

Introduction to Terminal

Computing in Optimization and Statistics: Lecture 1
Jackie Baek

MIT

January 9, 2018

Overview

Introduction & Motivation

Navigation commands

Files

- File path shortcuts

- Basic file commands

- Hidden Files

- `.bashrc` / `.bash_profile`

Redirection

SSH

Simple Pattern Matching

How bash works

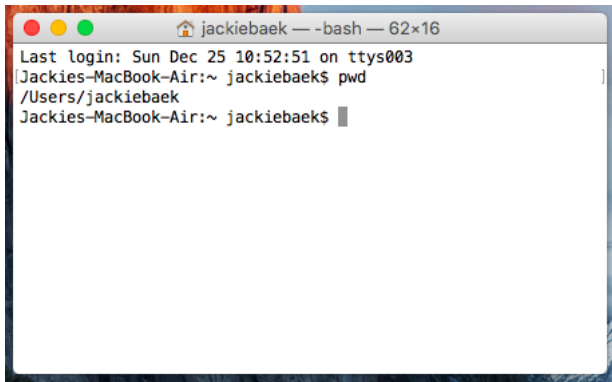
- Environment Variables

Key Takeaways

What is the terminal?



What is the terminal?



```
jackiebaek — -bash — 62x16
Last login: Sun Dec 25 10:52:51 on ttys003
[Jackies-MacBook-Air:~ jackiebaek$ pwd
/Users/jackiebaek
Jackies-MacBook-Air:~ jackiebaek$
```

- ▶ The terminal is a text-based interface to interact with the computer.
- ▶ Alternate names: console, shell, command line, command prompt

Example

- ▶ Say you want to delete all files in a directory that end with .pyc

```
$ rm *.pyc
```

- ▶ This is possible to do without the terminal, but it requires much more effort.

Why should I learn it?

- ▶ You can do almost everything using just the terminal.
- ▶ It can do many tasks faster than using a graphic interface.
- ▶ It is sometimes the only option (e.g. accessing a client's server using SSH).
- ▶ It is universal.

Use case: running code

- ▶ Without the terminal, you need to install an IDE (Integrated development environment) for every programming language.
 - ▶ e.g. RStudio for R

Use case: running code

- ▶ Without the terminal, you need to install an IDE (Integrated development environment) for every programming language.
 - ▶ e.g. RStudio for R
- ▶ Useful to use the terminal instead of IDEs when:

Use case: running code

- ▶ Without the terminal, you need to install an IDE (Integrated development environment) for every programming language.
 - ▶ e.g. RStudio for R
- ▶ Useful to use the terminal instead of IDEs when:
 - ▶ You use more than one programming language.

```
$ python process_stuff.py
```

```
$ R make_plots.R
```

Use case: running code

- ▶ Without the terminal, you need to install an IDE (Integrated development environment) for every programming language.
 - ▶ e.g. RStudio for R
- ▶ Useful to use the terminal instead of IDEs when:
 - ▶ You use more than one programming language.

```
$ python process_stuff.py
```

```
$ R make_plots.R
```

- ▶ You want to chain commands together.

```
$ python process_stuff.py && R make_plots.R
```

Use case: running code

- ▶ Without the terminal, you need to install an IDE (Integrated development environment) for every programming language.
 - ▶ e.g. RStudio for R
- ▶ Useful to use the terminal instead of IDEs when:
 - ▶ You use more than one programming language.

```
$ python process_stuff.py
```

```
$ R make_plots.R
```

- ▶ You want to chain commands together.

```
$ python process_stuff.py && R make_plots.R
```

- ▶ You want to run a program with different parameters and different output files.

```
$ python process_stuff.py 2015 100 > output_100.csv
```

Terminal Basics

- ▶ We will be using a **shell** called **bash**: a program that interprets and processes the commands you input into the terminal.
- ▶ The shell is always in a **working directory**.
- ▶ A typical command looks like:

```
$ command <argument1> <argument2> ...
```

Basic navigation commands

pwd: prints working directory.

```
$ pwd
```

```
/Users/jackiebaek
```

Basic navigation commands

pwd: prints working directory.

```
$ pwd  
/Users/jackiebaek
```

ls: lists directory contents.

```
$ ls  
Applications      Movies  
Desktop           Music  
Documents         Pictures
```

Basic navigation commands

pwd: prints working directory.

```
$ pwd  
/Users/jackiebaek
```

ls: lists directory contents.

```
$ ls  
Applications                Movies  
Desktop                     Music  
Documents                   Pictures
```

cd <**directory**>: change working directory to new directory.

```
$ cd Documents  
$ pwd  
/Users/jackiebaek/Documents
```

Tab and arrow keys are your friends

- ▶ Use **tab** to autocomplete *commands* and *file paths*.
- ▶ Use ↑ and ↓ arrow keys to navigate through your command history.

What is a file?

A file is a container of data (0's and 1's).

What is a file?

A file is a container of data (0's and 1's).

A file name usually has an **extension** (e.g. .pdf, .doc, .csv), but these are just conventions.

What is a file?

A file is a container of data (0's and 1's).

A file name usually has an **extension** (e.g. .pdf, .doc, .csv), but these are just conventions.

A file is contained in a **directory** (folder). Files within the same directory have unique names.

What is a file?

A file is a container of data (0's and 1's).

A file name usually has an **extension** (e.g. .pdf, .doc, .csv), but these are just conventions.

A file is contained in a **directory** (folder). Files within the same directory have unique names.

Every file and directory has a unique location in the file system, called a **path**.

- ▶ **Absolute path:**

- /Users/jackiebaek/Dropbox/Documents/hello.txt*

- ▶ **Relative path** (if my current working directory is */Users/jackiebaek/Dropbox*): *Documents/hello.txt*

File path shortcuts

. is current directory.

.. is parent directory.

- ▶ ../*file.txt* references a file named *file.txt* in the parent directory.

File path shortcuts

`.` is current directory.

`..` is parent directory.

- ▶ `../file.txt` references a file named `file.txt` in the parent directory.

`~` is home.

- ▶ expands to `/Users/<username>` (or wherever *home* is on that machine).
- ▶ `~/Documents` → `/Users/jackiebaek/Documents`
- ▶ The command **cd** (without any arguments) takes you to `~`.

Working with files

Working with files

mkdir *directory_name*: create a new directory.

```
$ mkdir new_directory
```


Working with files

mkdir *directory_name*: create a new directory.

```
$ mkdir new_directory
```

touch *file*: create an empty file.

rm *file*: delete a file (**Careful!** Can't be undone!)

```
$ touch brand_new_file.txt
```

```
$ rm brand_new_file.txt
```

Working with files

mkdir *directory_name*: create a new directory.

```
$ mkdir new_directory
```

touch *file*: create an empty file.

rm *file*: delete a file (**Careful!** Can't be undone!)

```
$ touch brand_new_file.txt
```

```
$ rm brand_new_file.txt
```

nano *file*: edit contents of a file (many other editors exist).

```
$ nano helloworld.txt
```

Working with files

mkdir *directory_name*: create a new directory.

```
$ mkdir new_directory
```

touch *file*: create an empty file.

rm *file*: delete a file (**Careful!** Can't be undone!)

```
$ touch brand_new_file.txt
```

```
$ rm brand_new_file.txt
```

nano *file*: edit contents of a file (many other editors exist).

```
$ nano helloworld.txt
```

cat *file*: prints contents of a file.

```
$ cat helloworld.txt
```

```
Hello, World!
```

Working with files

mkdir *directory_name*: create a new directory.

```
$ mkdir new_directory
```

touch *file*: create an empty file.

rm *file*: delete a file (**Careful!** Can't be undone!)

```
$ touch brand_new_file.txt
```

```
$ rm brand_new_file.txt
```

nano *file*: edit contents of a file (many other editors exist).

```
$ nano helloworld.txt
```

cat *file*: prints contents of a file.

```
$ cat helloworld.txt
```

```
Hello, World!
```

cp *source target*: copy.

mv *source target*: move/rename.

```
$ cp helloworld.txt helloworld_copy.txt
```

```
$ mv helloworld.txt goodbyeworld.txt
```

Hidden Files

- ▶ Files that start with a dot (.) are called **hidden** files.
- ▶ Used for storing preferences, config, settings.
- ▶ Use `ls -a` to list all files.

```
$ ls
```

```
github_notes.md  presentation  scripts
```

```
$ ls -a
```

```
.                .git          github_notes.md  scripts
..               .gitignore    presentation
```

.bashrc / .bash_profile

- ▶ There is a hidden file in `~` directory called `.bashrc` or `.bash_profile`.
- ▶ This file is a bash script that runs at the beginning of each session (i.e. when you open the terminal).

.bashrc / .bash_profile

- ▶ There is a hidden file in `~` directory called `.bashrc` or `.bash_profile`.
- ▶ This file is a bash script that runs at the beginning of each session (i.e. when you open the terminal).
- ▶ This file can be used to set variables or to declare **aliases**.
- ▶ **alias** *new_command=command*

```
$ alias athena="ssh baek@athena.dialup.mit.edu"
```

Redirection

> redirects output to a file, *overwriting* if file already exists.

```
$ ls > out.txt
```

>> redirects output to a file, *appending* if file already exists.

```
$ python fetch_data.py >> output.csv
```


Redirection

> redirects output to a file, *overwriting* if file already exists.

```
$ ls > out.txt
```

>> redirects output to a file, *appending* if file already exists.

```
$ python fetch_data.py >> output.csv
```

< uses contents of file as STDIN (standard input) to the command.

```
$ python process_stuff.py < input.txt
```

Secure Shell (SSH)

- ▶ Sometimes we need to work on a remote machine.
 - ▶ We need more computing power than just our local machine.
 - ▶ We need to access data from a client's server.
- ▶ Can use SSH to securely access the terminal for the remote machine.

Secure Shell (SSH)

- ▶ Sometimes we need to work on a remote machine.
 - ▶ We need more computing power than just our local machine.
 - ▶ We need to access data from a client's server.
- ▶ Can use SSH to securely access the terminal for the remote machine.

```
$ ssh baek@athena.dialup.mit.edu
```

Secure Shell (SSH)

- ▶ Sometimes we need to work on a remote machine.
 - ▶ We need more computing power than just our local machine.
 - ▶ We need to access data from a client's server.
- ▶ Can use SSH to securely access the terminal for the remote machine.

```
$ ssh baek@athena.dialup.mit.edu
```

```
Password:
```

Secure Shell (SSH)

- ▶ Sometimes we need to work on a remote machine.
 - ▶ We need more computing power than just our local machine.
 - ▶ We need to access data from a client's server.
- ▶ Can use SSH to securely access the terminal for the remote machine.

```
$ ssh baek@athena.dialup.mit.edu
```

```
Password:
```

```
Welcome to Ubuntu 14.04.5 LTS
```

```
...
```

```
Last login: Tue Aug 30 10:11:49 2016 from howe-and-ser-...
```

```
baek@howe-and-ser-moving:~$
```

Use *logout* to exit SSH session.

Secure Copy (scp)

Can transfer files between local and remote machines using the **scp** command on your local machine.

Move *my_file.txt* from local machine to remote home directory.

```
$ scp my_file.txt baek@athena.dialup.mit.edu:~
```

Move *remote_file.txt* from remote to local machine.

```
$ scp baek@athena.dialup.mit.edu:~/remote_file.txt .
```

Simple Pattern Matching (Globbing)

- ▶ Match [multiple] filenames with wildcard characters.
- ▶ Similar to *regular expressions*, but slightly different syntax.

Simple Pattern Matching (Globbing)

- ▶ Match [multiple] filenames with wildcard characters.
- ▶ Similar to *regular expressions*, but slightly different syntax.

Example:

```
$ ls
```

```
a1.txt    a2.pdf    apple.txt  bar.pdf
```


Simple Pattern Matching (Globbing)

- ▶ Match [multiple] filenames with wildcard characters.
- ▶ Similar to *regular expressions*, but slightly different syntax.

Example:

```
$ ls
```

```
a1.txt      a2.pdf      apple.txt   bar.pdf
```

```
$ ls a*
```

```
a1.txt      a2.pdf      apple.txt
```

Simple Pattern Matching (Globbing)

- ▶ Match [multiple] filenames with wildcard characters.
- ▶ Similar to *regular expressions*, but slightly different syntax.

Example:

```
$ ls
```

```
a1.txt      a2.pdf      apple.txt    bar.pdf
```

```
$ ls a*
```

```
a1.txt      a2.pdf      apple.txt
```

```
$ ls a[0-9]*
```

```
a1.txt      a2.pdf
```

Simple Pattern Matching (Globbing)

- ▶ Match [multiple] filenames with wildcard characters.
- ▶ Similar to *regular expressions*, but slightly different syntax.

Example:

```
$ ls
```

```
a1.txt      a2.pdf      apple.txt    bar.pdf
```

```
$ ls a*
```

```
a1.txt      a2.pdf      apple.txt
```

```
$ ls a[0-9]*
```

```
a1.txt      a2.pdf
```

```
$ ls *.pdf
```

```
a2.pdf      bar.pdf
```

Simple Pattern Matching (Globbing)

Wildcard	Description	Example	Matches
*	matches any number of any characters including none	Law*	Law , Laws , or Lawyer
		Law	Law , GrokLaw , or Lawyer .
?	matches any single character	?at	Cat , cat , Bat or bat
[abc]	matches one character given in the bracket	[CB]at	Cat or Bat
[a-z]	matches one character from the range given in the bracket	Letter[0-9]	Letter0 , Letter1 , Letter2 up to Letter9

Figure: Source: Wikipedia

Simple Pattern Matching (Globbing)

Wildcard	Description	Example	Matches
*	matches any number of any characters including none	Law*	Law , Laws , or Lawyer
		Law	Law , GrokLaw , or Lawyer .
?	matches any single character	?at	Cat , cat , Bat or bat
[abc]	matches one character given in the bracket	[CB]at	Cat or Bat
[a-z]	matches one character from the range given in the bracket	Letter[0-9]	Letter0 , Letter1 , Letter2 up to Letter9

Figure: Source: Wikipedia

Remove all files that end with .pyc

```
$ rm *.pyc
```

Simple Pattern Matching (Globbing)

Wildcard	Description	Example	Matches
*	matches any number of any characters including none	Law*	Law , Laws , or Lawyer
		Law	Law , GrokLaw , or Lawyer .
?	matches any single character	?at	Cat , cat , Bat or bat
[abc]	matches one character given in the bracket	[CB]at	Cat or Bat
[a-z]	matches one character from the range given in the bracket	Letter[0-9]	Letter0 , Letter1 , Letter2 up to Letter9

Figure: Source: Wikipedia

Remove all files that end with .pyc

```
$ rm *.pyc
```

Copy all files that has "dog" in its name to the *animal/* directory.

```
$ cp *dog* animal/
```

How bash works

How bash works

- ▶ Bash is a programming language.
 - ▶ Can set variables, use for loops, if statements, comments, etc.

How bash works

- ▶ Bash is a programming language.
 - ▶ Can set variables, use for loops, if statements, comments, etc.
- ▶ There are several special "environment" variables (i.e. \$PATH, \$HOME, \$USER, etc.) that many programs rely on.

How bash works

What happens when you type in a command, say *pwd*?

How bash works

What happens when you type in a command, say *pwd*?

- ▶ Bash runs the program called *pwd*.
- ▶ Where is this program?
 - ▶ Usually under a directory called *bin*, which stands for *binary*.

How bash works

What happens when you type in a command, say *pwd*?

- ▶ Bash runs the program called *pwd*.
- ▶ Where is this program?
 - ▶ Usually under a directory called *bin*, which stands for *binary*.
- ▶ When you type in a command, bash looks for a program with that name under the directories listed in the *\$PATH* environment variable.

How bash works

What happens when you type in a command, say *pwd*?

- ▶ Bash runs the program called *pwd*.
- ▶ Where is this program?
 - ▶ Usually under a directory called *bin*, which stands for *binary*.
- ▶ When you type in a command, bash looks for a program with that name under the directories listed in the *\$PATH* environment variable.

```
$ echo $PATH
```

```
/Users/jackiebaek/.local/bin:/Users/jackiebaek/.cabal/bin:/Applications/ghc-7.10.3.app/Contents/bin:/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/usr/texbin
```

- ▶ *\$PATH* contains is a list of directories separated by :
- ▶ Bash looks into each of these directories to look for the program *pwd*.

Key Takeaways

Key Takeaways

- ▶ Basic commands: ls, cd, pwd, cat, cp, mv, rm, mkdir

Key Takeaways

- ▶ Basic commands: ls, cd, pwd, cat, cp, mv, rm, mkdir
- ▶ Google is your friend.

Key Takeaways

- ▶ Basic commands: `ls`, `cd`, `pwd`, `cat`, `cp`, `mv`, `rm`, `mkdir`
- ▶ Google is your friend.
- ▶ So is *tab* for autocomplete, *arrow keys* for history.

Key Takeaways

- ▶ Basic commands: `ls`, `cd`, `pwd`, `cat`, `cp`, `mv`, `rm`, `mkdir`
- ▶ Google is your friend.
- ▶ So is *tab* for autocomplete, *arrow keys* for history.
- ▶ Be careful with *rm*.

Key Takeaways

- ▶ Basic commands: `ls`, `cd`, `pwd`, `cat`, `cp`, `mv`, `rm`, `mkdir`
- ▶ Google is your friend.
- ▶ So is *tab* for autocomplete, *arrow keys* for history.
- ▶ Be careful with *rm*.
- ▶ Getting comfortable with the terminal can be daunting at first, but it has the potential to greatly boost your efficiency!

Thank you!