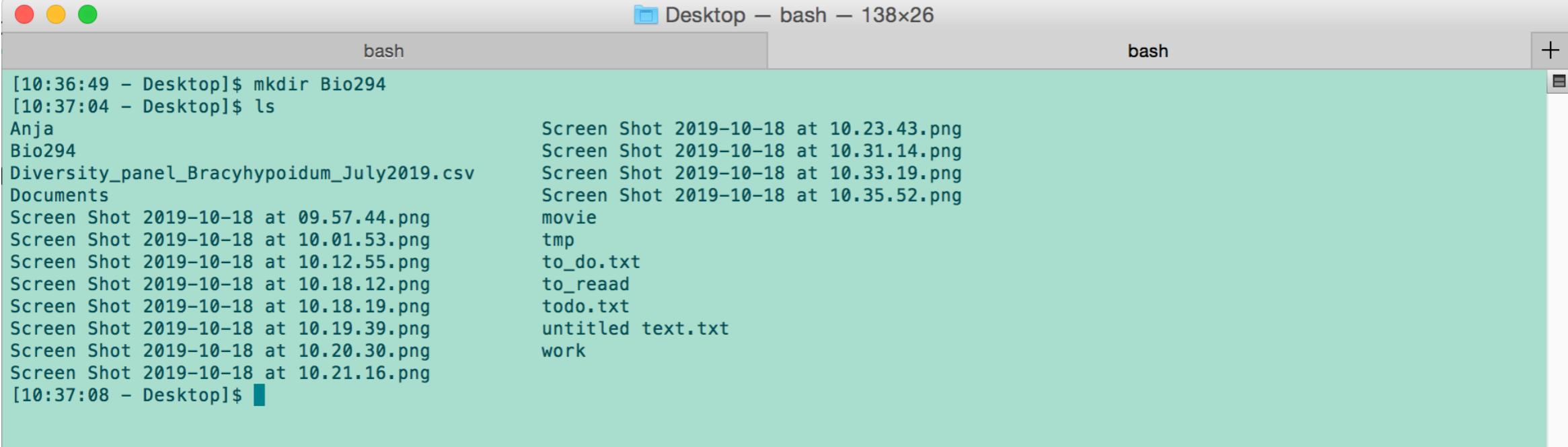


```
[10:32:52 - Desktop]$ cd /
[10:32:58 - /]$
[10:32:59 - /]$
[10:32:59 - /]$/ ls
Applications          Users
Library              Volumes
Network              bin
System               cores
User Information     dev
[10:33:00 - /]$
[10:33:03 - /]$
[10:33:03 - /]$/ cd Users/anneroulin/Desktop/
[10:33:13 - Desktop]$
```

Clean your terminal: clear



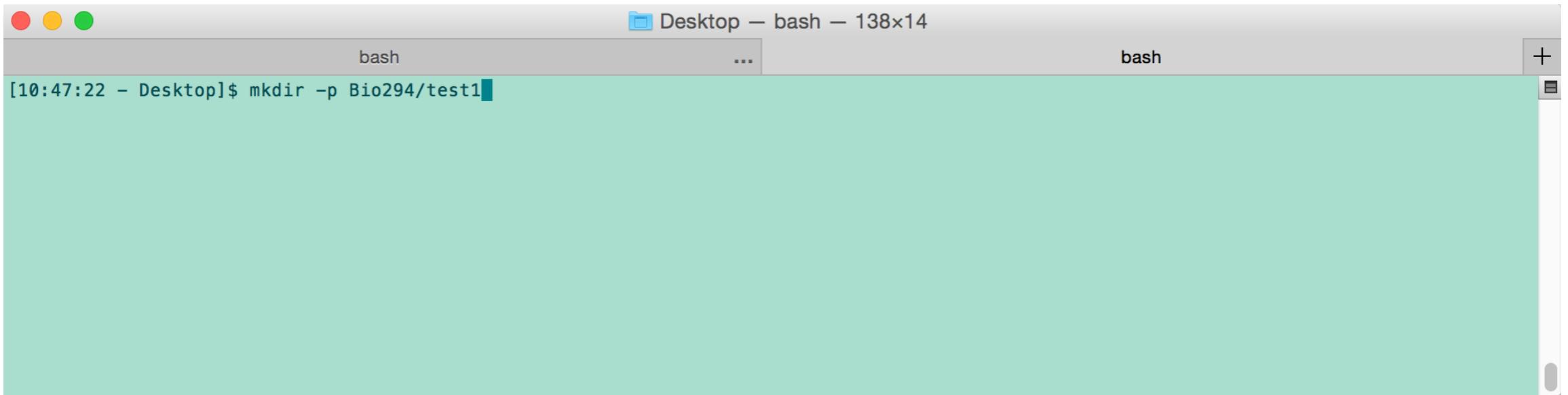
Make a new directory: mkdir



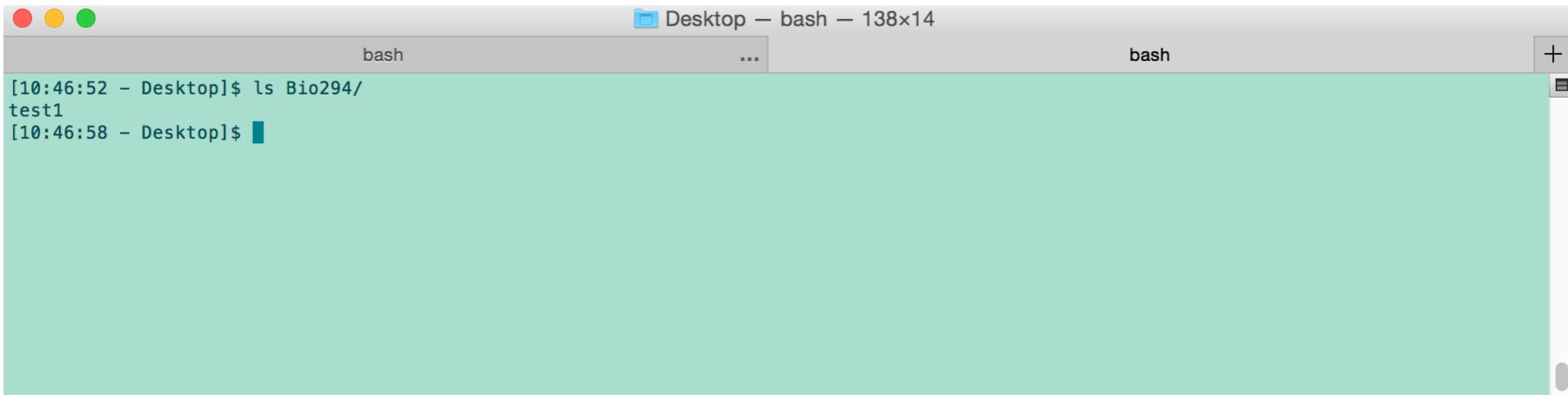
```
[10:36:49 - Desktop]$ mkdir Bio294
[10:37:04 - Desktop]$ ls
Anja
Bio294
Diversity_panel_Brachypodium_July2019.csv
Documents
Screen Shot 2019-10-18 at 09.57.44.png
Screen Shot 2019-10-18 at 10.01.53.png
Screen Shot 2019-10-18 at 10.12.55.png
Screen Shot 2019-10-18 at 10.18.12.png
Screen Shot 2019-10-18 at 10.18.19.png
Screen Shot 2019-10-18 at 10.19.39.png
Screen Shot 2019-10-18 at 10.20.30.png
Screen Shot 2019-10-18 at 10.21.16.png
[10:37:08 - Desktop]$ 

[Desktop] Desktop – bash – 138x26
bash
+ [Desktop] Desktop – bash – 138x26
bash
```

Make a new directory within a new directory: mkdir -p

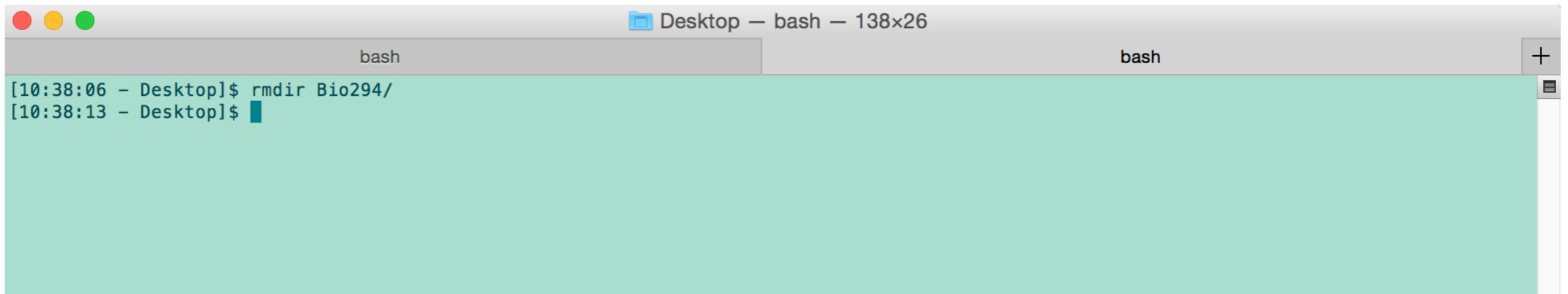


```
[10:47:22 - Desktop]$ mkdir -p Bio294/test1
```



```
[10:46:52 - Desktop]$ ls Bio294/
test1
[10:46:58 - Desktop]$
```

Delete a directory: rm

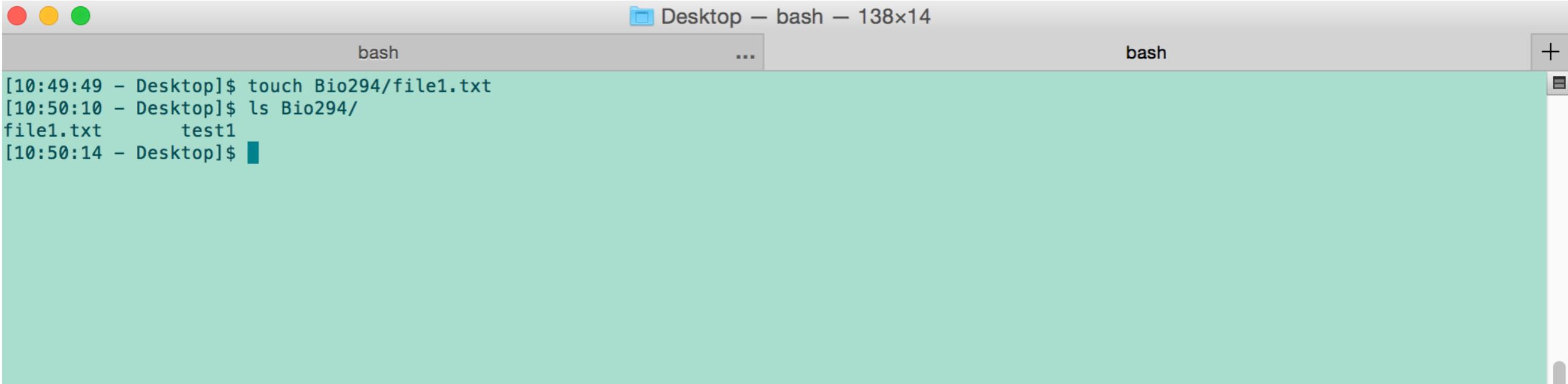


```
[10:38:06 - Desktop]$ rm -r Bio294/  
[10:38:13 - Desktop]$
```



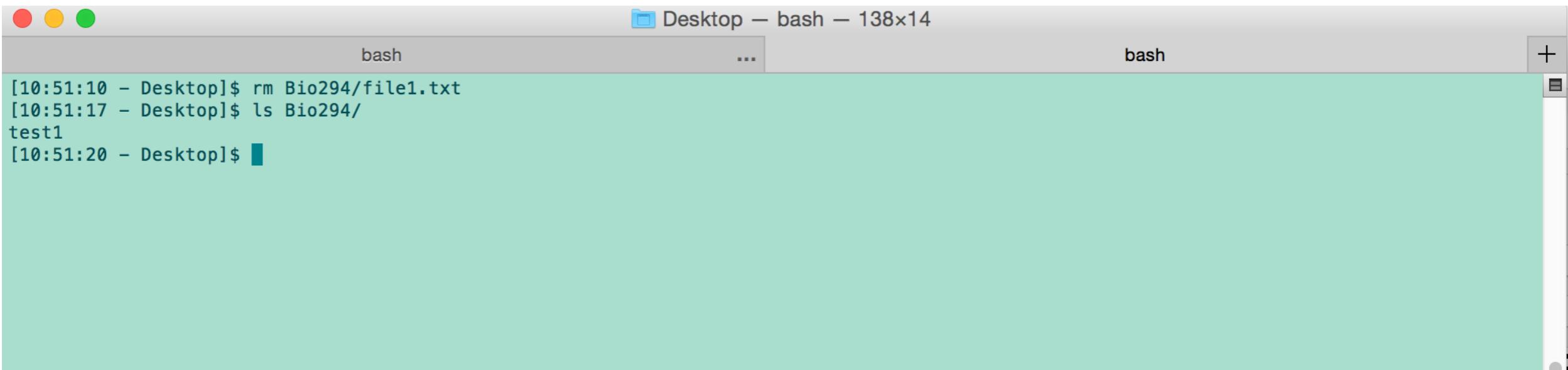
You will not be asked “are you sure you want to remove your directory”

Create a new file: touch



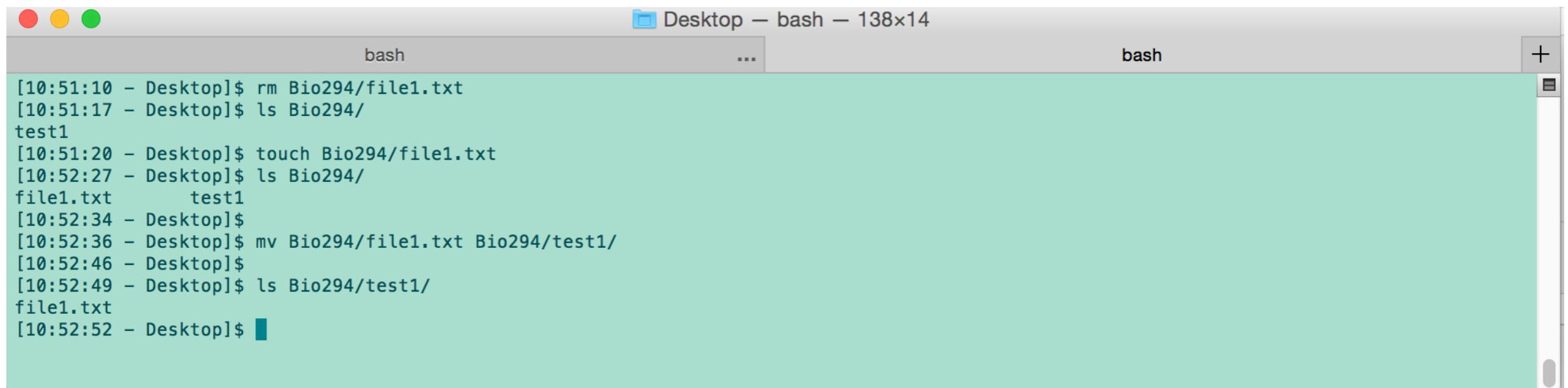
```
[10:49:49 - Desktop]$ touch Bio294/file1.txt
[10:50:10 - Desktop]$ ls Bio294/
file1.txt      test1
[10:50:14 - Desktop]$
```

Delete a file: rm



```
[10:51:10 - Desktop]$ rm Bio294/file1.txt
[10:51:17 - Desktop]$ ls Bio294/
test1
[10:51:20 - Desktop]$
```

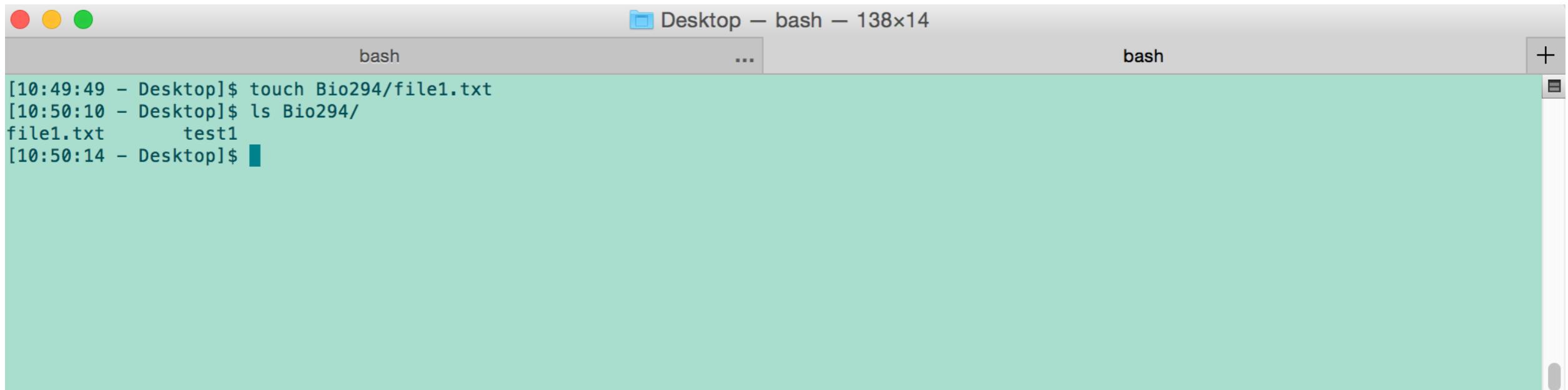
Move a file: mv



The screenshot shows a single terminal window titled "Desktop – bash – 138x14". The window has three tabs, all labeled "bash". The terminal content is as follows:

```
[10:51:10 - Desktop]$ rm Bio294/file1.txt
[10:51:17 - Desktop]$ ls Bio294/
test1
[10:51:20 - Desktop]$ touch Bio294/file1.txt
[10:52:27 - Desktop]$ ls Bio294/
file1.txt      test1
[10:52:34 - Desktop]$
[10:52:36 - Desktop]$ mv Bio294/file1.txt Bio294/test1/
[10:52:46 - Desktop]$
[10:52:49 - Desktop]$ ls Bio294/test1/
file1.txt
[10:52:52 - Desktop]$
```

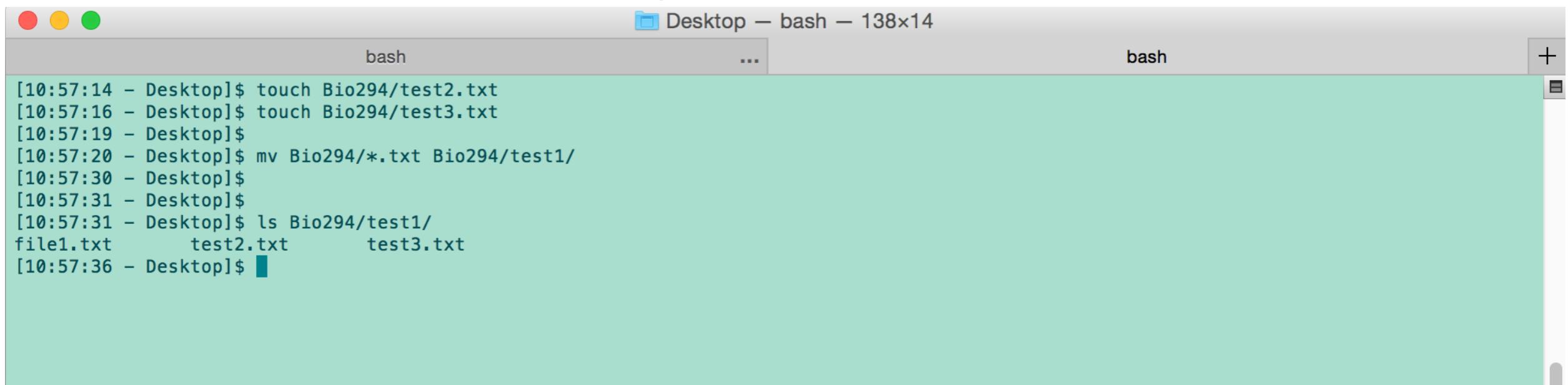
Create additional files...



A screenshot of a terminal window titled "Desktop – bash – 138x14". The window contains the following command history:

```
[10:49:49 - Desktop]$ touch Bio294/file1.txt
[10:50:10 - Desktop]$ ls Bio294/
file1.txt      test1
[10:50:14 - Desktop]$
```

... and move them all in one go



A screenshot of a terminal window titled "Desktop – bash – 138x14". The window contains the following command history:

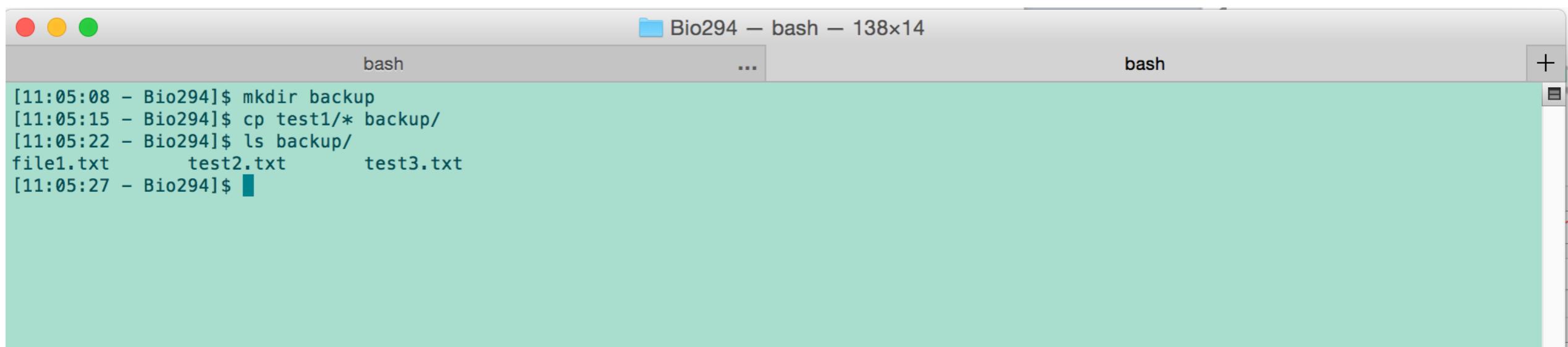
```
[10:57:14 - Desktop]$ touch Bio294/test2.txt
[10:57:16 - Desktop]$ touch Bio294/test3.txt
[10:57:19 - Desktop]$
[10:57:20 - Desktop]$ mv Bio294/*.txt Bio294/test1/
[10:57:30 - Desktop]$
[10:57:31 - Desktop]$
[10:57:31 - Desktop]$ ls Bio294/test1/
file1.txt      test2.txt      test3.txt
[10:57:36 - Desktop]$
```

What is the * doing?

Work safely: rm and rmdir and “dangerous” command line.

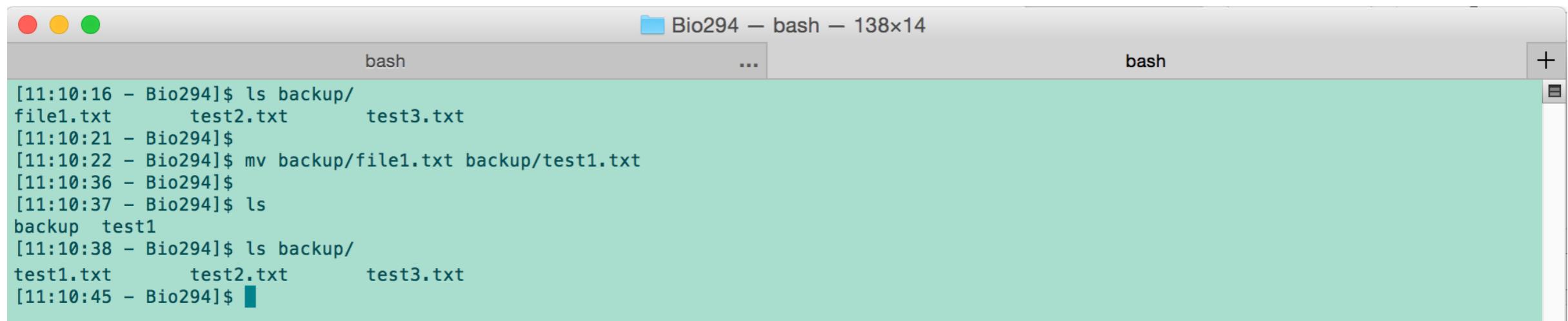
You can always save your raw data in an additional folder that you will never touch.

To copy your file to a new folder: cp



```
[11:05:08 - Bio294]$ mkdir backup
[11:05:15 - Bio294]$ cp test1/* backup/
[11:05:22 - Bio294]$ ls backup/
file1.txt      test2.txt      test3.txt
[11:05:27 - Bio294]$
```

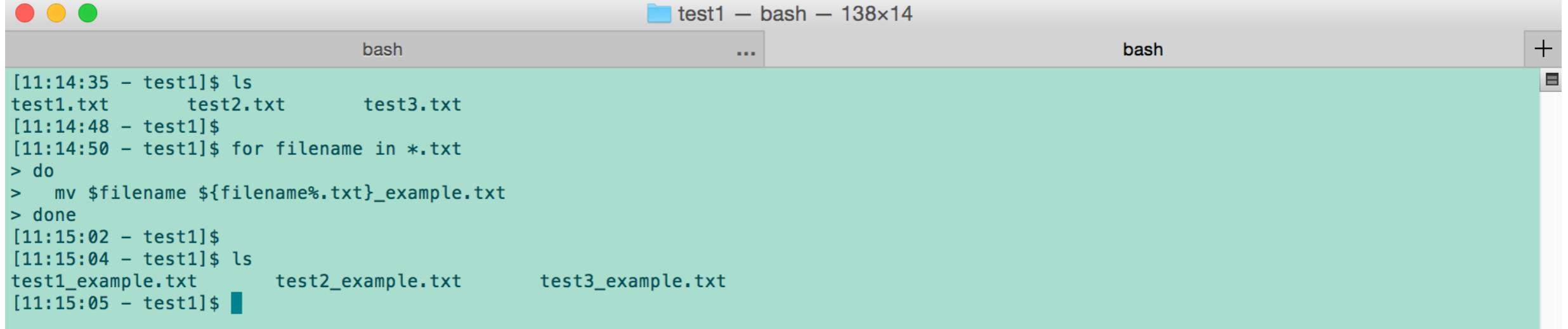
Rename files: mv



```
[11:10:16 - Bio294]$ ls backup/
file1.txt      test2.txt      test3.txt
[11:10:21 - Bio294]$
[11:10:22 - Bio294]$ mv backup/file1.txt backup/test1.txt
[11:10:36 - Bio294]$
[11:10:37 - Bio294]$ ls
backup  test1
[11:10:38 - Bio294]$ ls backup/
test1.txt      test2.txt      test3.txt
[11:10:45 - Bio294]$
```

Rename all files in folder:

```
for filename in *.txt
do
    mv $filename ${filename%.txt}_example.txt
done
```

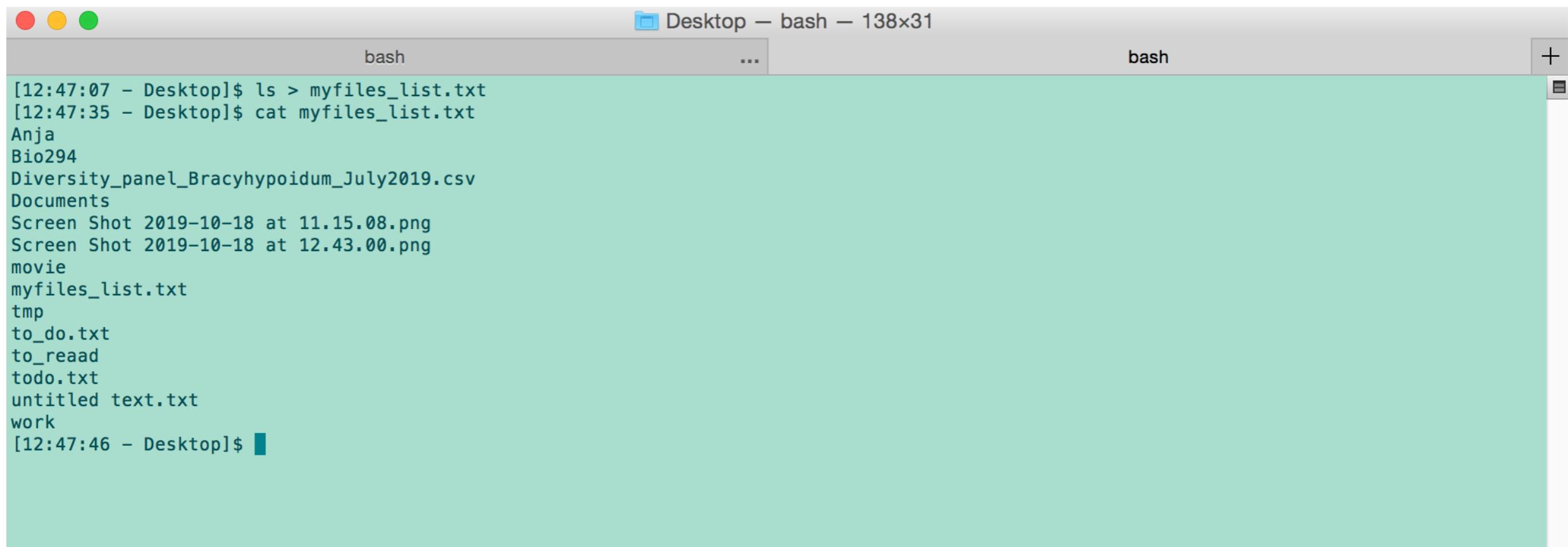


The screenshot shows a terminal window titled "test1 — bash — 138x14". The terminal content is as follows:

```
[11:14:35 - test1]$ ls
test1.txt      test2.txt      test3.txt
[11:14:48 - test1]$
[11:14:50 - test1]$ for filename in *.txt
> do
>     mv $filename ${filename%.txt}_example.txt
> done
[11:15:02 - test1]$
[11:15:04 - test1]$ ls
test1_example.txt      test2_example.txt      test3_example.txt
[11:15:05 - test1]$
```

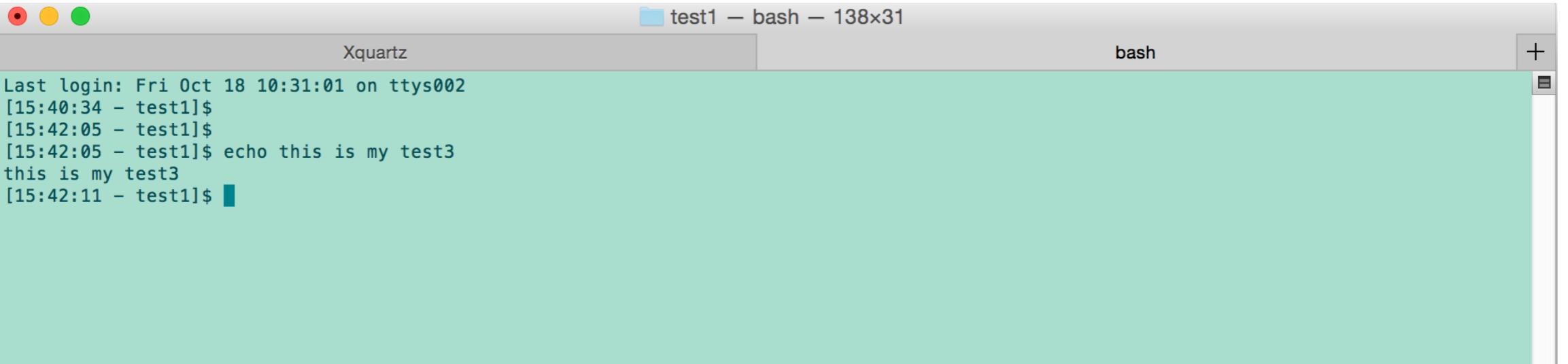
Can you explain how the loop is working?

Output redirection: >



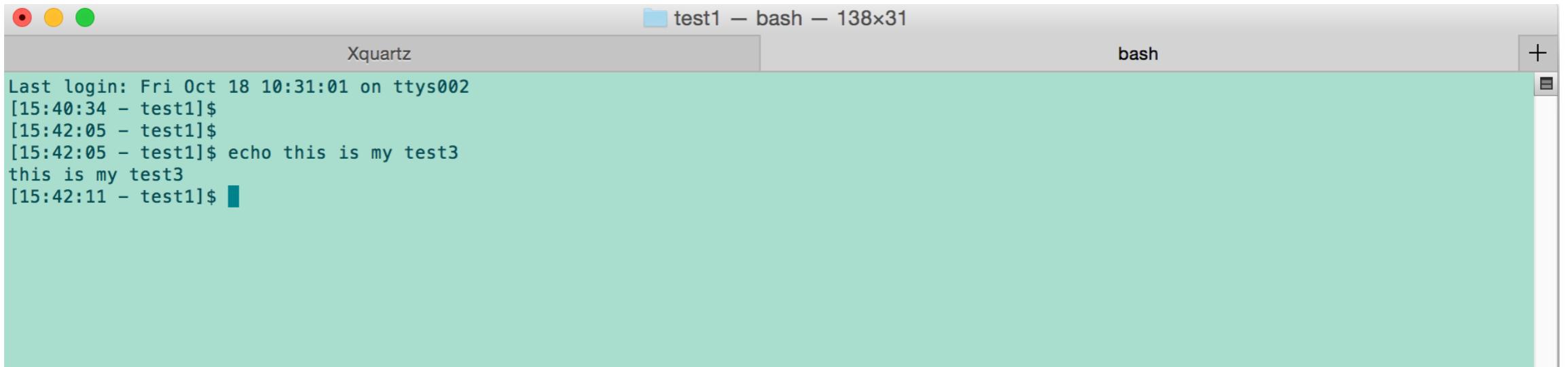
```
[12:47:07 - Desktop]$ ls > myfiles_list.txt
[12:47:35 - Desktop]$ cat myfiles_list.txt
Anja
Bio294
Diversity_panel_Bracyhypoidum_July2019.csv
Documents
Screen Shot 2019-10-18 at 11.15.08.png
Screen Shot 2019-10-18 at 12.43.00.png
movie
myfiles_list.txt
tmp
to_do.txt
to_reaad
todo.txt
untitled text.txt
work
[12:47:46 - Desktop]$
```

Writes arguments to standard output: echo



```
Last login: Fri Oct 18 10:31:01 on ttys002
[15:40:34 - test1]$
[15:42:05 - test1]$
[15:42:05 - test1]$ echo this is my test3
this is my test3
[15:42:11 - test1]$
```

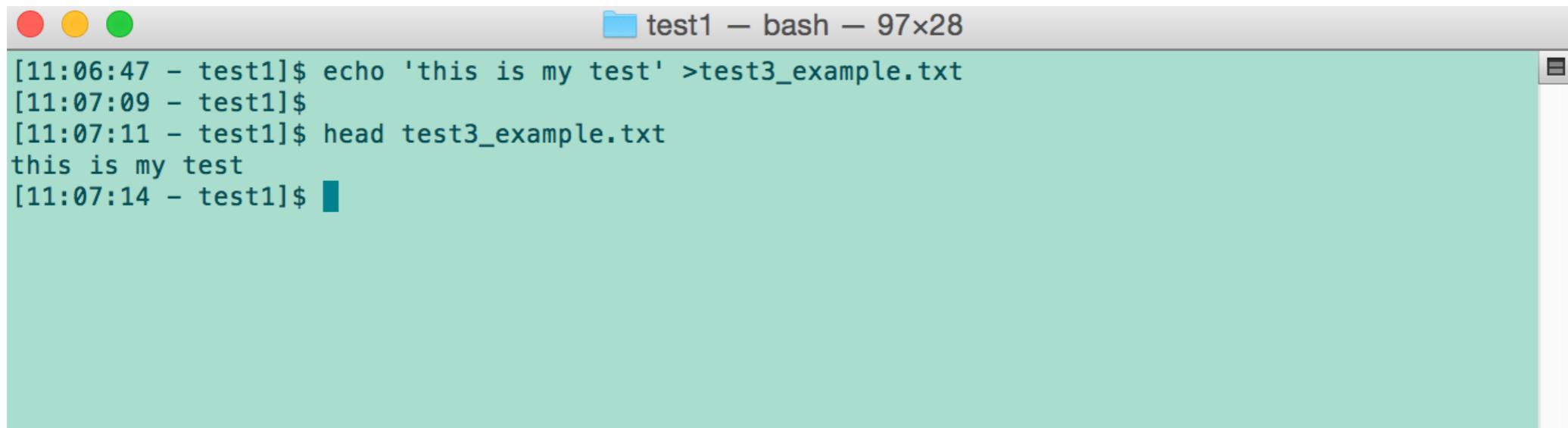
Writes arguments to standard output: echo



```
Last login: Fri Oct 18 10:31:01 on ttys002
[15:40:34 - test1]$
[15:42:05 - test1]$
[15:42:05 - test1]$ echo this is my test3
this is my test3
[15:42:11 - test1]$
```

How would you write “this is my test 3” in the file test3_example.txt?

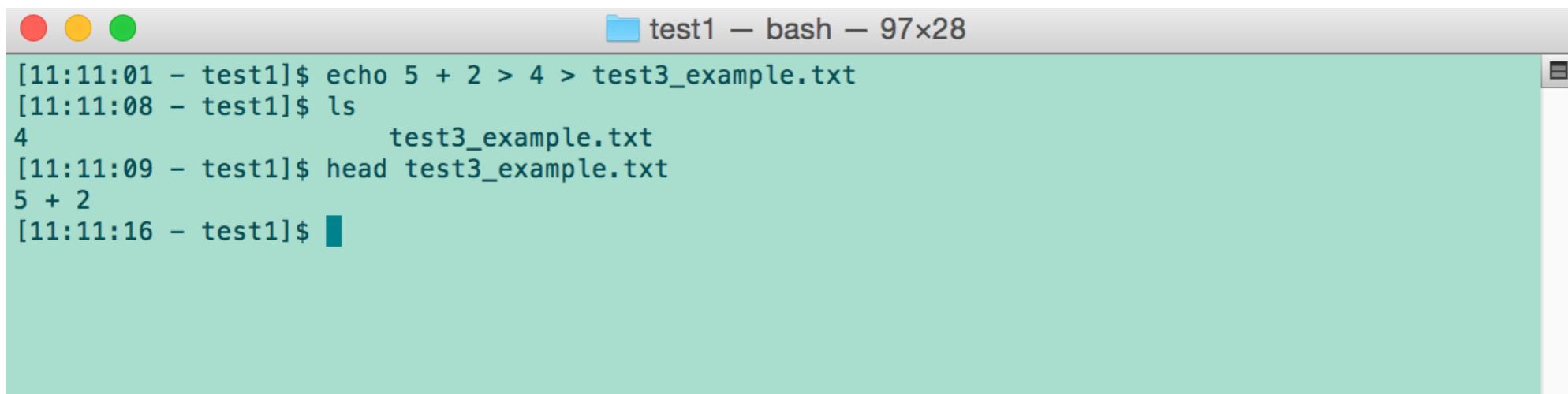
Writes arguments to standard output: echo



```
[11:06:47 - test1]$ echo 'this is my test' >test3_example.txt
[11:07:09 - test1]$
[11:07:11 - test1]$ head test3_example.txt
this is my test
[11:07:14 - test1]$ █
```

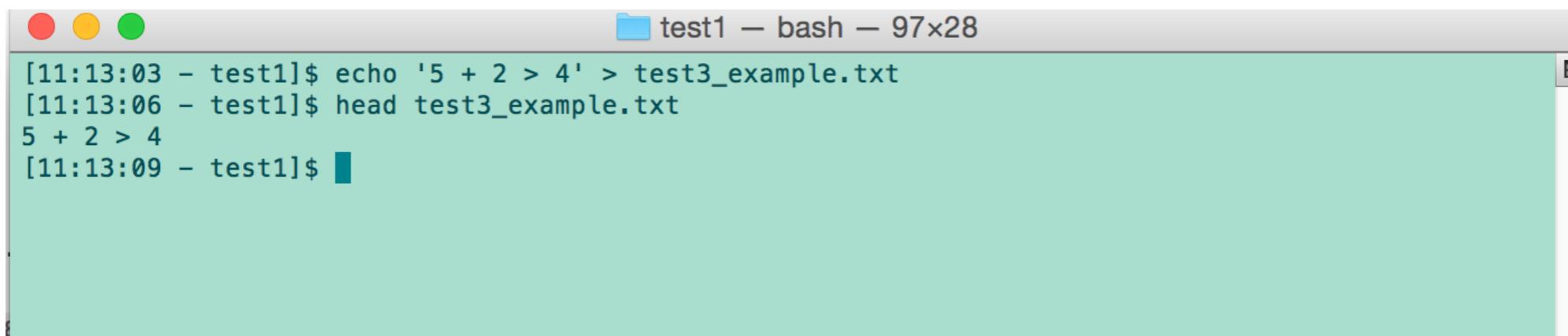
It is a good practice to use quotes to prevent the shell from interpreting a character.

for example, if you want to write $5 + 2 > 4$ to the file test3_example.txt



```
[11:11:01 - test1]$ echo 5 + 2 > 4 > test3_example.txt
[11:11:08 - test1]$ ls
4
        test3_example.txt
[11:11:09 - test1]$ head test3_example.txt
5 + 2
[11:11:16 - test1]$
```

Now, protect your test with “ ”

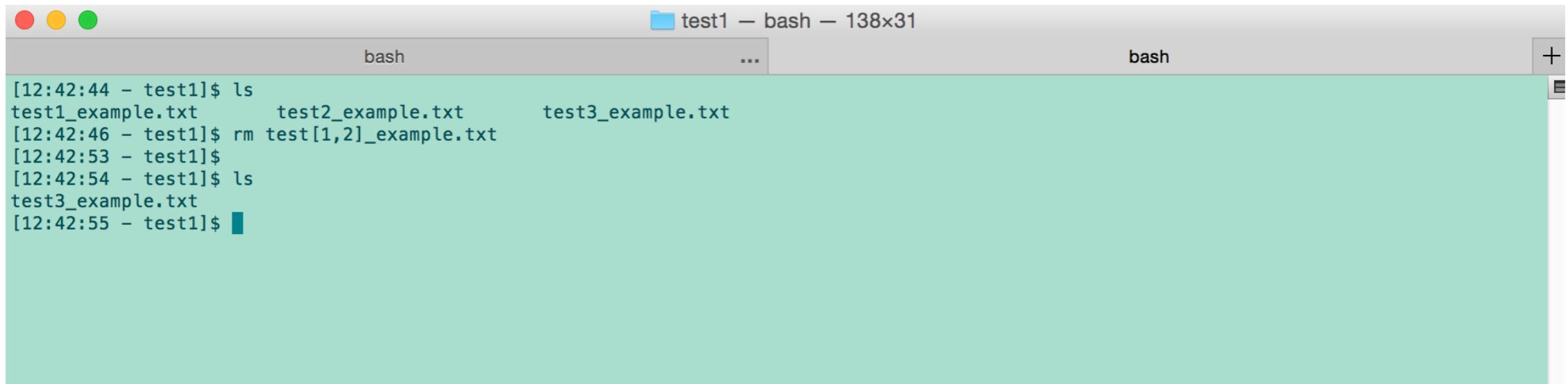


```
[11:13:03 - test1]$ echo '5 + 2 > 4' > test3_example.txt
[11:13:06 - test1]$ head test3_example.txt
5 + 2 > 4
[11:13:09 - test1]$
```

For more on quotes, see https://www.gnu.org/software/bash/manual/html_node/Quoting.html or <https://bash.cyberciti.biz/guide/Quoting>

Few other tips:

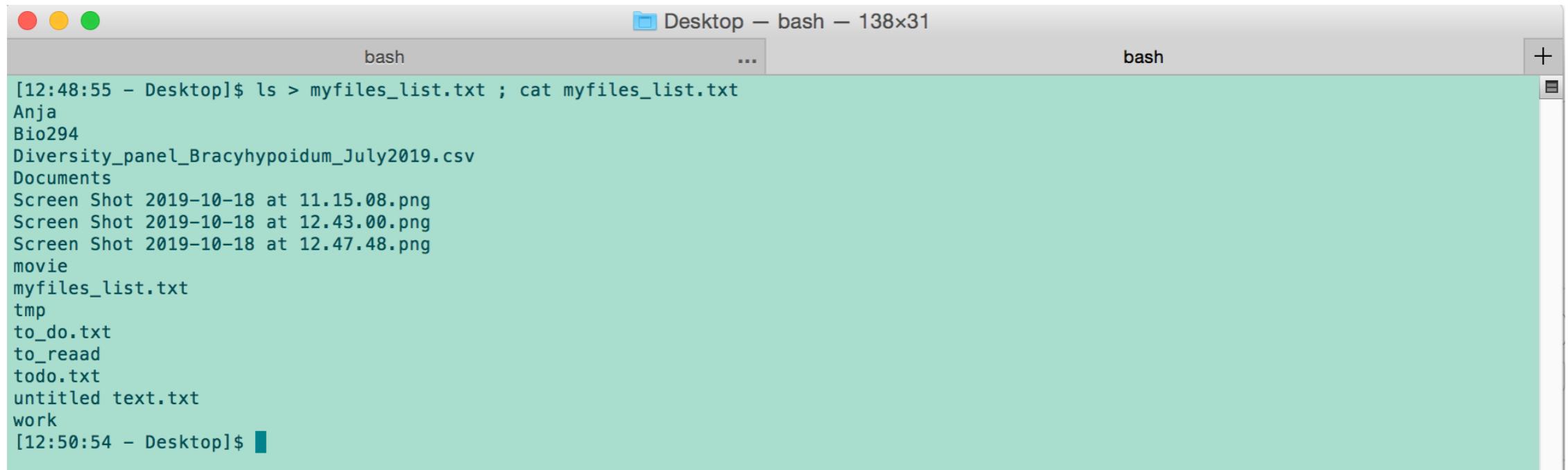
Select a pattern that contains specific characters with “ [] ”



```
[12:42:44 - test1]$ ls
test1_example.txt      test2_example.txt      test3_example.txt
[12:42:46 - test1]$ rm test[1,2]_example.txt
[12:42:53 - test1]$
[12:42:54 - test1]$ ls
test3_example.txt
[12:42:55 - test1]$
```

Few other tips:

Use “ ; ” as a command separator

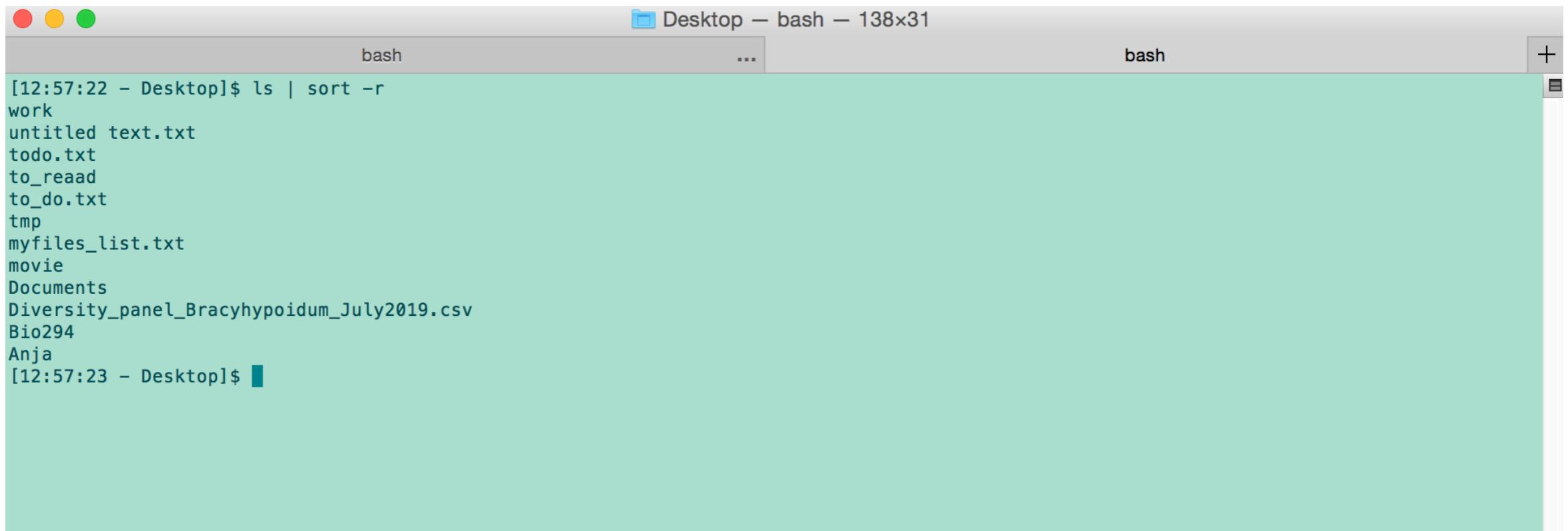


The screenshot shows a terminal window titled "Desktop – bash – 138x31". The window contains the following text:

```
[12:48:55 - Desktop]$ ls > myfiles_list.txt ; cat myfiles_list.txt
Anja
Bio294
Diversity_panel_Brachypodium_July2019.csv
Documents
Screen Shot 2019-10-18 at 11.15.08.png
Screen Shot 2019-10-18 at 12.43.00.png
Screen Shot 2019-10-18 at 12.47.48.png
movie
myfiles_list.txt
tmp
to_do.txt
to_reaad
todo.txt
untitled text.txt
work
[12:50:54 - Desktop]$
```

Few other tips:

“ | “ : A “pipe” chains commands together. It takes the output from one command and feeds it to the next as input



The screenshot shows a single terminal window titled "Desktop – bash – 138x31". The window contains the following command and its output:

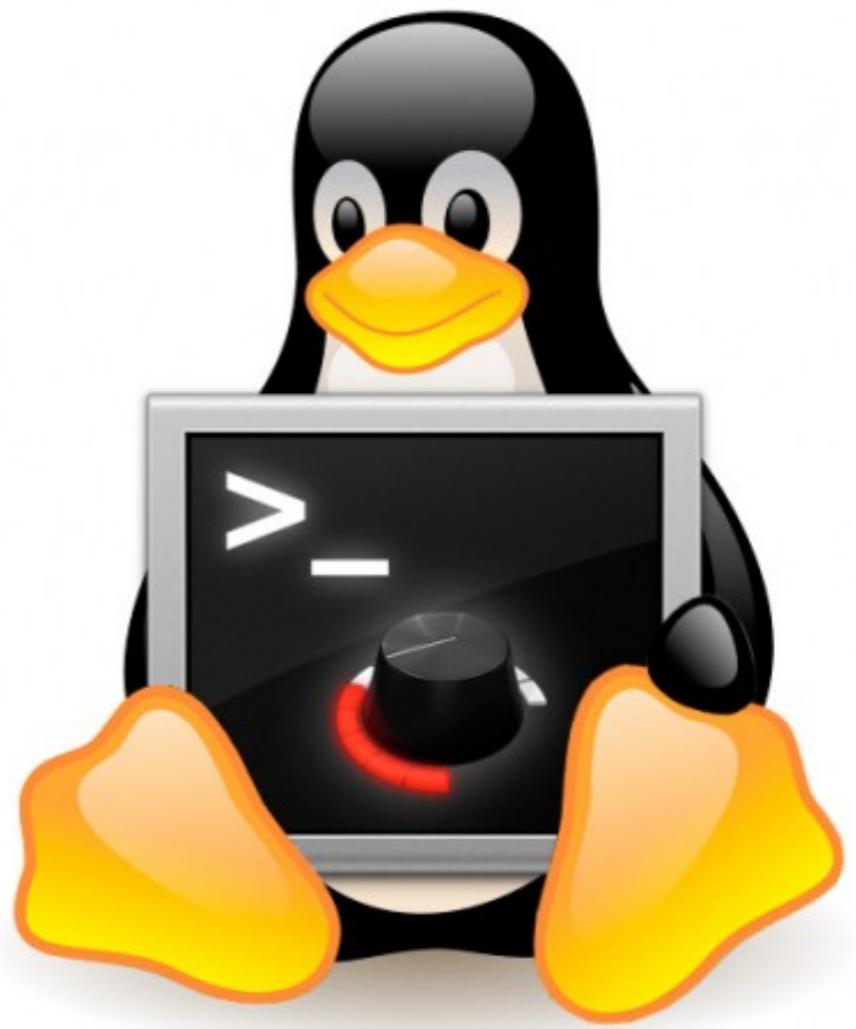
```
[12:57:22 - Desktop]$ ls | sort -r
work
untitled text.txt
todo.txt
to_reaad
to_do.txt
tmp
myfiles_list.txt
movie
Documents
Diversity_panel_Bracyhypoidum_July2019.csv
Bio294
Anja
[12:57:23 - Desktop]$
```

The command `ls` lists all files in the current directory. The pipe operator `|` is used to feed the output of `ls` into the `sort -r` command, which sorts the files in reverse alphabetical order.

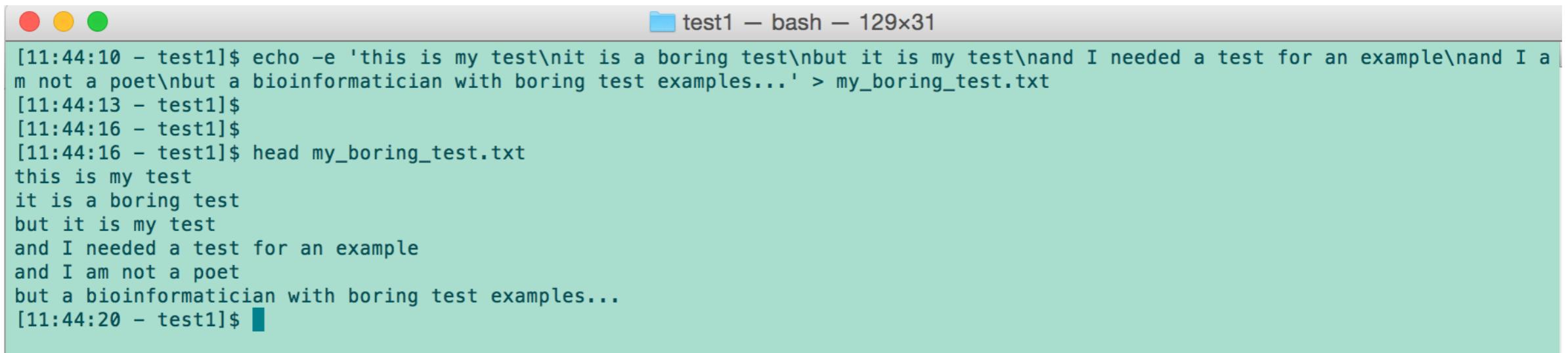
Few other tips:

ctrl + A : bring the cursor at the beginning of your command
ctrl + E : bring the cursor at the end of your command

How to work with files ?



Let's create a file containing the following text: echo -e 'this is my test\nit is a boring test\nbut it is my test\nand I needed a test for an example\nand I am not a poet\nbut a bioinformatician with boring test examples...' > my_boring_test.txt



```
[11:44:10 - test1]$ echo -e 'this is my test\nit is a boring test\nbut it is my test\nand I needed a test for an example\nand I am not a poet\nbut a bioinformatician with boring test examples...' > my_boring_test.txt
[11:44:13 - test1]$
[11:44:16 - test1]$
[11:44:16 - test1]$ head my_boring_test.txt
this is my test
it is a boring test
but it is my test
and I needed a test for an example
and I am not a poet
but a bioinformatician with boring test examples...
[11:44:20 - test1]$ █
```

- \n = new line
- The ‘-e’ option in Linux acts as interpretation of escaped characters that are backslashed.



Cat

Cat can be used for the following purposes:

- **Display text files on screen.**
- Copy text files.
- Combine text files.

The screenshot shows a terminal window with three colored tabs (red, yellow, green) at the top. The title bar reads "test1 – bash – 129x31". The terminal content displays the following text:

```
[11:47:45 - test1]$ cat my_boring_test.txt
this is my test
it is a boring test
but it is my test
and I needed a test for an example
and I am not a poet
but a bioinformatician with boring test examples...
[11:47:52 - test1]$
[11:47:55 - test1]$
[11:47:56 - test1]$ █
```

NB: compared to “head”, cat display the entire content of the file



Cat

Cat can be used for the following purposes:

- Display text files on screen.
- **Copy text files.**
- Combine text files.

The screenshot shows a terminal window with the title bar "test1 – bash – 129x31". The window contains the following text:

```
[11:50:25 - test1]$ cat my_boring_test.txt > my_boring_test2.txt
[11:50:43 - test1]$
[11:50:44 - test1]$ ls
my_boring_test.txt      my_boring_test2.txt      test3_example.txt
[11:50:45 - test1]$
[11:50:51 - test1]$ head my_boring_test2.txt
this is my test
it is a boring test
but it is my test
and I needed a test for an example
and I am not a poet
but a bioinformatician with boring test examples...
[11:50:55 - test1]$ █
```



Cat

Cat can be used for the following purposes:

- Display text files on screen.
- Copy text files.
- **Combine text files.**

The screenshot shows a terminal window with the title bar "test1 – bash – 129x31". The window contains the following text:

```
[11:51:41 - test1]$ cat my_boring_test.txt test3_example.txt
this is my test
it is a boring test
but it is my test
and I needed a test for an example
and I am not a poet
but a bioinformatician with boring test examples...
5 + 2 > 4
[11:51:50 - test1]$
```



Sort

Can be used to sort files alphabetically or numerically

```
[11:59:55 - test1]$ sort my_boring_test.txt
and I am not a poet
and I needed a test for an example
but a bioinformatician with boring test examples...
but it is my test
it is a boring test
this is my test
[11:59:58 - test1]$
[12:00:00 - test1]$
[12:00:00 - test1]$ sort -r my_boring_test.txt
this is my test
it is a boring test
but it is my test
but a bioinformatician with boring test examples...
and I needed a test for an example
and I am not a poet
[12:00:48 - test1]$
[12:00:51 - test1]$ █
```

For more option on sort:

<https://www.softwaretestinghelp.com/unix-sort-command/>



SED

- SED is a powerful text stream editor. Can do insertion, deletion, search and replace(substitution).
- SED command in unix supports regular expression which allows it perform complex pattern matching.



SED

Modify our previous text: echo -e 'this is my test\nit is a boring test\nbut it is my test and I needed a test for an example\nand I am not a poet\nbut a bioinformatician with boring test examples...' > my_boring_test.txt



SED

Replacing (substituting) string: sed s

```
[12:13:05 - test1]$ sed 's/test/Test/' my_boring_test.txt
this is my Test
it is a boring Test
but it is my Test and I needed a test for an example
and I am not a poet
but a bioinformatician with boring Test examples...
[12:13:10 - test1]$
```

sed -s: replace the first occurrence in each line



Little exercises

Can you explain these commands?

```
sed 's/test/Test/g' my_boring_test.txt
```

```
sed '2 s/test/Test/' my_boring_test.txt
```

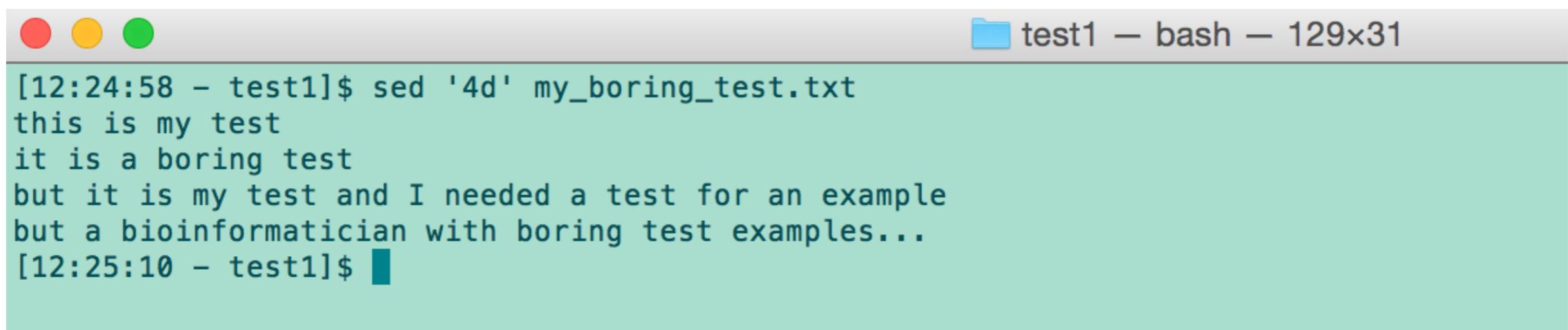
```
sed '2,4 s/test/Test/' my_boring_test.txt
```

```
sed 's/test/Test/2' my_boring_test.txt
```



SED

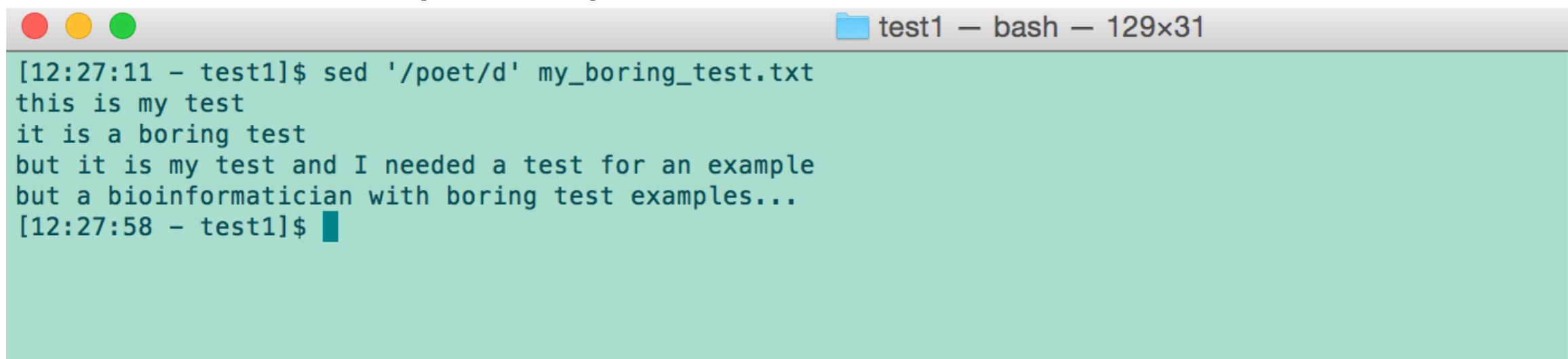
Delete the nth line: d



A terminal window titled "test1 – bash – 129x31". The window shows a command being run: [12:24:58 – test1]\$ sed '4d' my_boring_test.txt. The output of the command is displayed below the command line, showing all lines except the 4th one.

```
[12:24:58 - test1]$ sed '4d' my_boring_test.txt
this is my test
it is a boring test
but it is my test and I needed a test for an example
but a bioinformatician with boring test examples...
[12:25:10 - test1]$
```

Or a line with a specific pattern



A terminal window titled "test1 – bash – 129x31". The window shows a command being run: [12:27:11 – test1]\$ sed '/poet/d' my_boring_test.txt. The output of the command is displayed below the command line, showing all lines except those containing the word "poet".

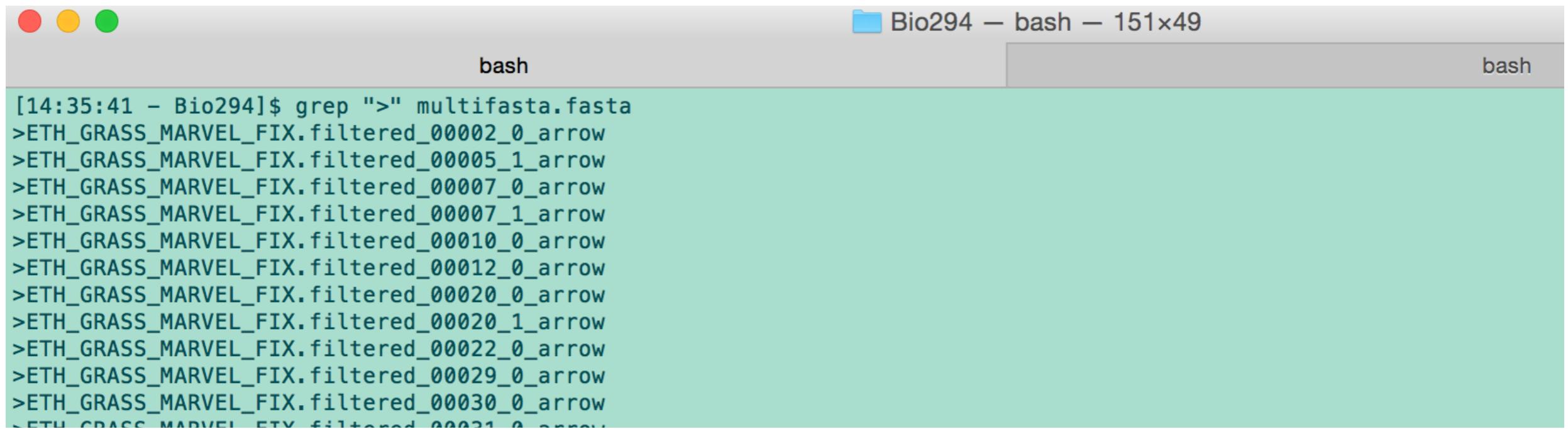
```
[12:27:11 - test1]$ sed '/poet/d' my_boring_test.txt
this is my test
it is a boring test
but it is my test and I needed a test for an example
but a bioinformatician with boring test examples...
[12:27:58 - test1]$
```



grep

grep is a command-line utility for searching plain-text data sets for **lines** that match a regular expression.

grep [*OPTION...*] *PATTERNS* [*FILE...*]



A screenshot of a terminal window titled "Bio294 – bash – 151x49". The window shows the command "grep > multifasta.fasta" being run, followed by a list of sequence identifiers starting with ">ETH_GRASS_MARVEL_FIX.filtered_00002_0_arrow". The terminal has three colored status indicators (red, yellow, green) in the top-left corner.

```
[14:35:41 - Bio294]$ grep ">" multifasta.fasta
>ETH_GRASS_MARVEL_FIX.filtered_00002_0_arrow
>ETH_GRASS_MARVEL_FIX.filtered_00005_1_arrow
>ETH_GRASS_MARVEL_FIX.filtered_00007_0_arrow
>ETH_GRASS_MARVEL_FIX.filtered_00007_1_arrow
>ETH_GRASS_MARVEL_FIX.filtered_00010_0_arrow
>ETH_GRASS_MARVEL_FIX.filtered_00012_0_arrow
>ETH_GRASS_MARVEL_FIX.filtered_00020_0_arrow
>ETH_GRASS_MARVEL_FIX.filtered_00020_1_arrow
>ETH_GRASS_MARVEL_FIX.filtered_00022_0_arrow
>ETH_GRASS_MARVEL_FIX.filtered_00029_0_arrow
>ETH_GRASS_MARVEL_FIX.filtered_00030_0_arrow
>ETH_GRASS_MARVEL_FIX.filtered_00031_0_arrow
```



Little exercises

You will find more information about the different options available at: <http://man7.org/linux/man-pages/man1/grep.1.html>

How many lines contains the pattern “ATCG”

How many lines contains the pattern “atcg”

How many lines contains the pattern “ATCG” or “atcg”

Write all the headers of the fasta sequences in a new file



Answers

You will find more information about the different options available at: <http://man7.org/linux/man-pages/man1/grep.1.html>

```
grep -c "ATCG" multifasta.fasta
```

```
grep -c "atcg" multifasta.fasta
```

```
grep -ci "atcg" multifasta.fasta
```

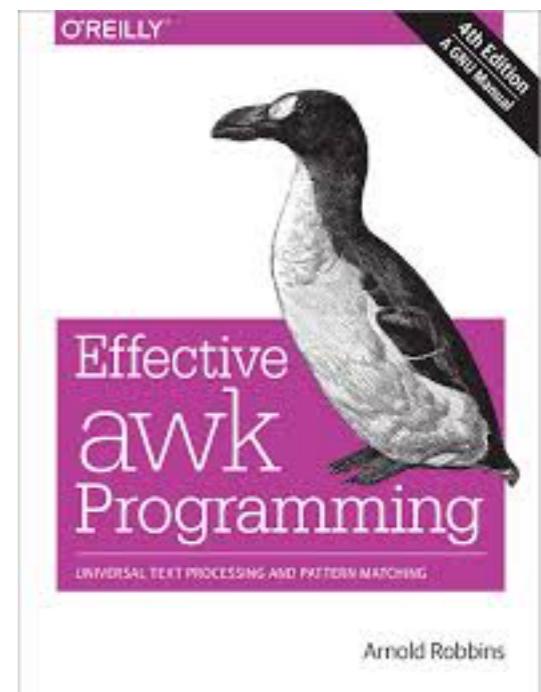
```
grep ">" multifasta.fasta > header.txt
```



awk

AWK is a domain-specific language designed for text processing and typically used as a data extraction and reporting tool. It is a standard feature of most Unix-like operating systems.

pattern { action }





awk

```
awk '{print $1,$2,$3}' Supp_Table2.csv
```

```
awk '{print $1,$2,"data_set1"}' Supp_Table2.csv
```

```
awk '{if ($4 > 6) {print $0}}' Supp_Table2.csv
```

Operator	Meaning
==	Is equal
!=	Is not equal to
>	Is greater than
>=	Is greater than or equal to
<	Is less than
<=	Is less than or equal to



awk

```
awk '$1 ~ "Bd1" {print $0}' Supp_Table2.csv
```

Operator	Meaning
<code>~</code>	Matches
<code>!~</code>	Doesn't match



Little exercises

- How many uniq “term” are present in Supp_Table2.csv?
- Keep only the lines for which the count is > 5 AND the Pvalue < 0.05, and sort the table according to descending Pvalues
- add a column specifying whether the Pvalue is significant (< 0.05) or not_significant



Answers

```
awk '{print $8}' Supp_Table2.csv|sort -u|wc -l
```

```
awk '{if ($5 > 5 && $9 <0.05) {print $0}}' Supp_Table2.csv| sort -rk9,9
```

```
awk '{if ($9 > 0.005) {print $0, "significant"} else {print $0,  
"not_significant"}}' Supp_Table2.csv
```



University of
Zurich ^{UZH}

emboss

<http://manuals.bioinformatics.ucr.edu/home/emboss>



First step

We will use the file multifasta.fasta

Rename each header so that the term “eth_grass_marvel_fix.filtered_” is replaced by “scaffold”, and write the sequences in a new file (eg. multifasta_renamed.fasta)

```
sed 's/ETH_GRASS_MARVEL_FIX.filtered_/scaffold/g' multifasta.fasta| sed 's/_0_arrow//g'|cat > multifasta_renamed.fasta
```



Infoseq

```
infoseq multifasta_renamed.fasta
```

```
infoseq multifasta_renamed.fasta -only -name -length
```

```
infoseq multifasta_renamed.fasta:scaffold*
```

Create a table that contains only the name of each sequences as well as their length and GC content.



Seqret

Performs sequence retrieval from databases, feature parsing, sequence and alignment reformatting.

```
seqret multifasta.fasta -sbegin 15 -send 50 -out seqret_test.fasta
```

```
seqretpsplit multifasta.fasta
```

Extract the sequence of scaffold00002_0_arrow

Extract the sequence of the scaffold between 200 and 300, and write them in a new fasta file. Use only 1 command line



Answers

Extract the sequence of scaffold00002_0_arrow

```
seqret multifasta_renamed.fasta:scaffold00002_0_arrow
```

Extract the sequence of the scaffold between 200 and 300, and write them in a new fasta file. Use only 1 command line

1) create a list: grep "d002" multifasta_renamed.fasta|sed 's/>/multifasta_renamed.fasta:/g' >my_list|

2) extract from list directly: seqret @my_list



Extractseq

Find out how to use extractseq...

And extract the first 1000 nt from the scaffold of your choice



Revseq

Find out how to use revseq...

And do a dot plot of a sequence against its reverse complement