Introduction to the Command Line Interface

Jeffrey Leek
Johns Hopkins Bloomberg School of Public Health

What is the Command Line Interface?

Nearly ever computer comes with a CLI

- Windows: Git Bash (See "Introduction to Git")
- Mac/Linux: Terminal

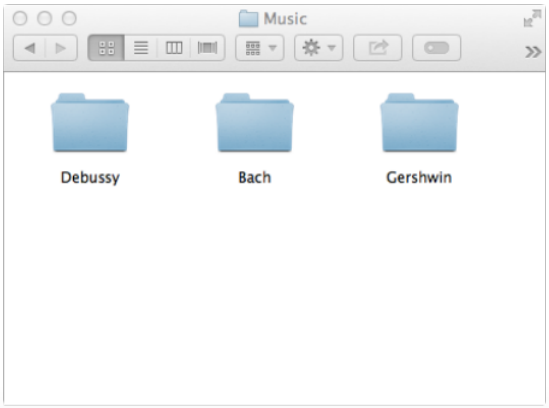
What can the CLI do?

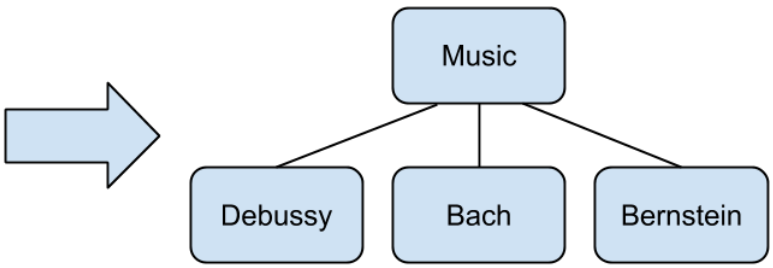
The CLI can help you:

- Navigate folders
- Create files, folders, and programs
- Edit files, folders, and programs
- Run computer programs

Basics of Directories

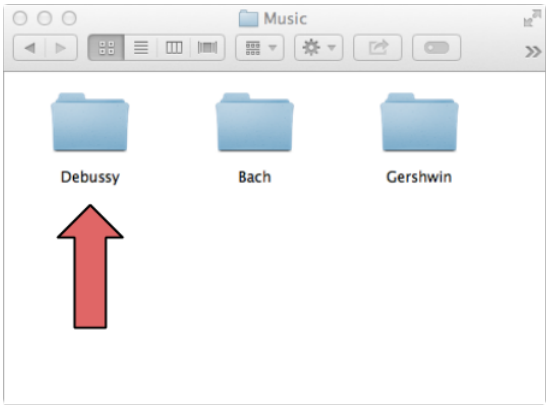
- "Directory" is just another name for folder
- Directories on your computer are organized like a tree
- Directories can be inside other directories
- We can navigate directories using the CLI

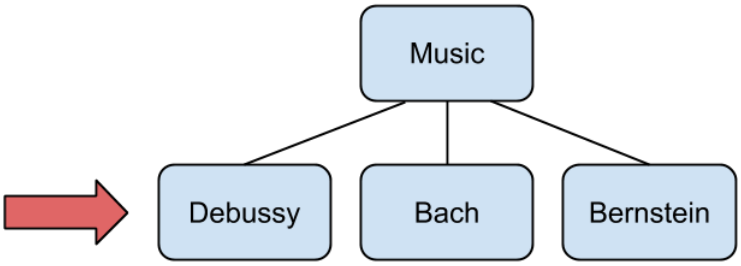




Basics of Directories

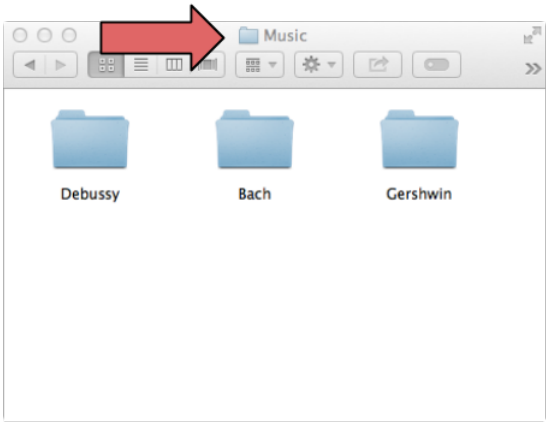
- My "Debussy" directory is contained inside of my "Music" directory

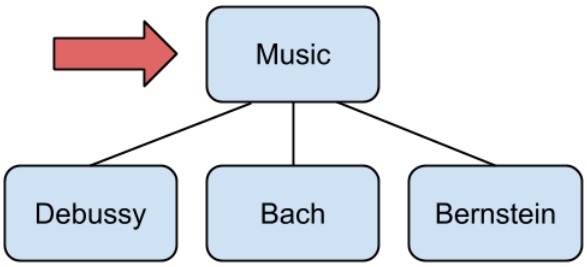




Basics of Directories

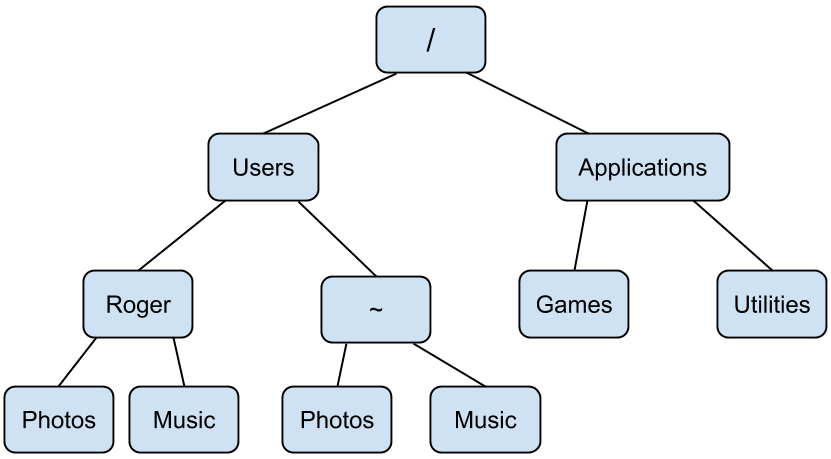
- One directory "up" from my Debussy directory is my Music directory





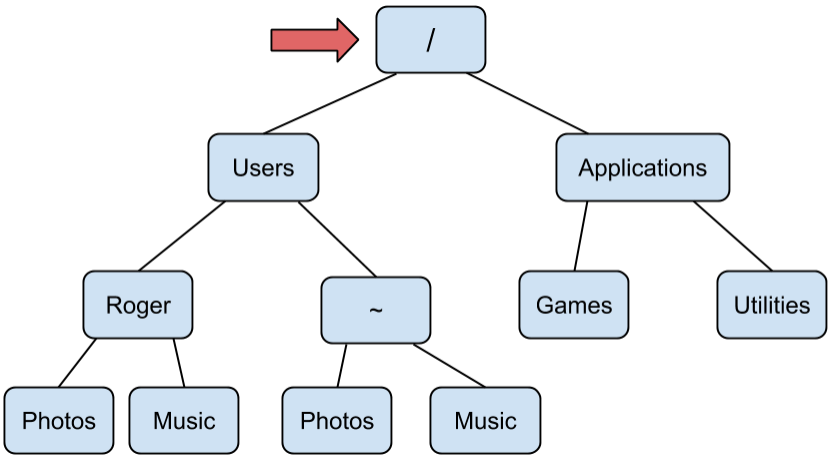
Your computer's directory structure

- The directory structure on your computer looks something like this



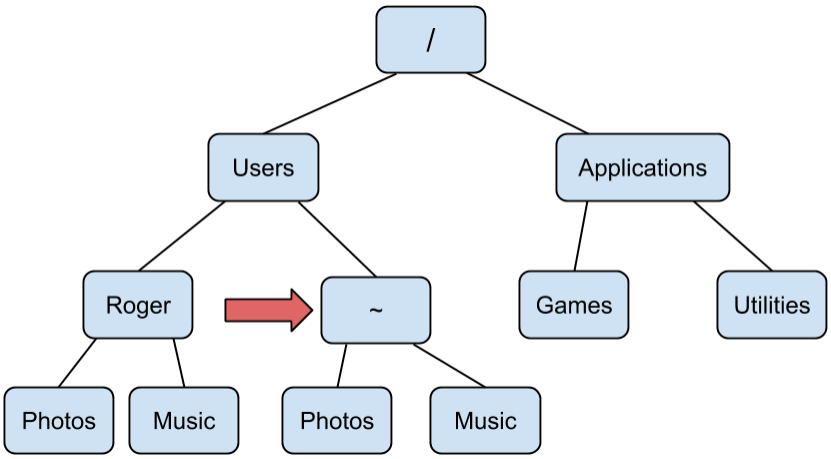
Special directories: root

- The directory at the top of the tree is called the root directory
- The root directory contains all other directories
- The name of this directory is represented by a slash: `/`



Special directories: home

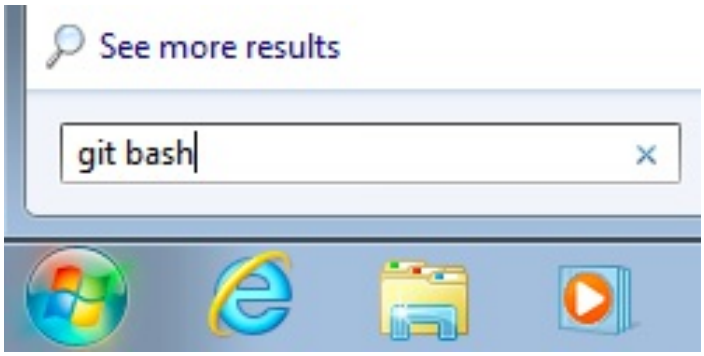
- Your home directory is represented by a tilde: ~
- Your home directory usually contains most of your personal files, pictures, music, etc.
- The name of your home directory is usually the name you use to log into your computer



Navigating directories with the CLI

Windows users:

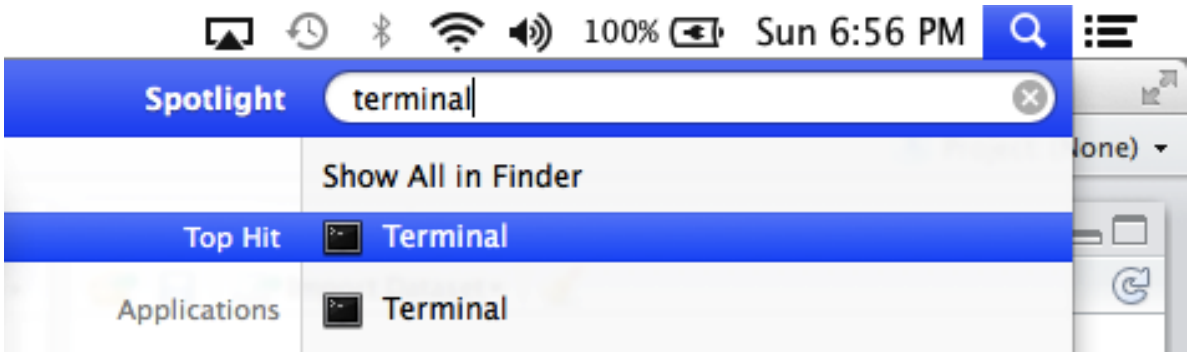
- Open the start menu
- Search for Git Bash
- Open Git Bash



Navigating directories with the CLI

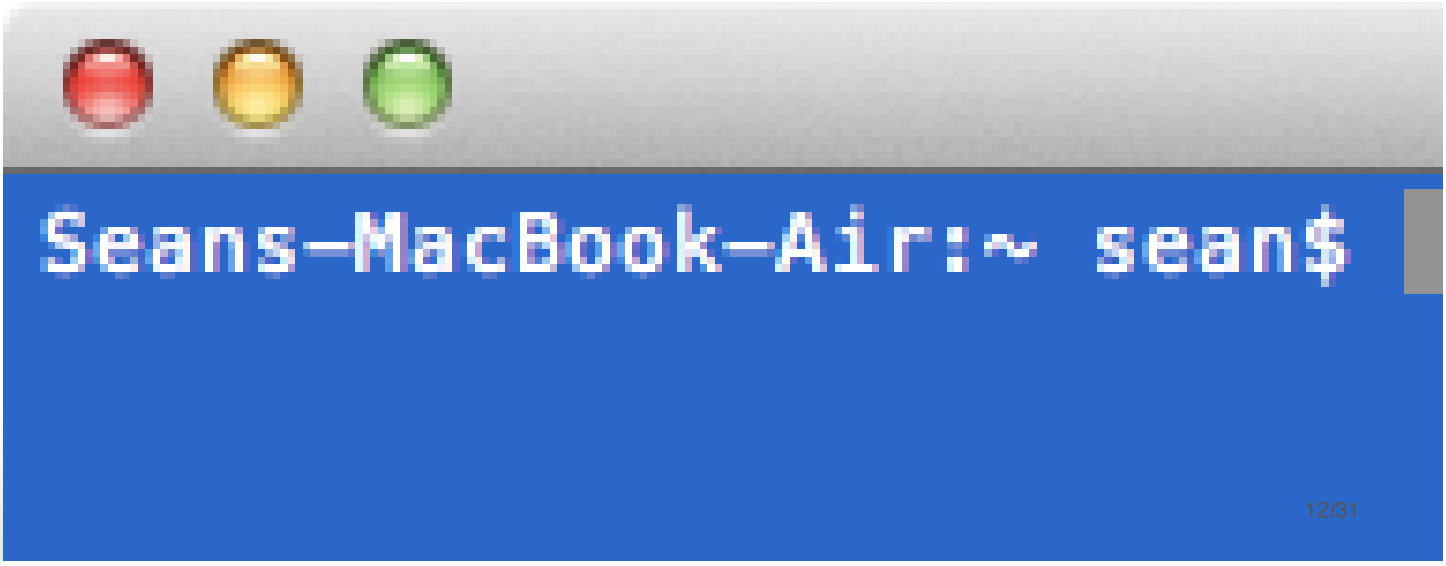
Mac users:

- Open Spotlight
- Search Terminal
- Open Terminal



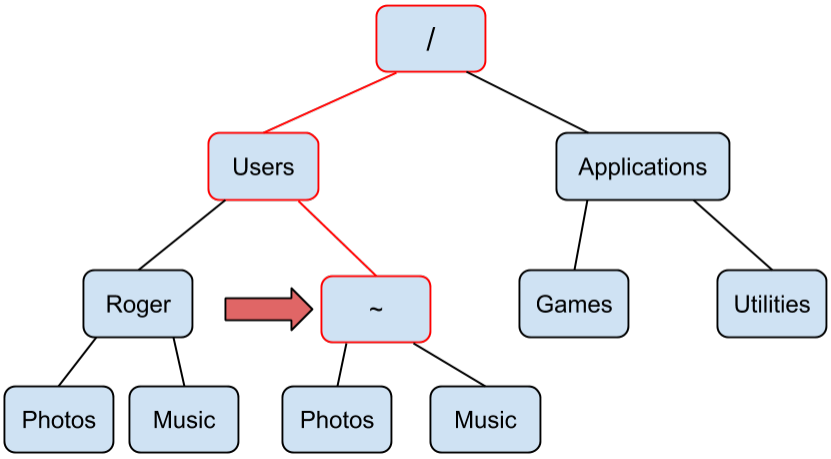
CLI Basics

- When you open your CLI you will see your prompt, which will looks something like the name of your computer, followed by your username, followed by a `$`
- When you open your CLI you start in your home directory.
- Whatever directory directory you're currently working with in your CLI is called the "working directory"



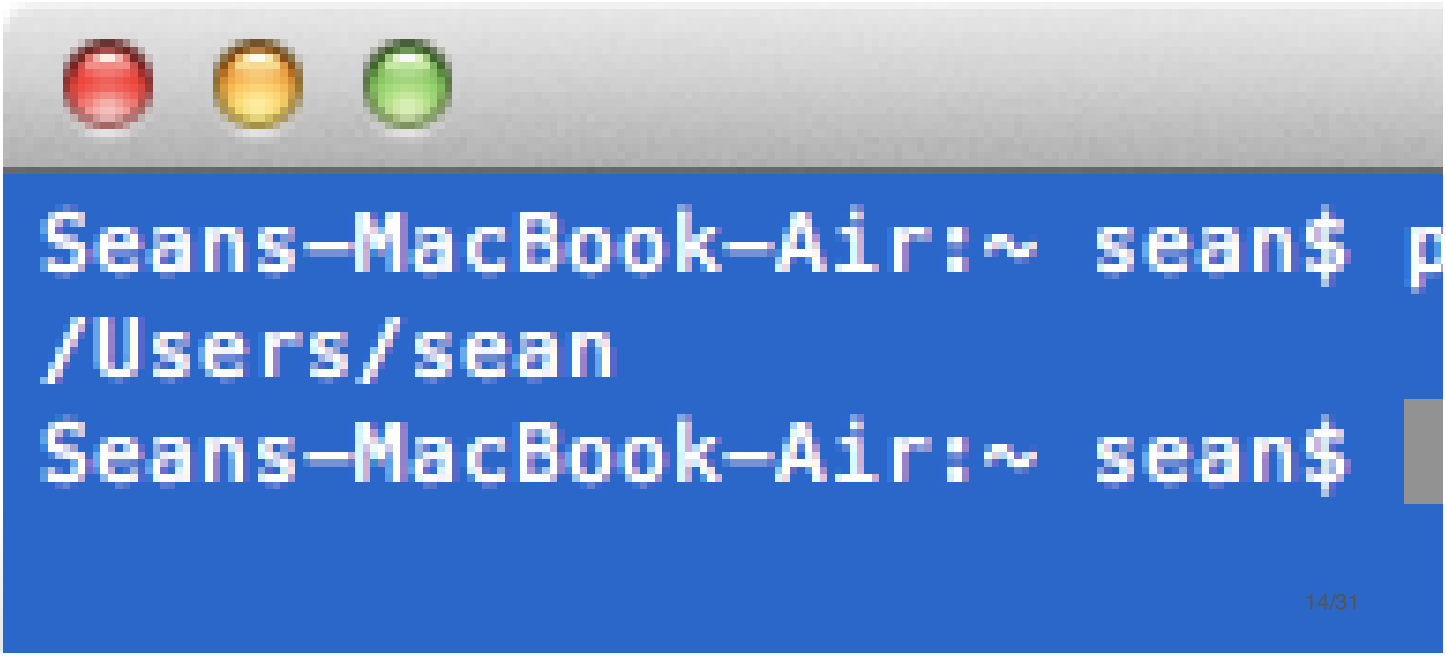
CLI Basics

- You can imagine tracing all of the directories from your root directory to the directory you're currently in.
- This is called the "path" to your working directory.



CLI Basics

- In your CLI prompt, type `pwd` and press enter.
- This will display the path to you're working directory.
- As you can see we get the prompt back after entering a command.



CLI Commands

- You use the CLI prompt by typing in a command and pressing enter.
- `pwd` can be used at any time to display the path to your working directory (`pwd` is an abbreviation for "print working directory")

CLI Commands

- CLI commands follow this recipe: ***command flags arguments***
- ***command*** is the CLI command which does a specific task
- ***flags*** are options we give to the ***command*** to trigger certain behaviors, preceded by a **–**
- ***arguments*** can be what the ***command*** is going to modify, or other options for the ***command***
- Depending on the ***command***, there can be zero or more ***flags*** and ***arguments***
- For example `pwd` is a ***command*** that requires no ***flags*** or ***arguments***

CLI Commands

- `pwd` displays the path to the current working directory

```
jeff$ pwd
/Users/jeff
jeff$
```

CLI Commands

- `clear` will clear out the commands in your current CLI window

```
jeff$ pwd
/Users/jeff
jeff$ clear
```

```
jeff$
```

CLI Commands

- `ls` lists files and folders in the current directory
- `ls -a` lists hidden and unhidden files and folders
- `ls -al` lists details for hidden and unhidden files and folders
- Notice that `-a` and `-l` are flags (they're preceded by a `-`)
- They can be combined into the flag: `-al`

```
jeff$ ls
Desktop   Photos   Music
jeff$ ls -a
Desktop   Photos   Music  .Trash  .DS_Store
jeff$
```

CLI Commands

- `cd` stands for "change directory"
- `cd` takes as an argument the directory you want to visit
- `cd` with no argument takes you to your home directory
- `cd ..` allows you to chnage directory to one level above your current directory

```
jeff$ cd Music/Debussy
jeff$ pwd
/Users/jeff/Music/Debussy
jeff$ cd ..
jeff$ pwd
/Users/jeff/Music
jeff$ cd
jeff$ pwd
/Users/jeff
jeff$
```

CLI Commands

- `mkdir` stands for "make directory"
- Just like: right click -> create new folder
- `mkdir` takes as an argument the name of the directory you're creating

```
jeff$ mkdir Documents
jeff$ ls
Desktop   Photos  Music   Documents
jeff$ cd Documents
jeff$ pwd
/Users/jeff/Documents
jeff$ cd
jeff$
```

CLI Commands

- `touch` creates an empty file

```
jeff$ touch test_file
jeff$ ls
Desktop   Photos  Music   Documents   test_file
jeff$
```

CLI Commands

- `cp` stands for "copy"
- `cp` takes as its first argument a file, and as its second argument the path to where you want the file to be copied

```
jeff$ cp test_file Documents
jeff$ cd Documents
jeff$ ls
test_file
jeff$ cd ..
jeff$
```

CLI Commands

- `cp` can also be used for copying the contents of directories, but you must use the `-r` flag
- The line: `cp -r Documents More_docs` copies the contents of `Documents` into `More_docs`

```
jeff$ mkdir More_docs
jeff$ cp -r Documents More_docs
jeff$ cd More_docs
jeff$ ls
test_file
jeff$ cd ..
jeff$
```

CLI Commands

- `rm` stands for "remove"
- `rm` takes the name of a file you wish to remove as its argument

```
jeff$ ls
Desktop   Photos  Music   Documents   More_docs   test_file
jeff$ rm test_file
jeff$ ls
Desktop   Photos  Music   Documents   More_docs
jeff$
```

CLI Commands

- You can also use `rm` to delete entire directories and their contents by using the `-r` flag
- **Be very careful when you do this, there is no was to undo an `rm`**

```
jeff$ ls
Desktop   Photos  Music   Documents   More_docs
jeff$ rm -r More_docs
jeff$ ls
Desktop   Photos  Music   Documents
jeff$
```

CLI Commands

- `mv` stands for "move"
- With `mv` you can move files between directories

```
jeff$ touch new_file
jeff$ mv new_file Documents
jeff$ ls
Desktop   Photos  Music   Documents
jeff$ cd Documents
jeff$ ls
test_file   new_file
jeff$
```

CLI Commands

- You can also use `mv` to rename files

```
jeff$ ls
test_file   new_file
jeff$ mv new_file renamed_file
jeff$ ls
test_file renamed_file
jeff$
```

CLI Commands

- `echo` will print whatever arguments you provide

```
jeff$ echo Hello World!
Hello World!
jeff$
```

CLI Commands

- `date` will print today's date

```
jeff$ date
Mon Nov  4 20:48:03 EST 2013
jeff$
```

Summary of Commands

- `pwd`
- `clear`
- `ls`
- `cd`
- `mkdir`
- `touch`
- `cp`
- `rm`
- `mv`
- `date`
- `echo`

Introduction to Git

Jeffrey Leek
Johns Hopkins Bloomberg School of Public Health

What is Version Control?

> “Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.”

http://git-scm.com/book/en/Getting-Started-About-Version-Control

What is Version Control?

- Many of us constantly create something, save it, change it, then save it again
- Version (or revision) control is a means of managing this process in a reliable and efficient way
- Especially important when collaborating with others

http://en.wikipedia.org/wiki/Revision_control

What is Git?

> “Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.”

http://git-scm.com/

What is Git?

- Created by the same people who developed Linux
- The most popular implementation of version control today
- Everything is stored in local repositories on your computer
- Operated from the command line

http://git-scm.com/book/en/Getting-Started-A-Short-History-of-Git

Download Git

- Go to the following website and click on the download link for your operating system (Mac, Windows, Linux, etc):

http://git-scm.com/downloads

Install Git

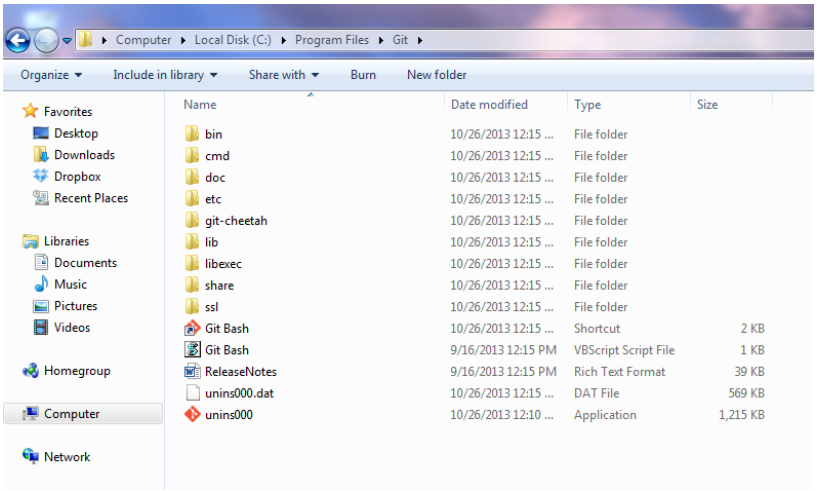
- Once the file is done downloading, open it up to begin the Git installation

Install Git

- Unless you really know what you are doing, just go with the default options at each step of the installation
- Once the install is complete, hit the "Finish" button (you may want to uncheck the box next to "Review ReleaseNotes.rtf")

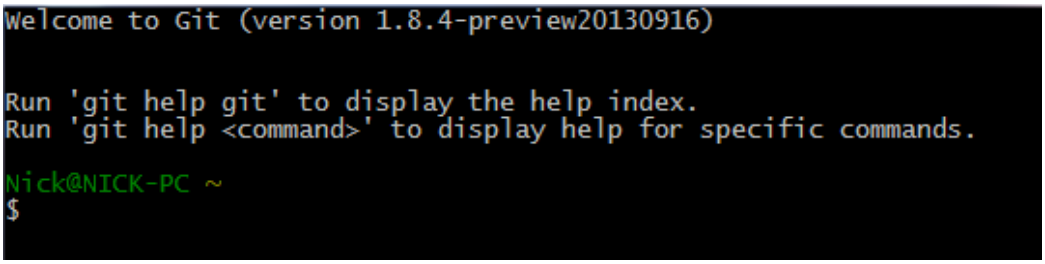
Open Git Bash

- Find a program called Git Bash, which is the command line environment for interacting with Git
- It should be located in the directory into which Git was installed (or, for Windows users, in the Start Menu)



Open Git Bash

- Once Git Bash opens, you'll see a short welcome message followed by the name of your computer and a dollar sign on the next line
- The dollar sign means that it's your turn to type a command



Configure Username and Email

- Each commit to a Git repository will be "tagged" with the username of the person who made the commit
- Enter the following commands in Git Bash, one at a time, to set your username and email:

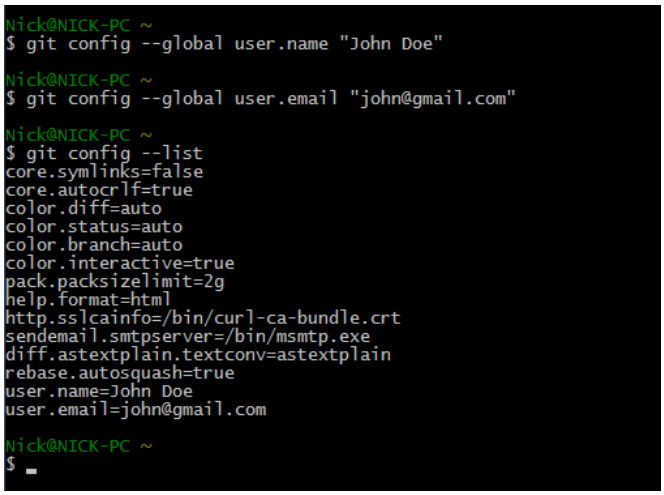
```
$ git config --global user.name "Your Name Here"
$ git config --global user.email "your_email@example.com"
```

- You'll only have to do this once, but you can always change these down the road using the same commands

Configure Username and Email

- Now type the following to confirm your changes (they may be listed toward the bottom):

```
$ git config --list
```



What's Next?

- Go ahead and close Git Bash with following command:

```
$ exit
```

- Now that Git is set up on your computer, we're ready to move on to GitHub, which is a web-based platform that lets you do some pretty cool stuff
- Once GitHub is up and running, we'll show you how to start using these tools to your benefit

Introduction to GitHub

Jeffrey Leek, Assistant Professor of Biostatistics
Johns Hopkins Bloomberg School of Public Health

What is GitHub?

> “GitHub is a web-based hosting service for software development projects that use the Git revision control system.”

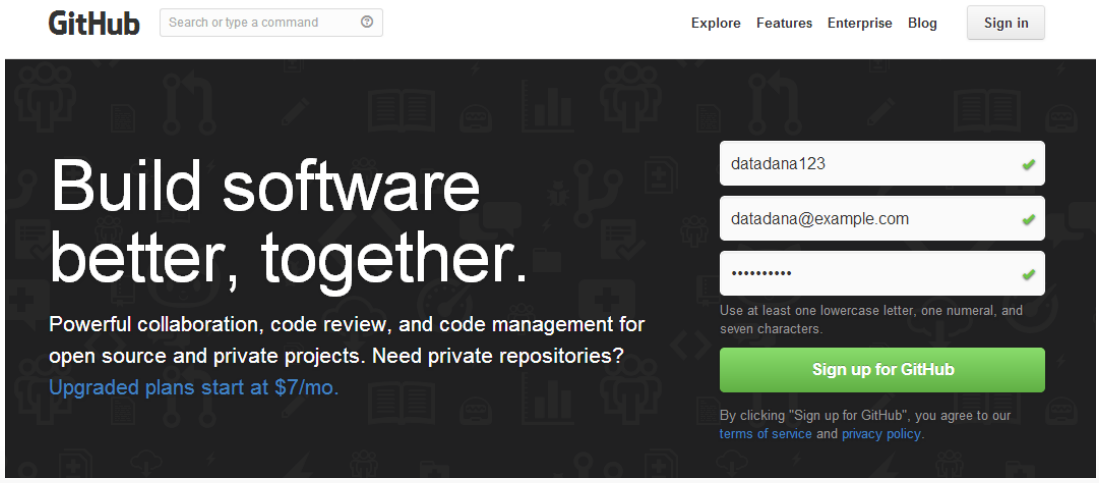
http://en.wikipedia.org/wiki/GitHub

What is GitHub?

- Allows users to "push" and "pull" their local repositories to and from remote repositories on the web
- Provides users with a homepage that displays their public repositories
- Users' repositories are backed up on the GitHub server in case something happens to the local copies
- Social aspect allows users to follow one another and share projects

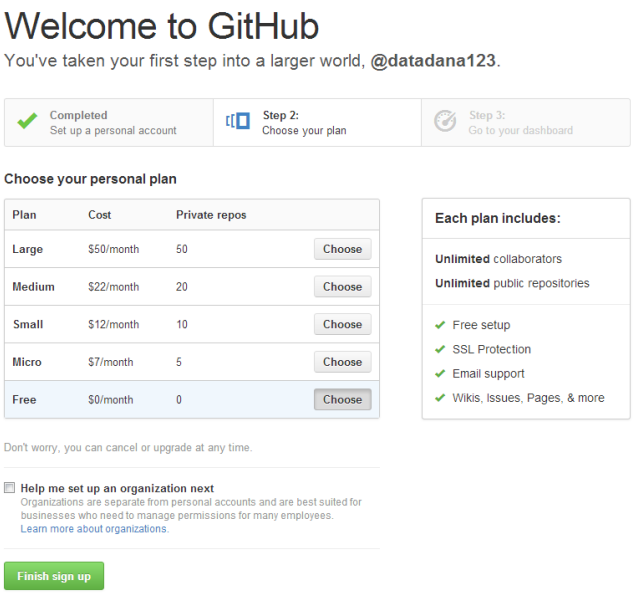
Set Up a GitHub Account

- Go to the GitHub homepage at https://github.com/
- Enter a username, email, and password and click "Sign up for GitHub"
- **NOTE: You should use the same email address that you used when setting up Git in the previous lecture**



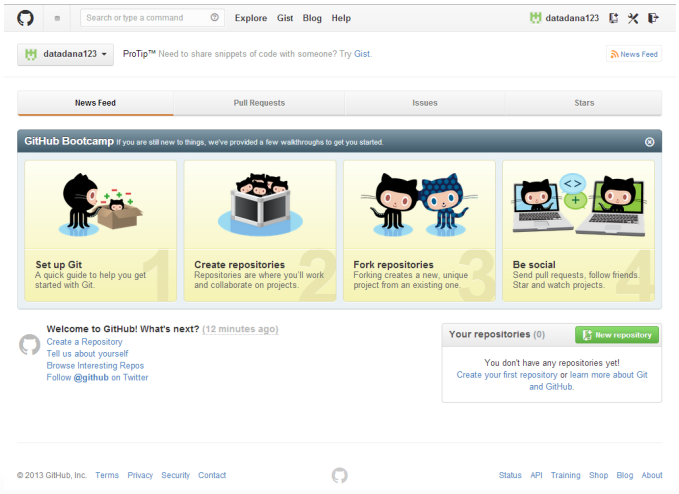
Set Up a GitHub Account

- On the next screen, select the free plan and click "Finish sign up"



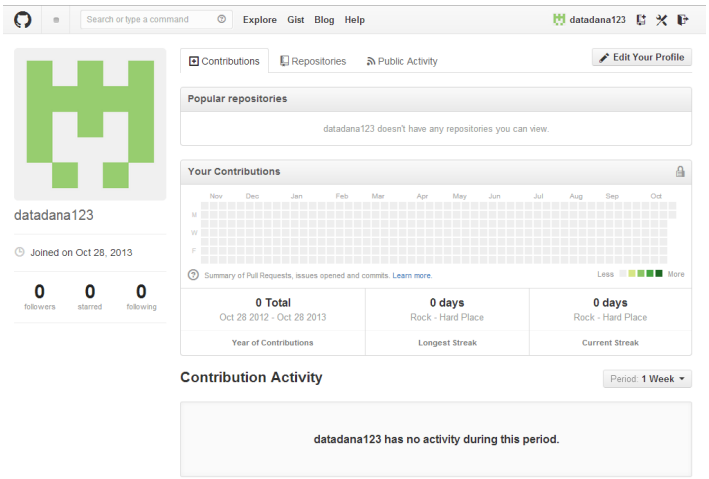
Navigating GitHub

- After signing up, you will find yourself on this page, which has several helpful resources for learning more about Git and GitHub
- Try clicking on your username in the upper righthand corner of the screen to view your GitHub profile



Your GitHub Profile

- Your profile is where all of your activity on GitHub is displayed
- Allows you to show other people who you are and what you are working on
- As you work on more and more projects, your profile becomes a portfolio of your work



Your GitHub Profile

- Finally, if you click on "Edit Your Profile" in the top righthand portion of the screen you can add some basic information about yourself to your profile
- This is totally optional, but if you do good work, you ought to take some credit for it!
- In the next lecture, we'll get you started by walking you through two ways of creating a repository
- In the meantime, feel free to explore the GitHub site for interesting projects that others are working on

Creating a GitHub Repository

Jeffrey Leek, Assistant Professor of Biostatistics
Johns Hopkins Bloomberg School of Public Health

Recap: Git vs. GitHub

- You don't need GitHub to use Git
- Git = Local (on your computer); GitHub = Remote (on the web)
- GitHub allows you to:
  1. Share your repositories with others
  2. Access other users' repositories
  3. Store remote copies of your repositories (on GitHub's server) in case something happens to your local copies (on your computer)

Creating a GitHub Repository

- Two methods of creating a GitHub repository:
  1. Start a repository from scratch
  2. "Fork" another user's repository
- We'll start with the first method
- *NOTE: A repository is often referred to as a "repo"*

Start a Repository From Scratch

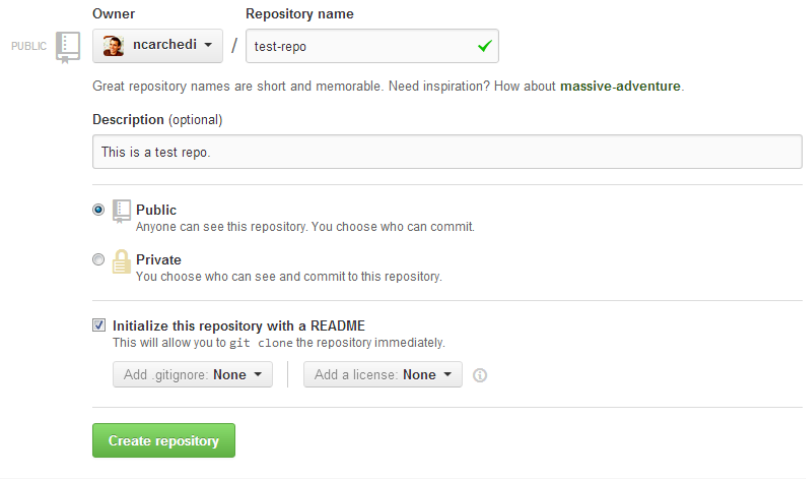
- Either go to your profile page (https://github.com/yourUserNameHere/) and click on "Create a new repo" in the upper righthand corner of the page

...OR...

- Go directly to https://github.com/new (you'll need to log into your GitHub account if you haven't already done so)

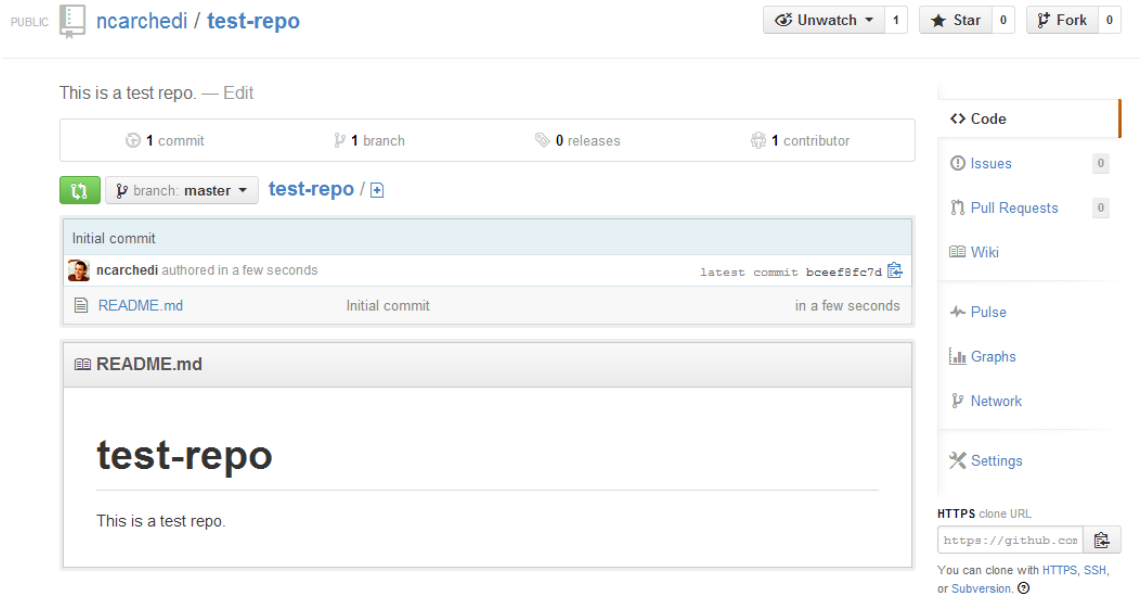
Start a Repository From Scratch

- Create a name for your repo and type a brief description of it
- Select "Public" (Private repos require a paid [or education] account)
- Check the box next to "Initialize this repository with a README"
- Click the "Create repository" button



Start a Repository From Scratch

- Congratulations! You've created a GitHub repository.



Creating a Local Copy

- Now you need to create a copy of this repo on your computer so that you can make changes to it
- Open Git Bash
- Create a directory on your computer where you will store your copy of the repo:

```
$ mkdir ~/test-repo
```

- Navigate to this new directory using the following command:

```
$ cd ~/test-repo
```

Creating a Local Copy

- Initialize a local Git repository in this directory

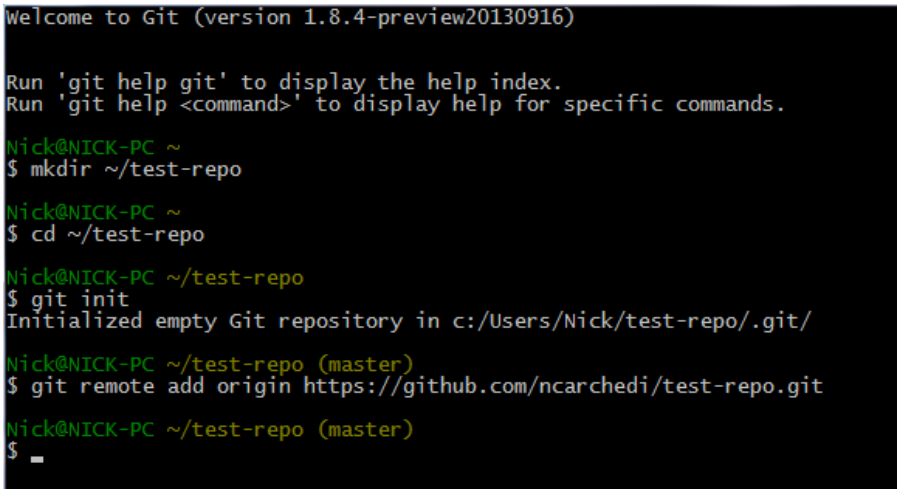
```
$ git init
```

- Point your local repository at the remote repository you just created on the GitHub server

```
$ git remote add origin https://github.com/yourUserNameHere/test-repo.git
```

Creating a Local Copy

- Here's what this process looks like in action:



Fork a Another User's Repository

- The second method of creating a respository is to make a copy of someone else's
- This process is called "forking" and is an important aspect of open-source software development
- Begin by navigating to the desired repository on the GitHub website and click the "Fork" button shown below

https://help.github.com/articles/fork-a-repo

Clone the Repo

- You now have a copy of the desired respository on your GitHub account
- Need to make a local copy of the repo on your computer
- This process is called "cloning" and can be done using the following command:

```
$ git clone https://github.com/yourUserNameHere/repoNameHere.git
```

- *NOTE: This will clone the repository into your current directory.*

What Else?

- If you make changes to your local copy of the repo, you'll probably want to push your changes to GitHub at some point
- You also may be interested in staying current with any changes made to the original repository from which you forked your copy
- We will cover some more Git/GitHub basics in coming lectures, but in the meantime, here are some great resources:
  - https://help.github.com/articles/fork-a-repo
  - http://git-scm.com/book/en/Git-Basics-Getting-a-Git-Repository

Basic markdown

Jeffrey Leek, Assistant Professor of Biostatistics
Johns Hopkins Bloomberg School of Public Health

Markdown Syntax

Headings

```
## This is a secondary heading
### This is a tertiary heading
```

This is a secondary heading

This is a tertiary heading

Markdown Syntax

Unordered Lists

```
* first item in list
* second item in list
* third item in list
```

- first item in list
- second item in list
- third item in list

Getting markdown help

- An introduction to markdown http://daringfireball.net/projects/markdown/
- Click the MD button in Rstudio for a quick guide
- R markdown http://www.rstudio.com/ide/docs/authoring/using_markdown (you don't need this until Reproducible Research)

Installing R Packages

Jeffrey Leek
Johns Hopkins Bloomberg School of Public Health

R Packages

- When you download R from the Comprehensive R Archive Network (CRAN), you get that ``base'' R system
- The base R system comes with basic functionality; implements the R language
- One reason R is so useful is the large collection of packages that extend the basic functionality of R
- R packages are developed and published by the larger R community

Obtaining R Packages

- The primary location for obtaining R packages is CRAN
- For biological applications, many packages are available from the Bioconductor Project
- You can obtain information about the available packages on CRAN with the `available.packages()` function

```
a <- available.packages()
head(rownames(a), 3)  ## Show the names of the first few packages
```

```
## [1] "A3"       "abc"      "abcdeFBA"
```

- There are approximately 5200 packages on CRAN covering a wide range of topics
- A list of some topics is available through the Task Views link, which groups together many R packages related to a given topic

Installing an R Package

- Packages can be installed with the `install.packages()` function in R
- To install a single package, pass the name of the lecture to the `install.packages()` function as the first argument
- The following the code installs the **slidify** package from CRAN

```
install.packages("slidify")
```

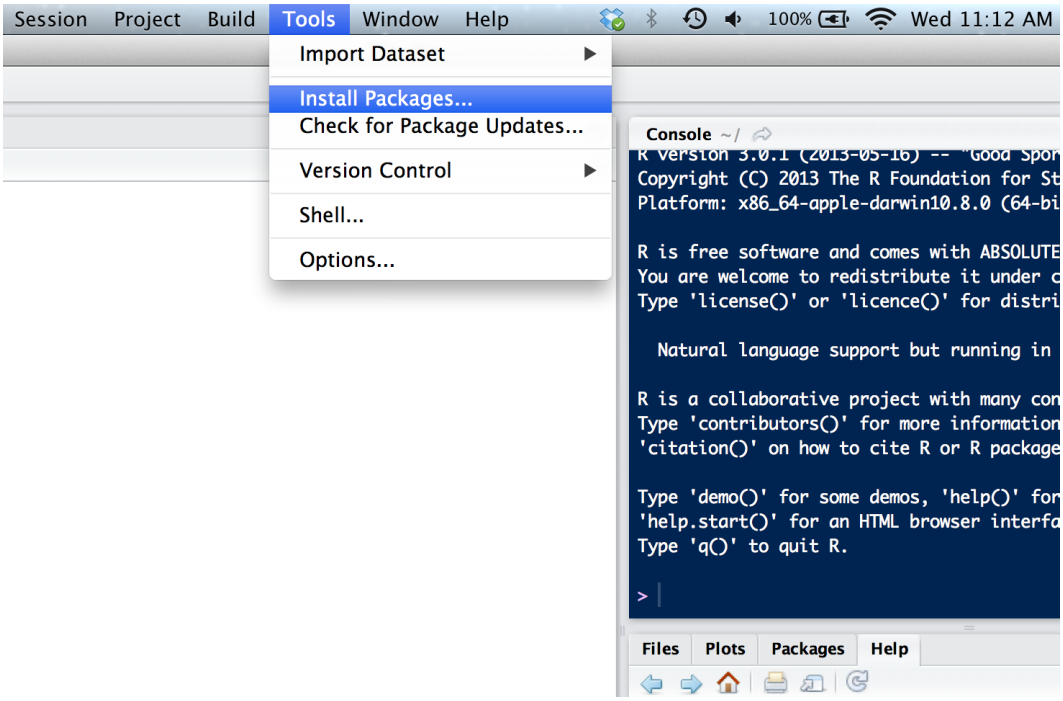
- This command downloads the **slidify** package from CRAN and installs it on your computer
- Any packages on which this package depends will also be downloaded and installed

Installing an R Package

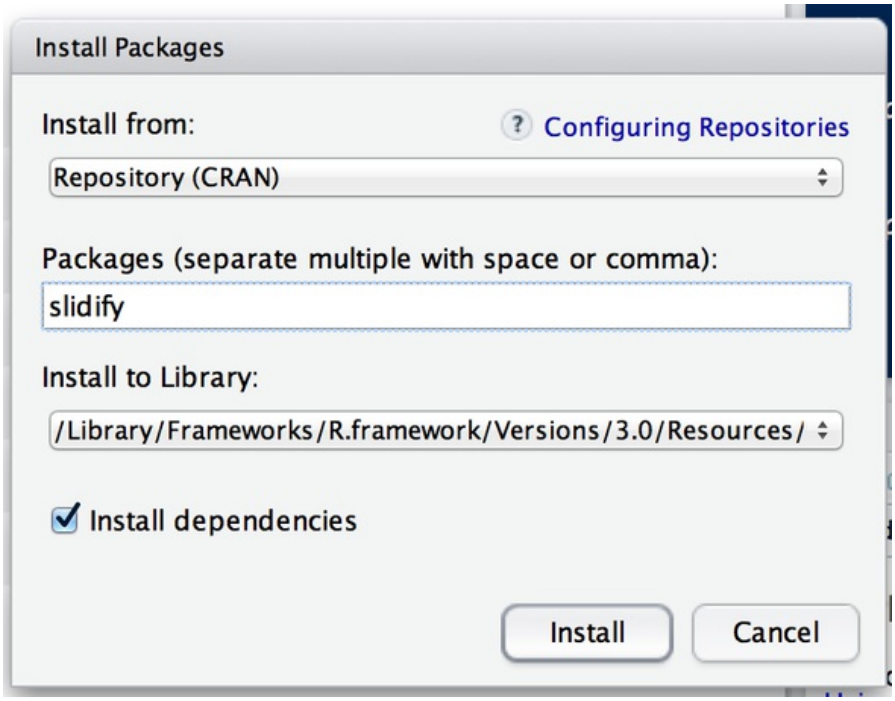
- You can install multiple R packages at once with a single call to `install.packages()`
- Place the names of the R packages in a character vector

```
install.packages(c("slidify", "ggplot2", "devtools"))
```

Installing an R Package in RStudio



Installing an R Package in RStudio



Installing an R Package from Bioconductor

- To get the basic installer and basic set of R packages (warning, will install multiple packages)

```
source("http://bioconductor.org/biocLite.R")
biocLite()
```

- Place the names of the R packages in a character vector

```
biocLite(c("GenomicFeatures", "AnnotationDbi"))
```

http://www.bioconductor.org/install/

Loading R Packages

- Installing a package does not make it immediately available to you in R; you must load the package
- The `library()` function is used to **load** packages into R
- The following code is used to load the **ggplot2** package into R

```
library(ggplot2)
```

- Any packages that need to be loaded as dependencies will be loaded first, before the named package is loaded
- NOTE: Do not put the package name in quotes!
- Some packages produce messages when they are loaded (but some don't)

Loading R Packages

After loading a package, the functions exported by that package will be attached to the top of the `search()` list (after the workspace)

```
library(ggplot2)
search()
```

```
##  [1] ".GlobalEnv"        "package:kernlab"   "package:caret"
##  [4] "package:lattice"   "package:ggplot2"   "package:makeslides"
##  [7] "package:knitr"     "package:slidify"   "tools:rstudio"
## [10] "package:stats"     "package:graphics"  "package:grDevices"
## [13] "package:utils"     "package:datasets"  "package:methods"
## [16] "Autoloads"         "package:base"
```

Summary

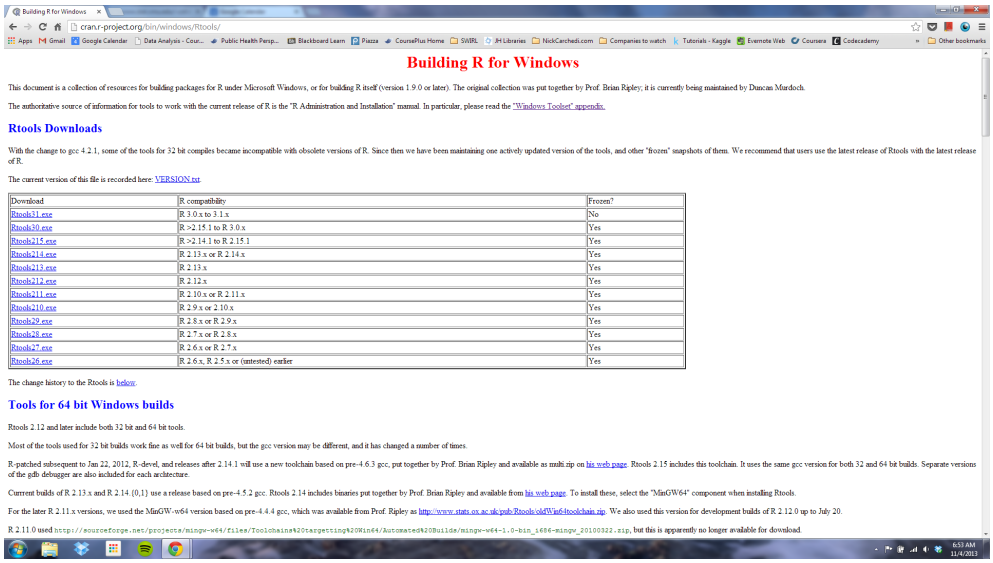
- R packages provide a powerful mechanism for extending the functionality of R
- R packages can be obtained from CRAN or other repositories
- The `install.packages()` can be used to install packages at the R console
- The `library()` function loads packages that have been installed so that you may access the functionality in the package

Installing Rtools

Jeffrey Leek, Assistant Professor of Biostatistics
Johns Hopkins Bloomberg School of Public Health

What is Rtools?

- A collection of tools necessary for building R packages in Windows
- Available for download at http://cran.r-project.org/bin/windows/Rtools/



Building R for Windows

cran.r-project.org/bin/windows/Rtools/

Building R for Windows

This document is a collection of resources for building packages for R under Microsoft Windows, or for building R itself (version 1.9.0 or later). The original collection was put together by Prof. Brian Ripley; it is currently being maintained by Duncan Murdoch.

The authoritative source of information for tools to work with the current release of R is the "R Administration and Installation" manual. In particular, please read the "Windows Toolset" appendix.

Rtools Downloads

With the change to gcc 4.2.1, some of the tools for 32 bit compiles became incompatible with obsolete versions of R. Since then we have been maintaining one actively updated version of the tools, and other "frozen" snapshots of them. We recommend that users use the latest release of Rtools with the latest release of R.

The current version of this file is recorded here: VERSION.txt.

Download	R compatibility	Frozen?
Rtools31.exe	R 3.0.x to 3.1.x	No
Rtools30.exe	R >2.15.1 to R 3.0.x	Yes
Rtools215.exe	R >2.14.1 to R 2.15.1	Yes
Rtools214.exe	R 2.13.x or R 2.14.x	Yes
Rtools213.exe	R 2.13.x	Yes
Rtools212.exe	R 2.12.x	Yes
Rtools211.exe	R 2.10.x or R 2.11.x	Yes
Rtools210.exe	R 2.9.x or 2.10.x	Yes
Rtools29.exe	R 2.8.x or R 2.9.x	Yes
Rtools28.exe	R 2.7.x or R 2.8.x	Yes
Rtools27.exe	R 2.6.x or R 2.7.x	Yes
Rtools26.exe	R 2.6.x, R 2.5.x or (untested) earlier	Yes

The change history to the Rtools is below.

Tools for 64 bit Windows builds

Rtools 2.12 and later include both 32 bit and 64 bit tools.

Most of the tools used for 32 bit builds work fine as well for 64 bit builds, but the gcc version may be different, and it has changed a number of times.

R-patched subsequent to Jan 22, 2012, R-devel, and releases after 2.14.1 will use a new toolchain based on pre-4.6.3 gcc, put together by Prof. Brian Ripley and available as multi.zip on his web page. Rtools 2.15 includes this toolchain. It uses the same gcc version for both 32 and 64 bit builds. Separate versions of the gdb debugger are also included for each architecture.

Current builds of R 2.13.x and R 2.14.{0,1} use a release based on pre-4.5.2 gcc. Rtools 2.14 includes binaries put together by Prof. Brian Ripley and available from his web page. To install these, select the "MinGW64" component when installing Rtools.

For the later R 2.11.x versions, we used the MinGW-w64 version based on pre-4.4.4 gcc, which was available from Prof. Ripley as http://www.stats.ox.ac.uk/pub/Rtools/oldWin64toolchain.zip. We also used this version for development builds of R 2.12.0 up to July 20.

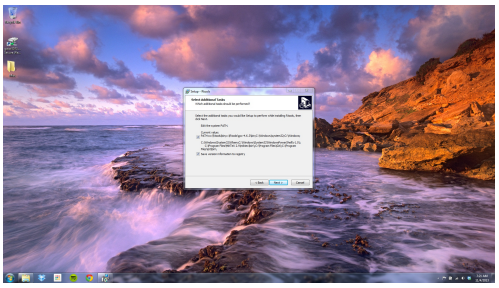
R 2.11.0 used http://sourceforge.net/projects/mingw-w64/files/Toolchains%20targetting%20Win64/Automated%20Builds/mingw-w64-1.0-bin_i686-mingw_20100322.zip, but this is apparently no longer available for download.

Download Rtools

- Select the .exe download link from the table that corresponds to your version of R
  - Note: If you're not sure what version of R you have, open or restart R and it's the first thing that comes up in the console
- If you have the most recent version of R, you should select the most recent Rtools download (at the top of the chart)
- Once the download completes, open the .exe file to begin the installation

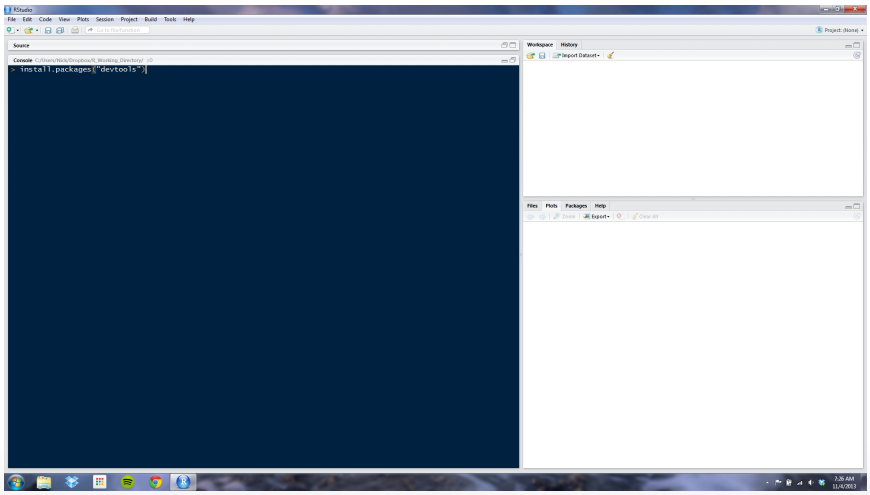
Install Rtools

- Unless you really know what you are doing, you should just go with the default selections at each step of the installation
- There are only two exceptions worth noting:
  - If you already have Cygwin installed on your machine, you should follow the instructions given during installation (and linked to here: http://cran.r-project.org/bin/windows/Rtools/Rtools.txt)
  - *IMPORTANT: You should make sure that the box is checked to have the installer edit your PATH (see below).*



Install devtools

- Once the Rtools installation completes, open RStudio
- Install the devtools R package if you have not previously done so
  - If you aren't sure, enter `find.package("devtools")` in the console
- To install devtools, use `install.packages("devtools")`



Verify Rtools installation

- After devtools is done installing, load it using `library(devtools)`
- Then type `find_rtools()` as shown below
- This should return `TRUE` in the console if your Rtools installation worked properly