
Read the Docs Template Documentation

Release 1.0

Read the Docs

Mar 28, 2019

1	Overview	3
2	What is Unix?	5
3	Unix Tutorial #1: Navigating the directory tree	7
4	Unix Tutorial #2: Copying and Removing Files	11
5	Unix Tutorial #3: Reading Text Files	13
6	Unix Tutorial #4: Shells and Path Variables	17
7	Introduction	19
8	fMRI Tutorial #1: Downloading the Data	21
9	fMRI Tutorial #2: Overview of The Flanker Task	25
10	fMRI Tutorial #3: Looking at the Data	27
11	ROI Analysis	33
12	Glossary	37

This book, *Andy's Brain Book*, is intended for both beginners and more advanced neuroimagers. We will start with basic fMRI analysis, with more advanced analyses being added later. It is the ReadTheDocs companion to [Andy's Brain Blog](#).

If you are completely new to fMRI analysis, I recommend starting with the [installation page](#).

In order to analyze fMRI data, you will need to download an fMRI analysis package. The three most popular are:

- [SPM](#) (Statistical Parametric Mapping, created by University College London)
- [FSL](#) (FMRIB Software Library, created by the University of Oxford)
- [AFNI](#) (Analysis of Functional Neuroimages, created by the National Institutes of Health, USA)

Most laboratories use one of these three packages to analyze fMRI data. Each package is maintained by a team of professionals, and each is updated at least every few years or so. This course will focus on analyzing an online dataset using FSL; in the future, tutorials will be added for SPM and AFNI.

1.1 Installing FSL

FSL can be installed on different operating systems, such as Windows, Macintosh, and Linux. The Windows version requires a virtual machine to run FSL. This course will assume that you have a Macintosh or Linux operating system, and the tutorials will be recorded using an iMac desktop.

The instructions for installing FSL on different operating systems can be found [here](#). The instructions are detailed and will not be repeated here; but for those who are using a Macintosh computer, a screencast can be found here to walk you through the installation and setup: [FSL Installation Video Walkthrough](#)

1.2 Next Steps

If you have never used a command line before, click on the Next button to begin the Unix tutorials. If you do feel comfortable using Unix, then you can skip to the [fMRI Short Course](#).

CHAPTER 2

What is Unix?

Unix is an operating system, just like Macintosh or Windows. The major difference between Unix and other operating systems is that it uses a command line interface, or **command line** for short: Instead of pointing and clicking like you would in a Windows or Macintosh machine, you type the commands that you want to run. The command line is typed in a **Terminal**, a window in which you can only type text. The Terminal emulates (i.e., reproduces the function of) a [console](#).

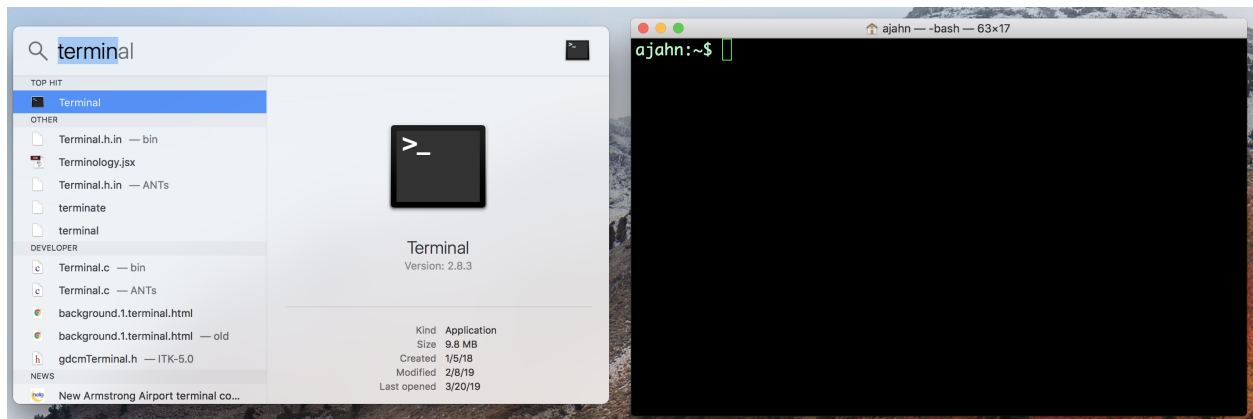


Fig. 1: Macintosh and Linux operating systems have Terminals installed. On a Macintosh, for example, click on the Finder magnifying glass, and type “Terminal”. When you open it, you should see something like the window on the right.

While this may look strange and maybe even tedious, in fact using the command line will allow you to be quicker, more flexible, and more efficient when running programs and analyzing data. You will also need to know Unix in order to use neuroimaging packages that are run from the command line, such as AFNI or FSL. And as you’ll see, becoming fluent in programming will save you time, minimize errors, and make you more versatile as a scientist.

Note: Linux operating systems (such as Ubuntu) come with a Terminal, as do recent versions of Macintosh operating systems. Windows users will need to download a Terminal emulator such as [Cygwin](#). Although I record my tutorials

on a Macintosh computer, once you have downloaded and installed Cygwin, you should be able to follow the rest of the tutorials.

2.1 Video

Click [here](#) to see a video introducing you to Unix. When you have finished watching the video, click the “Next” button down below to begin the Unix tutorial series.

Unix Tutorial #1: Navigating the directory tree

Note: Topics covered: Directories, navigation

Commands used: `pwd`, `cd`, `ls`

3.1 Overview

Like other operating systems, Unix organizes folders and files using a directory tree - also known as a directory hierarchy, or directory structure. At the top of the hierarchy is a folder called `root`, written as a forward slash (`/`). All other folders (also known as directories) are contained within the `root` folder, and those folders in turn can contain other folders.

Think of the directory hierarchy as an upside-down tree: `root` is the base of the tree, and all of the other folders extend from it, just as branches extend from the trunk.

To navigate around your computer, you will need to know the commands `pwd`, `cd`, and `ls`. `pwd` stands for “print working directory”; `cd` stands for “change directory”; and `ls` stands for “list”, as in “list the contents of the current directory.” This is analogous to pointing and clicking on a folder on your Desktop, and then seeing what’s inside. Note that in these tutorials, the words “folder” and “directory” are used interchangeably.

3.2 Video

Click [here](#) to see a video overview of the commands `cd`, `ls`, and `pwd` - the basic commands you will need to navigate around your directory tree.

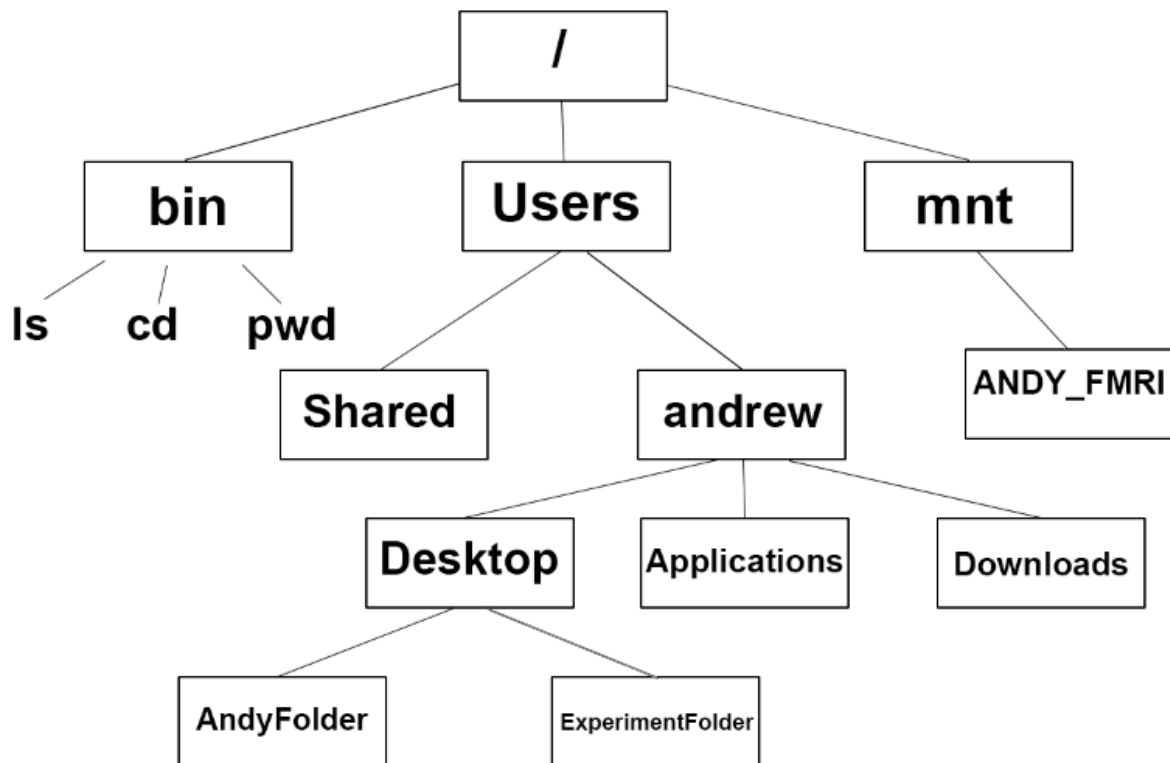


Fig. 1: Root, symbolized by a forward slash (/), is the highest level of the directory tree; it contains folders such as **bin** (which contains binaries, or Unix commands such as **pwd**, **cd**, **ls**, and so on), **mnt** (which shows any currently mounted drives, such as external hard drives), and **Users**. These directories in turn contain other directories - for example, **Users** contains the folder **andrew**, which in turn contains the **Desktop**, **Applications**, and **Downloads** directories. This is how folders and files are organized within a directory tree.

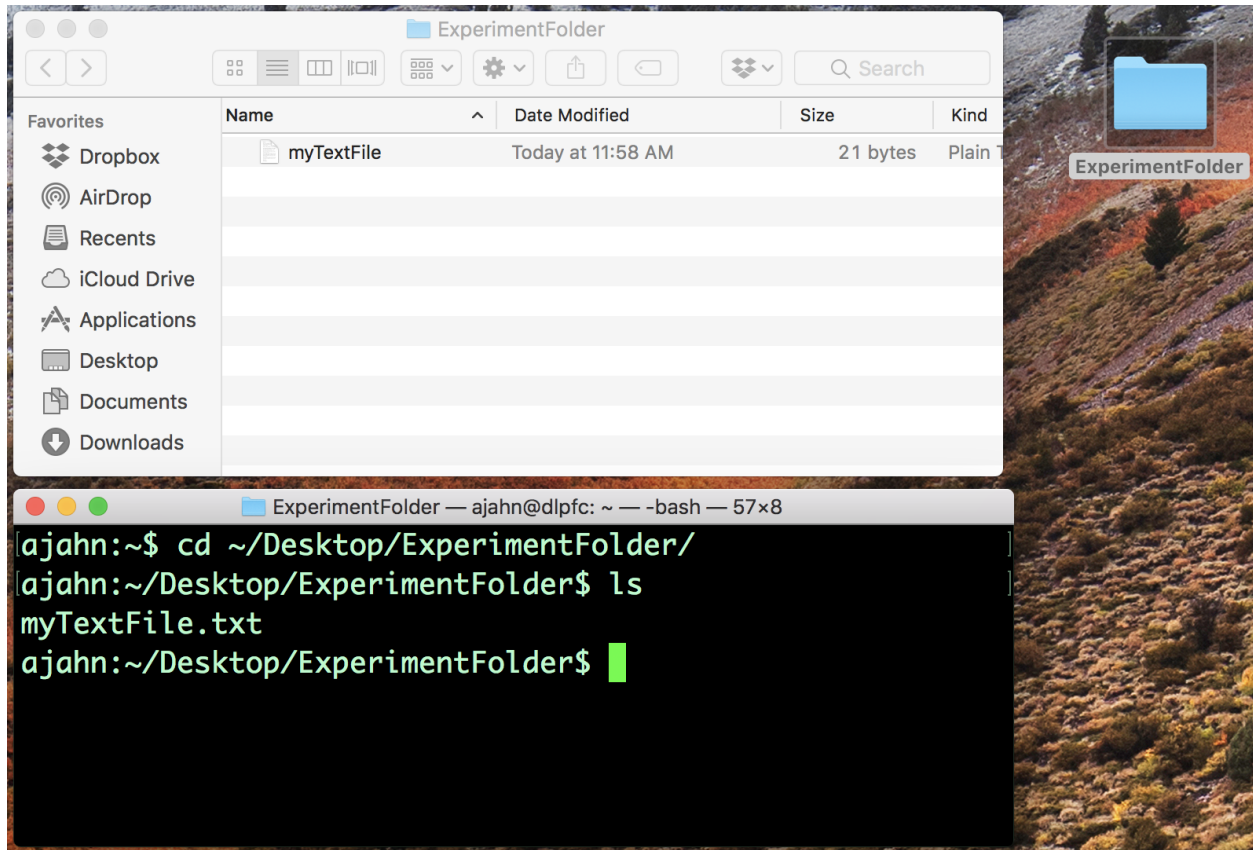


Fig. 2: Navigation in Unix is the same thing as pointing and clicking in a typical graphical user interface. For example, if you have the folder “ExperimentFolder” on my Desktop, you can point and double-click to open it. You can do the same thing by typing `cd ~/Desktop/ExperimentFolder` in the Terminal and then typing `ls` to see what’s in the directory.

3.3 Exercises

When you're done watching the video, try the following exercises:

1. Type `ls ~` and note what it returns; they type `ls ~/Desktop`. How are the outputs different? Why?
2. Navigate to the Desktop by typing `cd ~/Desktop`. Type `pwd` and note what the path is. Then create a new directory using the `mkdir` command, choosing a name for the directory on your own. Navigate into that new directory and think about how your current path has been updated. Does that match what you see from typing `pwd` from your new directory?
3. Define the terms `cd`, `ls`, and `pwd` in your own words.

Unix Tutorial #2: Copying and Removing Files

Note: Topics covered: File manipulation; arguments; options; copying, deleting, and renaming files

Commands used: `cp`, `mv`, `rm`

Now that you have experience getting around your directories with `cd` and `ls`, let's expand our repertoire with the commands `cp`, `mv`, and `rm`. These are **file manipulation** commands, analogous to right clicking on a file and copying and pasting it, renaming or moving it to a folder, or moving it to the trash bin. The `cp` and `mv` commands are more complex than the commands in the previous tutorial, since they require two inputs, or **arguments**. We will also use these commands to introduce the concept of **options**, or flags, which give your commands greater flexibility.

```
ajahn:~$ ls -l
total 112
drwxr-xr-x@ 15 ajahn  UMR00T\Domain Users   510 Jun 29  2018 AFNI_data6
drwxr-xr-x  20 ajahn  UMR00T\Domain Users   680 Feb 18 14:51 ANTs
drwx-----@  4 ajahn  UMR00T\Domain Users   136 Dec  2 08:09 Applications
drwx-----@  4 ajahn  UMR00T\Domain Users   136 Dec  3 17:03 Applications (Parallels)
drwx-----+  7 ajahn  UMR00T\Domain Users   238 Mar 21 15:29 Desktop
drwx-----+ 10 ajahn  UMR00T\Domain Users   340 Sep 14  2018 Documents
drwx-----+ 12 ajahn  UMR00T\Domain Users   408 Mar 21 15:46 Downloads
```

Fig. 1: An example of using the `ls` command with the `-l` option, which lists files and directories in “long” format. This option extends the ability of the `ls` command to not only list the file name, but also show details about when the file was last updated, who has permission to edit the file, and how large the file is.

To see the similarities between using the graphical interface and using the command line, create a text file in an editor of your choice, such as TextEdit. Save the file to your Desktop - in this example, I'll call it `myFile.txt` - and try the following commands, remembering to press the Enter key after each line:

```
cp myFile.txt myFile2.txt
mv myFile.txt myNewFile.txt
rm myFile2.txt
```

The first line creates a copy of `myFile.txt` and labels it `myFile2.txt`; the second line renames `myFile.txt` to `myNewFile.txt`; and the last line removes `myFile2.txt`. As you type these in the command line you should see the files being copied, renamed, and deleted - just as you would if you were using your mouse with the graphical user interface.

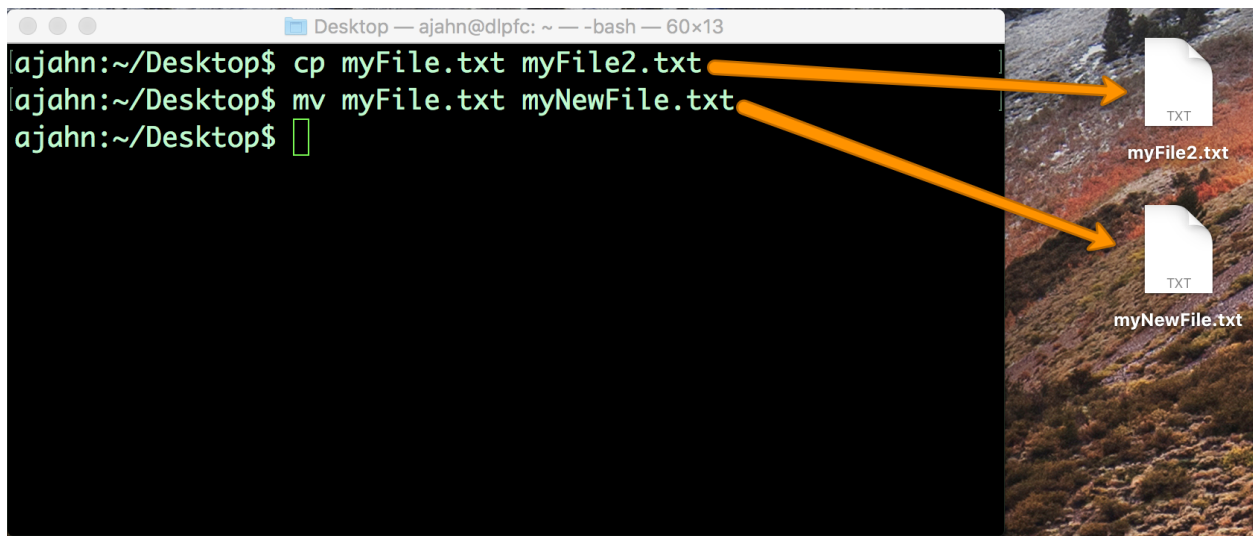


Fig. 2: An illustration of how commands typed in the Terminal have the same effect as copying and renaming files through the GUI.

Warning: There is one important difference between the GUI and the command line when removing files and directories. A file that is removed using the GUI will first be put in the Trash bin, and will only be permanently deleted if you choose to empty the Trash bin. Removing a file with `rm`, on the other hand, permanently deletes the file right away; you will not be able to recover the file after you remove it with `rm`!

4.1 Video

Click [here](#) to see a video walkthrough about copying and removing files. This video will also show you how to move, rename, and delete folders. When you've finished watching it, try these exercises to consolidate what you've learned.

4.2 Exercises

1. Make a new directory in your home directory and either create a couple of new files in that directory, or copy files in to that directory from another directory.
2. Now copy the directory to the Desktop, but append a forward slash to the end of the directory that you are copying. For example, if the directory is called `myFolder`, type `cp -R myFolder/ Desktop`. What happens?
3. Type `man cp` and find the section about the `-R` option. Try using one of the other options with the `cp` command, and observe what happens.

Unix Tutorial #3: Reading Text Files

Note: Topics covered: File manipulation, redirection, streams, stdin, stdout, stderr
Commands used: cat, less, head, wc

The command line is useful for both viewing and manipulating text files. **Manipulation** means editing text - for example, replacing words in text files, or appending text from the command line to the end of a file (also known as **redirection**). This is useful for creating **scripts**, text files containing one or more commands that are run consecutively. In later tutorials, you will use these techniques to automate your analyses, which can save enormous amounts of time.

You can display the contents of a file using the `cat` command, which stands for concatenate. Let's say we have a file on our Desktop called `myFile.txt`, which contains the words one through fifteen (i.e., one, two, three...fifteen), with each number on a separate line. Use the command line to navigate to the Desktop, and then type `cat myFile.txt`. This will print the contents of the file to your command line. This is the same idea as using the GUI to double-click on the text file to see its contents.

We refer to the output from this command as **stdout**, or standard output. The commands that are typed into the Terminal are called **stdin**, or standard input. This touches on the concept of **streams**, or the flow of information into and out of the command line, and we will use these ideas to give us more flexibility in manipulating text files. For now, think of **stdin** as anything you type into the Terminal, and **stdout** as what is returned if the command is run without any errors. If the command that you type does result in an error - for example, because the command was misspelled or because not enough arguments were provided - the text that is output to the Terminal is called **stderr**, or standard error.

The `cat` command is useful for viewing the contents of smaller files, but if the file contains hundreds of lines of text, it is overwhelming to have everything printed to the Terminal at once. To see only a part of the file, we can use the commands `head` and `tail` to see the first few or the last few lines of the file, respectively. Using `myFile.txt` as an example, typing

```
head myFile.txt
```

Would return the first five lines; whereas typing

```
tail myFile.txt
```

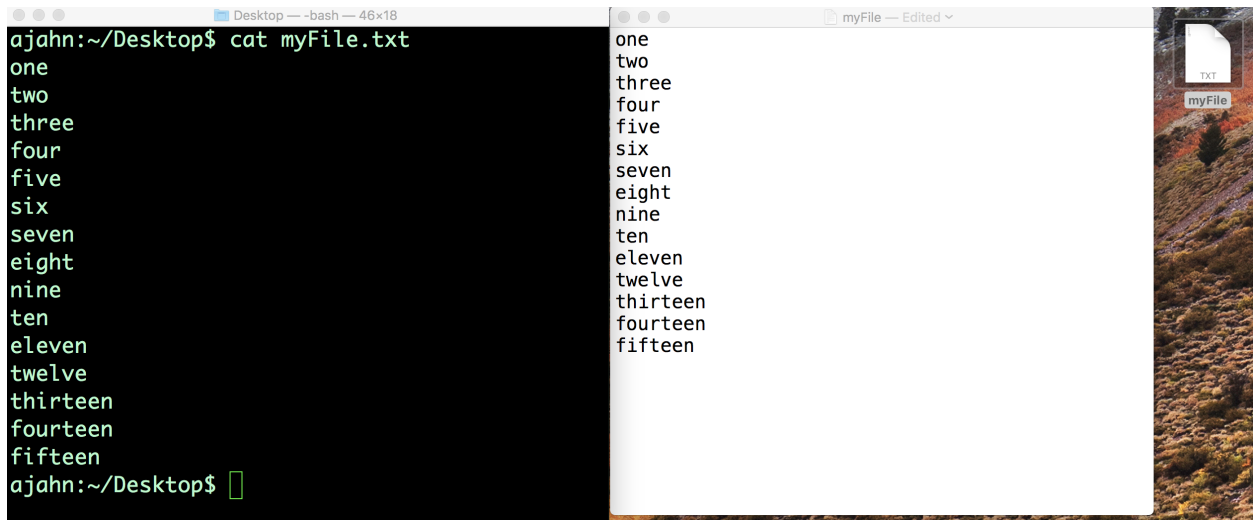


Fig. 1: Using the command line and the GUI to read the contents of a text file. On the left is the command line using the `cat` command, which prints the contents to the Terminal. On the right is the contents of the file displayed after using the mouse to double-click the file.

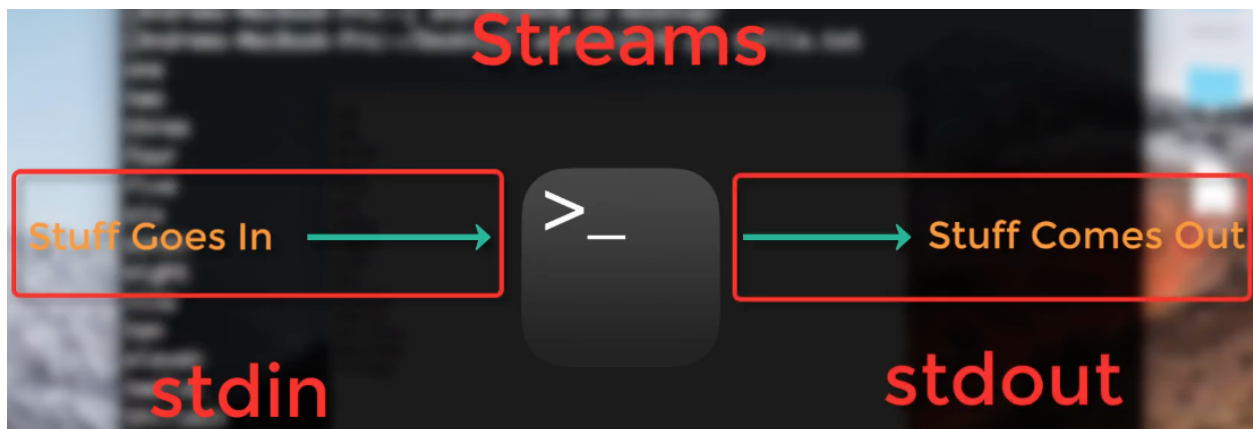


Fig. 2: Illustration of streams in Unix. Whatever is typed into the Terminal is **stdin**, and, if it runs without error, whatever is output is called **stdout**. If there is an error, the output is instead called **stderr**.

Would return the last five lines. Although the default is to return five lines, these commands have an option to display any amount of lines that you choose. For example,

```
head -10 myFile.txt
tail -10 myFile.txt
```

Would return the first ten lines and the last ten lines. Try these out yourself, changing the number of lines that are displayed.

5.1 Redirection

In addition to displaying the results of a command, **stdout** can be used to move or append the output to a file, a concept known as **redirection**. For example, if you type

```
echo sixteen > tmp.txt
```

The word “sixteen” goes into the file tmp.txt instead of being written to standard output. Notice that it creates the file tmp.txt even if it doesn’t exist. However, if we try that again with another string - for example,

```
echo seventeen > tmp.txt
```

It will overwrite the file with whatever we printed to standard output. If you want to append standard output to the end of a file without overwriting the other data in the file, use two greater-than signs. For example, type

```
echo eighteen >> tmp.txt
```

If you type `cat tmp.txt`, you will see both seventeen and eighteen.

Although these examples are trivial, redirection is invaluable for quickly editing text files and for writing **scripts**, which allow you to run analyses for hundreds or thousands of subjects with only a few lines of code.

5.2 Video

Click [here](#) for a video walkthrough of commands for reading text files. This video will also show you how to read help files using the `less` command and a paging window.

5.3 Exercises

1. Create a new file called “tmp.txt” and type whatever you want into the file. Use `cat` to string together both the myFile.txt and tmp.txt files, and redirect the output to create a new file. Print the contents of the new file to stdout.
2. If you have AFNI installed on your machine, use `less` on the command `3dcalc` to find strings matching “Example.” Now try it using the `less` command with an option to ignore whether the letters in the string are upper case or lower case. Hint: To find this option, search for the string “case” in the man file for `less`. (If you have FSL installed instead of AFNI, try the same exercise with the command `fslmerge`.)
3. Unix has a built-in command called `sort` which will sort text numerically or alphabetically. What happens when you use myFile.txt as an argument for `sort`? What about typing this command:

`cat myFile.txt | sort`

In your own words, explain the difference between the two methods.

Unix Tutorial #4: Shells and Path Variables

Note: This section is under construction; the video is complete, however.

Note: Topics covered: paths, variables, shells, FSL, installation, syntax, redirection Commands covered: set, setenv, export, tcsh, bash

Now that you've become more familiar with Unix commands, we can download an fMRI package and install it with Unix. If you haven't already downloaded FSL, watch this video. When you're done, come back to this tutorial.

When you downloaded and installed FSL, you may have seen a few things you didn't completely understand. For example, if you go to your home directory and type "cat .bashrc", you'll see this block of code. To understand what's going on here, you'll need to understand shells, paths, and variables. First, let's talk about shells. Think of the shell as an environment in which you can write Unix commands. We used a shell in the previous tutorials, but you may not have been aware of it. When you open the terminal, it uses a shell to interpret what you're typing. Also, there are many different shells, and each one has a different syntax, or specific way that the words in your command need to be organized - just like in human languages. For example, there are two major shells - the Bourne shell, with a widely-used version called bash, or Bourne-again shell, and the C-shells, of which one popular variation is the t-shell, or tcsh (Fade these in). The commands we've used so far - cd, ls, pwd, and so on - are called built-in commands, and they can be used the same way in both shells. But there are important differences when you do a more advanced operation, such as setting a variable.

6.1 Video

Click [here](#) for a video walkthrough explaining what shells and path variables are.

CHAPTER 7

Introduction

This course will show you how to analyze an fMRI dataset from start to finish. We will begin by downloading a sample dataset and inspecting the anatomical and functional images for each subject. We will then preprocess the data, which removes noise and enhances the signal in the images. Lastly, we will fit a model to each subject to measure the strength of the signal under different conditions - for example, we can take the difference of the signal between conditions A and B of the experiment to see which one leads to a larger BOLD response.

Once a model has been created for each subject and the magnitude of the signal calculated for each condition, we can do any kind of group analysis we like: Paired t-tests, between-group t-tests, interactions, and so on. The goal of this course is to calculate a simple within-subjects contrast between two conditions, and test whether it is significant across subjects. You will also learn how to create figures showing whole-brain analyses similar to what you see published in the neuroimaging journals, and how to do a region of interest (ROI) analysis.

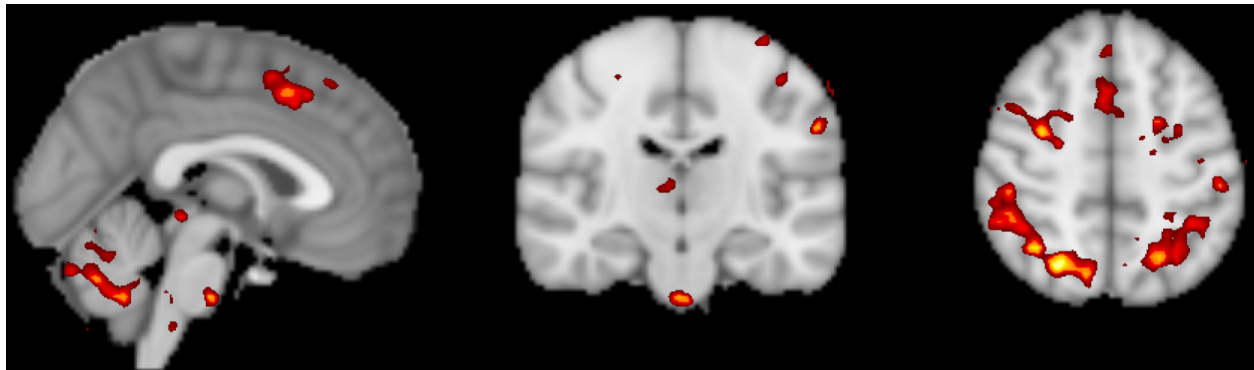


Fig. 1: A figure showing group-level results from the data used in this course, represented as a z-statistic map. Brighter colors indicate larger z-scores. You will begin by preprocessing the raw data and end with creating a statistical map like this one.

This course is designed to build your confidence in working with fMRI data, increase your fluency with the basic terms of fMRI analysis, and help you make educated choices during each step. Exercises at the end of each chapter will allow you consolidate what you've learned and prepare you for the next chapter. Once you have mastered these fundamentals, you will be able to apply them to other datasets of your choosing.

Note: We will not be covering MRI physics in depth. For a review of that topic, I recommend chapters 1-5 of the book *Functional Magnetic Resonance Imaging*, by Huettel, Song, & McCarthy (3rd Edition). Also see Allen Elster's excellent [MRI Questions](#) website for useful illustrations of MRI concepts.

fMRI Tutorial #1: Downloading the Data

8.1 Overview


For this course we will be analyzing an fMRI dataset that used the Flanker task. The dataset can be found [here](#) on the [OpenNeuro](#) website, an online repository for neuroimaging data.

Download the dataset by clicking on the “Download” button at the top of the page. The dataset is about 2 Gigabytes, and comes in a zipped folder. Extract it by double-clicking on the folder, and then move it to your Desktop.

After you have downloaded and unzipped the dataset, click on the Next button for an overview of the experimental task used in this study.

Dataset File Tree

<div><div> Flanker task (event-related)</div><div><div>– CHANGES</div><div><div> DOWNLOAD</div><div> VIEW</div></div></div><div><div>– dataset_description.json</div><div><div> DOWNLOAD</div><div> VIEW</div></div></div><div><div>– participants.tsv</div><div><div> DOWNLOAD</div><div> VIEW</div></div></div><div><div>– README</div><div><div> DOWNLOAD</div><div> VIEW</div></div></div><div><div>– T1w.json</div><div><div> DOWNLOAD</div><div> VIEW</div></div></div><div><div>– task-flanker_bold.json</div><div><div> DOWNLOAD</div><div> VIEW</div></div></div><div><div>–  derivatives</div></div><div><div>–  sub-01</div><div><div>–  anat</div><div><div>– sub-01_T1w.nii.gz</div><div><div> DOWNLOAD</div><div> VIEW</div></div></div></div><div><div>–  func</div><div><div>– sub-01_task-flanker_run-1_bold.nii.gz</div><div><div> DOWNLOAD</div><div> VIEW</div></div></div><div><div>– sub-01_task-flanker_run-1_events.tsv</div><div><div> DOWNLOAD</div><div> VIEW</div></div></div><div><div>– sub-01_task-flanker_run-2_bold.nii.gz</div><div><div> DOWNLOAD</div><div> VIEW</div></div></div></div></div></div>	22		Chapter 8. fMRI Tutorial #1: Downloading the Data


OpenNEURO

Versions

v00001

2018-07-14

v58016286cce88d0009a335df

2018-07-14

Flanker task (event-related)

uploaded on 2016-10-14 - over 2 years ago

last modified 2018-07-14 - 8 months ago

authored by Kelly AMC, Uddin LQ, Biswal BB, Castellanos FX and Milham MP

★ 1

👤 0

👁 0

📄 0

🟢 Published

fMRI Tutorial #2: Overview of The Flanker Task

This dataset uses the Flanker task, which is designed to tap into a mental process known as cognitive control. For this course, we're going to define cognitive control as the ability to ignore irrelevant stimuli in order to do the task correctly.

In the Flanker task, arrows point either to the left or the right, and the subject is instructed to press one of two buttons indicating the direction of the arrow in the middle. If it's pointing to the left, the subject presses the "left" button; if it's pointing to the right, the subject presses the "right" button. The middle arrow is flanked by other arrows which either point in the same direction as the middle arrow, or point in the opposite direction from the middle arrow.

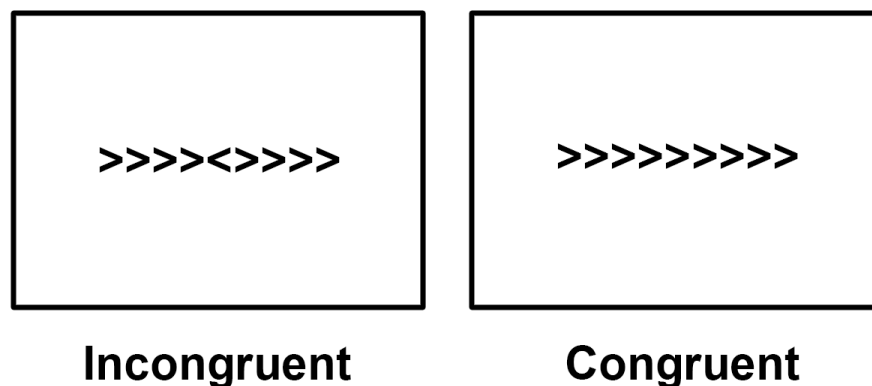


Fig. 1: An example of the two conditions of the Flanker task. In the Incongruent condition, the central arrow (which the subject is focusing on) points in the opposite direction as the flanking arrows; in the Congruent condition, the central arrow points in the same direction as the flanking arrows. In this example the correct response in the Incongruent condition would be to push the "left" button, and the correct response in the Congruent condition would be to push the "right" button. To run through a version of the Flanker task yourself, click [here](#).

You can imagine that the task is easier if the central arrow points in the same direction as the flanking arrow, and more difficult if it points in the opposite direction. We'll call the former condition the "Congruent" condition and the latter the "Incongruent" condition. Subjects are typically slower and more inaccurate in the Incongruent condition,

and faster and more accurate in the Congruent condition. Since the difference in reaction times is robust and reliable, it follows that in our fMRI data we should see a noticeable difference in the BOLD response as well.

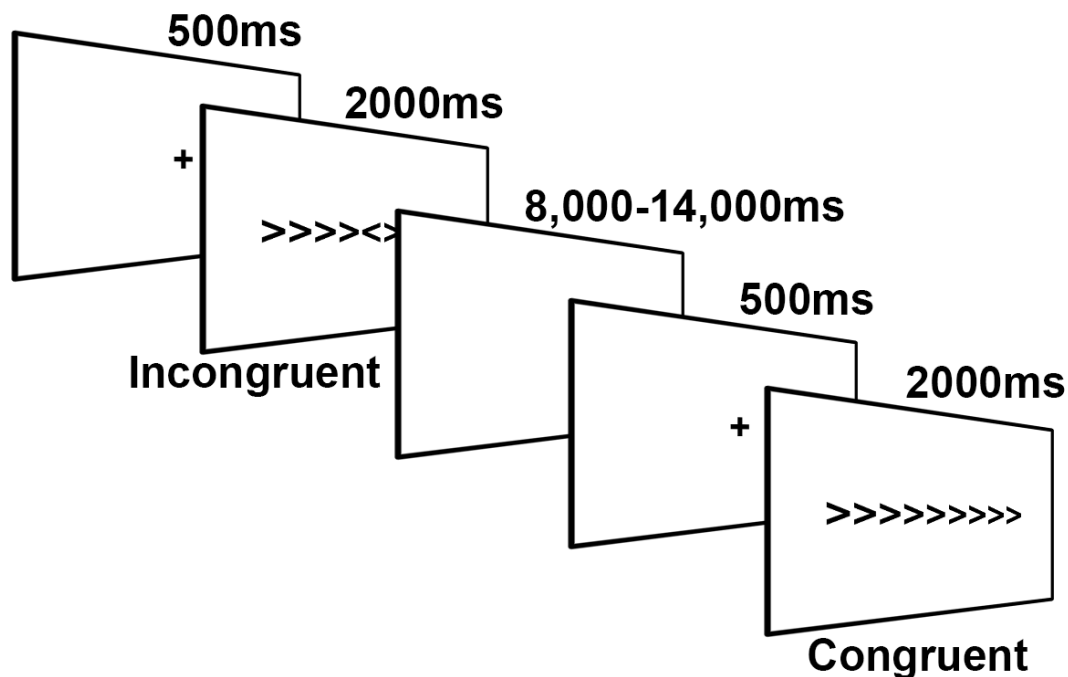


Fig. 2: Illustration of the Flanker task for this study, adapted from Kelly et al. (2008). The subject is shown a fixation cross in order to focus on the center of the screen, and then either a Congruent or Incongruent Flanker trial is presented for 2000ms. During the trial the subject presses either the left or right button. A jittered interval follows which lasts anywhere from 8,000ms to 14,000ms. (Note that jittered intervals typically increment in seconds; in this case, the jitter for a given trial would be a random selection of one of the following: 8,000ms, 9,000ms, 10,000ms, 11,000ms, 12,000ms, 13,000ms, and 14,000ms) Another fixation cross is presented to begin the next trial.

Our goal is to estimate the size of the BOLD response to each condition, and then contrast (i.e., take the difference of) the two conditions to see whether they are significantly different from each other.

Note: This brings up an important point about good practice for designing fMRI studies: If you can design a behavioral task that produces a strong and reliable effect, you will increase your odds of finding an effect in your imaging data. fMRI data is notoriously noisy - if you don't see a behavioral effect in your study, you most likely will not find an effect in your imaging data either.

fMRI Tutorial #3: Looking at the Data

10.1 Overview

Now that you’ve downloaded the dataset, let’s see what it looks like. If the dataset has been downloaded to your Download directory, navigate to the Desktop and type the following:

```
mv ~/Downloads/ds000102_0001/ Flanker
```

Which will rename the folder to `Flanker` and put it on your Desktop.

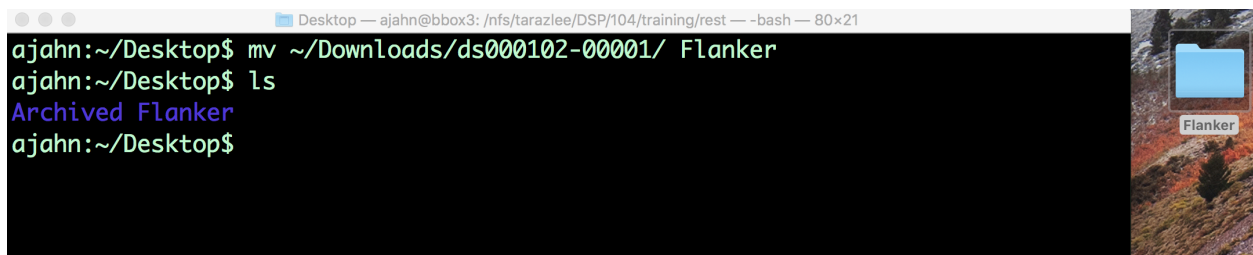


Fig. 1: After downloading the Flanker dataset, type the command above to move it to your Desktop.

As you saw in the previous [Data Download page](#), the dataset has a standardized structure: Each subject folder contains an anatomical directory and a functional directory labeled `anat` and `func`, and these in turn contain the anatomical and functional images, respectively. (The `func` directory also contains **onset times**, or timestamps for when the subject did either a Congruent or Incongruent task.) This format is known as **BIDS**, or Brain Imaging Data Structure, which makes it easy to organize and find your data.

```

ajahn:~/Desktop$ ls Flanker/
CHANGES                sub-06                sub-17
README                 sub-07                sub-18
T1w.json               sub-08                sub-19
dataset_description.json sub-09                sub-20
derivatives            sub-10                sub-21
participants.tsv       sub-11                sub-22
sub-01                 sub-12                sub-23
sub-02                 sub-13                sub-24
sub-03                 sub-14                sub-25
sub-04                 sub-15                sub-26
sub-05                 sub-16                task-flanker_bold.json
ajahn:~/Desktop$ ls Flanker/sub-01
anat func
ajahn:~/Desktop$ ls Flanker/sub-01/func
sub-01_task-flanker_run-1_bold.nii.gz sub-01_task-flanker_run-2_bold.nii.gz
sub-01_task-flanker_run-1_events.tsv  sub-01_task-flanker_run-2_events.tsv
ajahn:~/Desktop$ ls Flanker/sub-01/anat
sub-01_T1w.nii.gz
ajahn:~/Desktop$

```

Fig. 2: Example of the BIDS format. Note that the `func` directory contains both the functional runs - in this case, two runs - and corresponding “events.tsv” files, which contains onsets, or timestamps of which condition happened at what time. You can open these up in a text editor or as a spreadsheet.

10.2 Inspecting the Anatomical Image

Whenever you download imaging data, you should first check the anatomical and functional images to inspect them for any problems - scanner spikes, incorrect orientation, poor contrast, and so on. It will take some time to develop an eye for what these problems look like, but with practice it will become easier and quicker to check your data.

Let’s take a look at the anatomical image in the `anat` folder for `sub-08`. Navigate to the folder and then type

```
fsleyes sub-01_T1w.nii.gz
```

This will open the anatomical image in `fsleyes`, FSL’s image viewer.

Inspect the image by clicking and dragging the mouse around. You can switch viewing panes by clicking in the corresponding window. Note that the other windows are updated in real time as you move your mouse around. This is because MRI data is collected as a three-dimensional image, and moving along one of the dimensions will change the other windows as well.

Note: You may have noticed that this subject appears to be missing his face. That is because the data from Open-Neuro.org have been **deidentified**: Not only has information such as name and date of scanning been removed from the header, but the faces have also been erased. This is a common step in order to ensure the subject’s anonymity.

As you continue to inspect the image, here are two things you can watch out for:

1. Lines that look like ripples in a pond. These are called **Gibbs Ringing Artifacts**, and they may indicate an error in the reconstruction of the MR signal from the scanner. These ripples may also be caused by the subject



Fig. 3: The anatomical image displayed in fsleyes. The contrast seems low between the grey and white matter, but this is because the blood vessels of the neck (indicated by orange arrows) are much brighter than the rest of the brain.

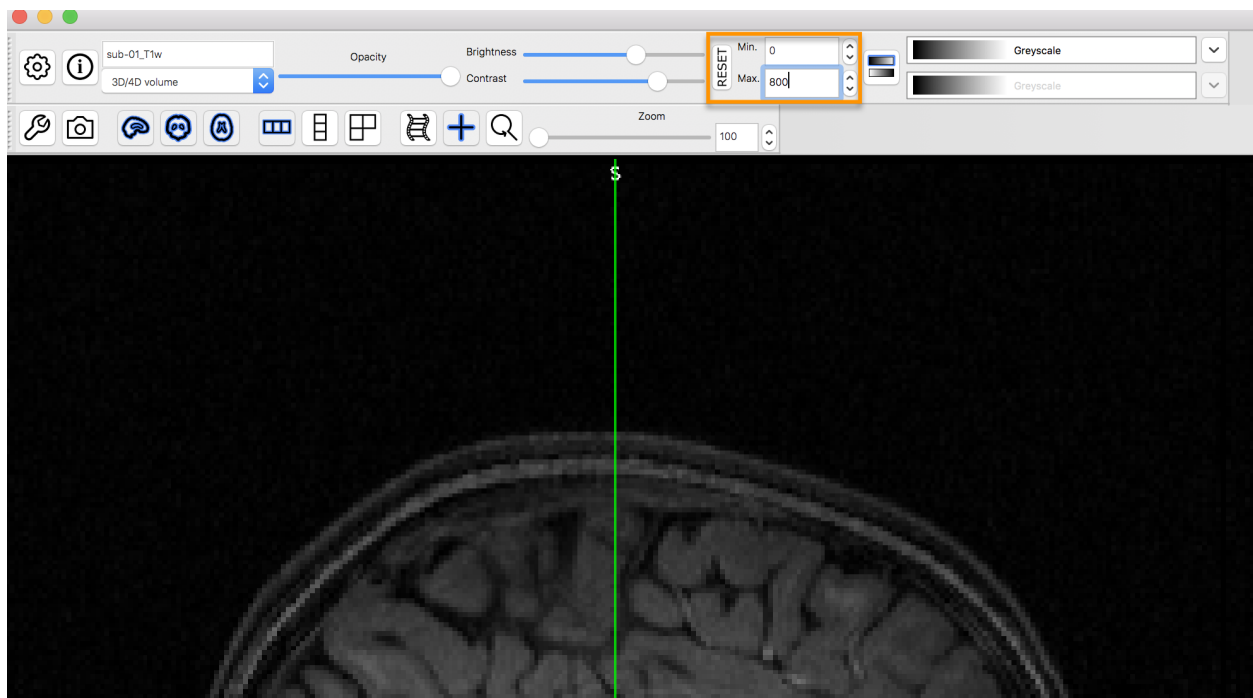


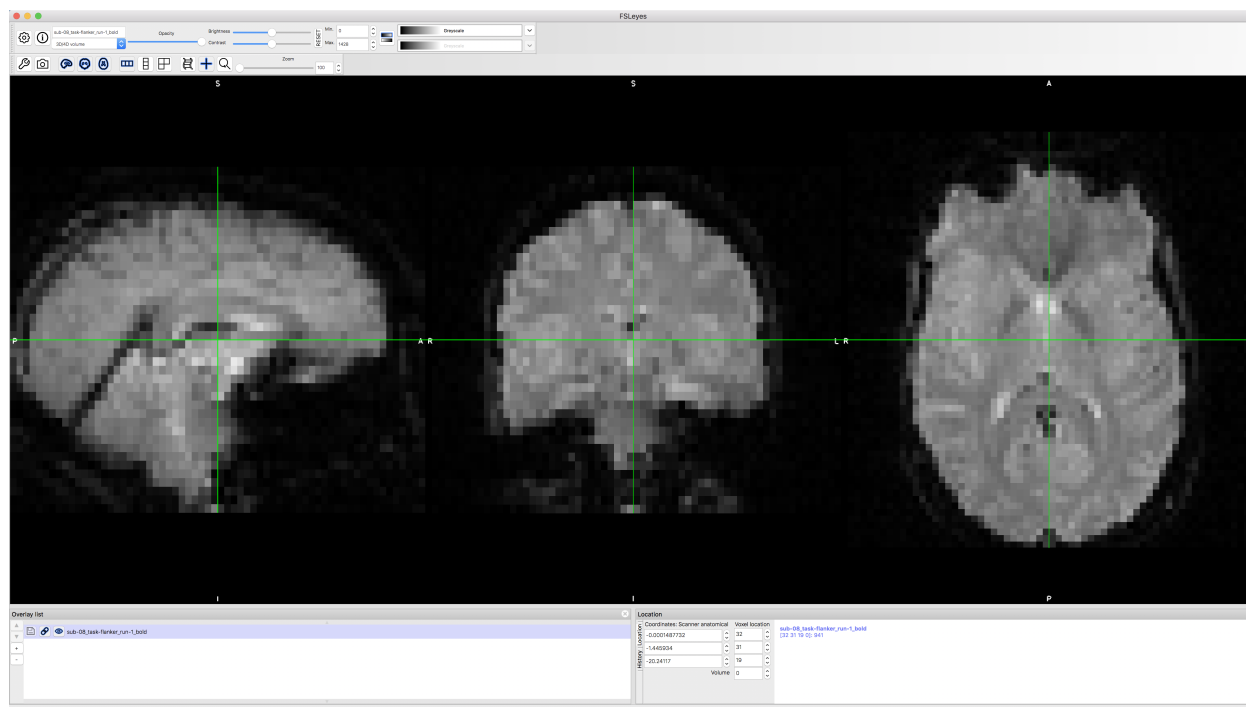
Fig. 4: This can be fixed by changing the numbers in the contrast box. Here, the Maximum has been lowered to 800, capping the brightest signal at that value. This makes it easier to see the contrast between the tissues.

moving too much during the scan. In either case, if the ripples are large enough, they may cause preprocessing steps like brain extraction or normalization to fail.

2. Abnormal intensity differences within the grey or the white matter. These may indicate pathologies such as aneurysms or cavernomas, and they should be reported to your radiologist right away; make sure you are familiar with your laboratory's protocols for reporting artifacts. For a gallery of pathologies you may see in an MRI image, click [here](#).

10.3 Inspecting the Functional Images

When you are done looking at the anatomical image, click on `Overlay -> Remove All` from the menu at the top of your screen. Then, click on `File -> Add from File`, navigate to sub-08's func directory, and select the image ending in `run-1_bold.nii.gz`. This image also looks like a brain, but it's not as clearly defined as the anatomical image. This is because the **resolution** is lower. It is typical for a study to collect a high-resolution T1-weighted (i.e., anatomical) image, and lower-resolution functional images, in part because we collect the functional images much more quickly.



Many of the quality checks for the functional image are the same as with the anatomical image: Watch out for extremely bright or extremely dark spots in the grey or white matter, as well as for image distortions such as abnormal stretching or warping. One place where it is common to see a little bit of distortion is in the orbitofrontal part of the brain, just above the eyeballs. There are ways to reduce this distortion, but for now we will ignore it.

Another quality check is to make sure there isn't excessive motion. Functional images are often collected as a time-series; that is, multiple volumes are concatenated together into a single dataset. You can rapidly flip through all of the



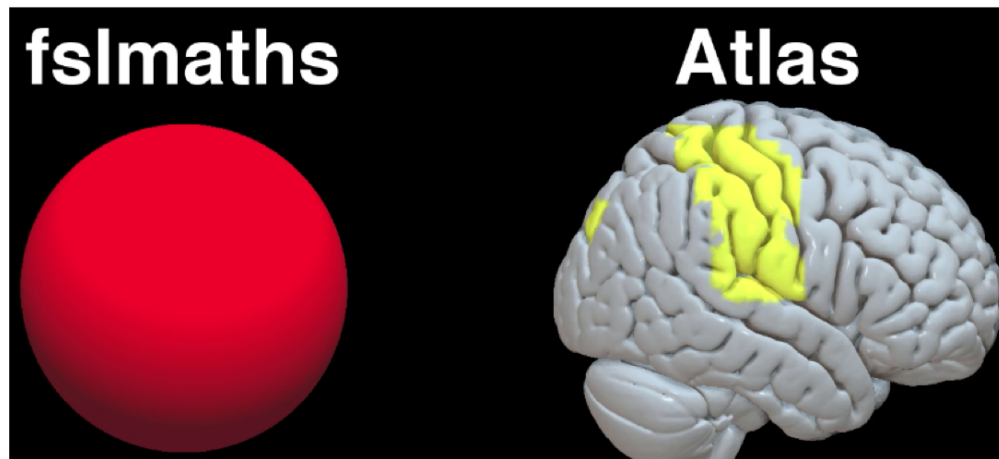
volumes like pages of a book by clicking on the icon in fsleyes. Note any jerky, sudden movements in any of the viewing panes. During preprocessing, we will quantify how much motion there was in order to decide whether to keep or to discard that subject's data.

10.4 Video

Follow along [here](#) for a demonstration of quality checking fMRI data. When you are finished, click on the Next button to learn about preprocessing the data.

11.1 Background

We've covered a lot of ground so far, and by this time you should be able to run a group-level analysis from start to finish: Processing the subjects, creating a model, and then fitting the model for each run for each subject. Group level analyses then allow you to determine whether an effect is consistent enough in the same area across subjects in order to pass a significance threshold that we define. We then used FSLeyes to view the results and get a sense for whether the results seem reasonable. Given previous studies and theory about what the prefrontal cortex does, looking at these maps seemed to lend evidence to our hypothesis that incongruent trials led to greater activity than congruent trials.



However, everything we've done so far is what we call whole-brain analysis. In a whole-brain analysis, we estimate a model at each voxel - that same "mass univariate" approach I discussed last week - and create contrast maps that cover the entire brain. However, we may not be interested in the whole brain. For example, we may not be interested in the cerebellum, and instead only have specific hypotheses about activation within the prefrontal areas. In that case we would gain more power by restricting our analysis to a subset of voxels. Furthermore, the whole-brain maps that we generate can hide important details about the effects that we're studying. We may find a significant effect of incongruent-congruent, but the reason the effect is significant could be because incongruent is greater than congruent, or because congruent is much more negative than congruent, or some combination of the two. The only way to

determine what is driving the effect is by ROI analysis, and this is especially important when dealing with interactions and more sophisticated designs.

This practical will be an introduction to what ROIs are and what they look like, in addition to how to create ROIs and do ROI analysis in FSL. Lastly, we will cover some concepts specific to ROI analysis that you should be aware of – in particular, non-independence and circularity.

===VIDEO: ROI INTRODUCTION===

Have them watch this introductory video: <https://www.youtube.com/watch?v=196ymIPMMww>

And also point out that the SPM tutorial on ROI analysis begins at about 1:48, which they can ignore for now. In other words, watch the video up to the 1:48 mark.

Next, watch this video about how to do ROI analysis in FSL using featquery:

<https://www.youtube.com/watch?v=N9hE0vAztnQ>

I'll need to make another video about how to do it from the command line. Sad!

===END VIDEO=== Let's begin with what an ROI is. An ROI, or region of interest, is a subset of voxels that you want to restrict your analysis to, in order to increase power. This ROI can be defined based on theory - for example, an anatomical region that you assume is involved in your condition, based on previous papers or models - or the coordinates of activation from other studies investigating the same research question you are.

===VIDEO: CREATING MASKS IN FSLEYES===

SCRIPT:

To do an anatomical ROI analysis, first open FSLEYES and open the standard brain you warped to (in this case, the MNI152_T1_2mm brain). Open up the Atlas panel and select an anatomical region by clicking on a region and selecting "Show" in the Atlas panel. From the display window, highlight the mask you just created and click on the Disk icon to save it. Rename it to whatever you want.

===END VIDEO===

Today we will focus on how to do an ROI analysis using `fslmeants`, a command including in the FSL library. First we will need to navigate to the directory containing the contrast we are interested in. Remember that we've computed three contrasts: 1)Incongruent 2)Congruent 3)Incongruent-Congruent

These will be found within the 2ndLevel folder (here, `Flanker_2ndLevel.gfeat`) which contains one contrast estimate per subject (i.e., 26 for each contrast). To do an ROI analysis for Incongruent-Congruent, we would need to navigate to: `Flanker_2ndLevel.gfeat/cope3.feats/zstat` Which contains one z-statistic per subject. You can merge the z-statistics together using `fslmerge -t allZstats.nii.gz zstat*` Which uses a wildcard ("*") to indicate every file that begins with "zstat". You can then extract the data from the subject (one contrast estimate per subject) using `fslmeants`: `fslmeants -i allZstats.nii.gz -m <path/to/mask/>PCG.nii.gz` In which you can replace `<path/to/mask/>` with the path pointing to your mask. For example, if your mask was saved in a directory three levels above the current one, you would write: `fslmeants -i allZstats.nii.gz -m ../../PCG.nii.gz` The same procedure can be done for contrasts 1 and 2 by navigating to their corresponding directories.

CONCEPT: DOUBLE DISSOCIATIONS

However, only reporting the contrast estimate, or interaction term if you have one, doesn't tell you what is driving the effect. To take a simple example, assume that we ran an experiment contrasting left and right button presses, and let's say that we found a significant contrast effect of Left-Right. It could be that in this ROI, Left button presses elicit more activation than Right button presses; or that Left button presses aren't significantly different from zero, but Right button presses are negative; or it could be some combination of the two. The only way to find out is to extract the parameter estimates for Left and Right button presses separately, and then plot them and run the necessary tests. ROIs are also useful for testing double dissociations, or whether a condition activates region A but not region B, while another condition activates region B, but not region A. In our current example, we may want to test whether Left button presses are selective for the right motor cortex, but not the left motor cortex, and vice versa for Right button presses.

In this case we would have to create another ROI in the left motor cortex. Again, I want to emphasize that these are placeholders you can use for your data, which are probably far more interesting.

CONCEPT: DOUBLE-DIPPING (AKA BIASED OR CIRCULAR ANALYSIS)

One potential pitfall with ROI analyses that you should avoid is called non-independence, or biased ROI analyses. You may also hear them referred to as circular ROI analyses. This is a very important conceptual and statistical issue that has a long history going back a decade.

The problem is this. Let's say that you've just run a whole-brain analysis of Left-Right, and you get a contrast map that looks like this, nice and significant, thresholded at $p < 0.001$ and even doing all that stuff Eklund talked about. So you know you have an effect, but let's say you didn't have a hypothesis going into the experiment about where you would find the effect. What you may be tempted to do is use the voxels that passed your significance threshold as an ROI, and then run inferential statistics on parameters extracted from that ROI. This is a biased analysis because these voxels by definition are going to have significant results; and furthermore it is likely they will contain noise that biases their effect size to be artificially high, and it's impossible to tell how much. Any inference you do in this ROI will therefore be invalid. There are a couple of ways to avoid this problem, such as creating an anatomical region or a sphere based on the peak coordinates from another study like we just did.

CHAPTER 12

Glossary

- terms