



# BIMM 143

## Find a Gene Assignment

Lecture 10

Barry Grant  
UC San Diego

<http://thegrantlab.org/bimm143>

# Find-a-Gene Project Assignment

- A total of 20% of the course grade will be assigned based on the “[find-a-gene project assignment](#)”
- The objective with this assignment is for you to demonstrate your grasp of database searching, sequence analysis, structure analysis and the R environment that we have covered to date in class.
- You may wish to consult the scoring rubric (in the linked project description) and the [example report](#) for format and content guidance.
  - ➔ Your responses to questions Q1-Q4 are due at the beginning of class Thursday February 22nd (02/22/18).
  - ➔ The complete assignment, including responses to all questions, is due at the beginning of class Thursday March 13th (03/13/18).

### Questions:

[Q1] Tell me the name of a protein you are interested in. Include the species and the accession number. This can be a human protein or a protein from any other species as long as its function is known.

If you do not have a favorite protein, select human RBP4 or KIF11. Do not use beta globin as this is in the worked example report that I provide you with online.

[Q2] Perform a BLAST search against a DNA database, such as a database consisting of genomic DNA or ESTs. The BLAST server can be at NCBI or elsewhere. Include details of the BLAST method used, database searched and any limits applied (e.g. Organism).

Also include the output of that BLAST search in your document. If appropriate, change the font to Courier size 10 so that the results are displayed neatly. You can also screen capture a BLAST output (e.g. alt print screen on a PC or on a MAC press ⌘-shift-4. The pointer becomes a bulls eye. Select the area you wish to capture and release. The image is saved as a file called Screen Shot [ ].png in your Desktop directory). It is not necessary to print out all of the blast results if there are many pages.

On the BLAST results, clearly indicate a match that represents a protein sequence, encoded from some DNA sequence, that is homologous to your query protein. I need to be able to inspect the pairwise alignment you have selected, including the E value and score. It should be labeled a "genomic clone" or "mRNA sequence", etc. - but include no functional annotation.

In general, [Q2] is the most difficult for students because it requires you to have a "feel" for how to interpret BLAST results. You need to distinguish between a perfect match to your query (i.e. a sequence that is not "novel"), a near match (something that might be "novel", depending on the results of [Q4]), and a non-homologous result.

If you are having trouble finding a novel gene try restricting your search to an organism that is poorly annotated.

[Q3] Gather information about this "novel" **protein**. At a minimum, show me the protein sequence of the "novel" protein as displayed in your BLAST results from [Q2] as FASTA format (you can copy and paste the aligned sequence subject lines from your BLAST result page if necessary) or translate your novel DNA sequence using a tool called EMBOSS Transeq at the EBI. Don't forget to translate all six reading frames; the ORF (open reading frame) is likely to be the longest sequence without a stop codon. It may not start with a methionine if you don't have the complete coding region. Make sure the sequence you provide includes a header/subject line and is in traditional FASTA format.

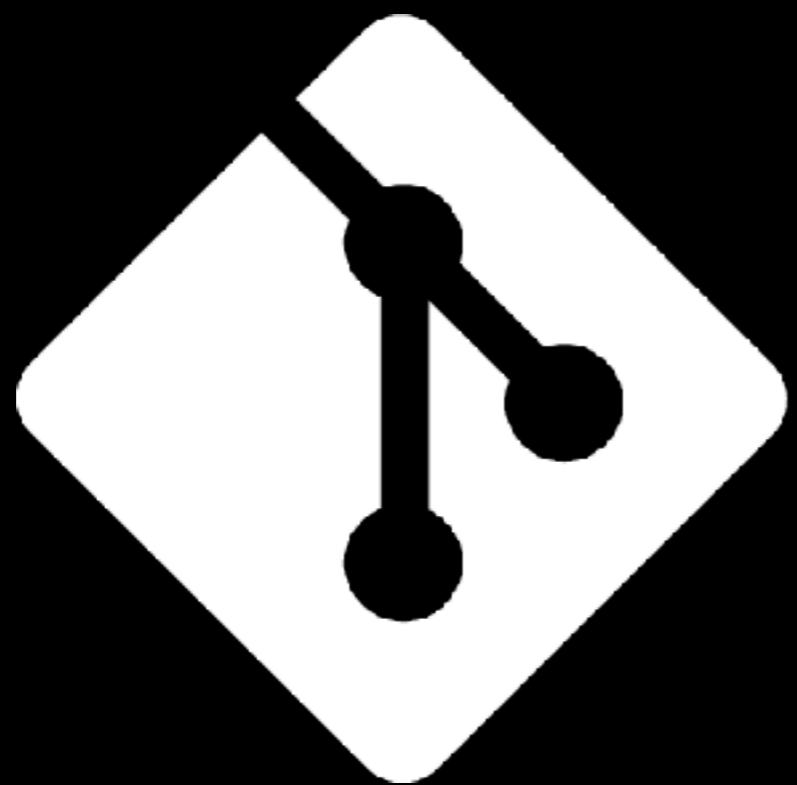
Here, tell me the name of the novel protein, and the species from which it derives. It is very unlikely (but still definitely possible) that you will find a novel gene from an organism such as *S. cerevisiae*, human or mouse, because those genomes have already been thoroughly annotated. It is more likely that you will discover a new gene in a genome that is currently being sequenced, such as bacteria or plants or protozoa.

[Q4] Prove that this gene, and its corresponding protein, are novel. For the purposes of this project, "novel" is defined as follows. Take the protein sequence (your answer to [Q3]), and use it as a query in a blastp search of the nr database at NCBI.

- If there is a match with 100% amino acid identity to a protein in the database, from the same species, then your protein is NOT novel (even if the match is to a protein with a name such as "unknown"). Someone has already found and annotated this sequence, and assigned it an accession number.
- If the top match reported has less than 100% identity, then it is likely that your protein is novel, and you have succeeded.
- If there is a match with 100% identity, but to a different species than the one you started with, then you have likely succeeded in finding a novel gene.
- If there are no database matches to the original query from [Q1], this indicates that you have partially succeeded: yes, you may have found a new gene, but no, it is not actually homologous to the original query. You should probably start over.

[Q5] Generate a multiple sequence alignment with your novel protein, your original query protein, and a group of other members of this family from different species. A typical number of proteins to use in a multiple sequence alignment for this assignment purpose is a minimum of 5 and a maximum of 20 - although the exact number is up to you. Include the multiple sequence alignment in your report. Use Courier font with a size appropriate to fit page width.

Side-note: Indicate your sequence in the alignment by choosing an appropriate name for each sequence in the input unaligned sequence file (i.e. edit the sequence file so that the species, or short common, names (rather than accession numbers) display in the output alignment and in the subsequent answers below). The goal in this step is to create an interesting alignment for building a phylogenetic tree that illustrates species divergence.



git

# What is Git?

(1) An unpleasant or contemptible person. Often incompetent, annoying, senile, elderly or childish in character.



(2) A modern distributed version control system with an emphasis on speed and data integrity.



# What is Git?

(1) An unpleasant or contemptible person. Often incompetent, annoying, senile, elderly or childish in character.



(2) A modern distributed version control system with an emphasis on speed and data integrity.



# Version Control

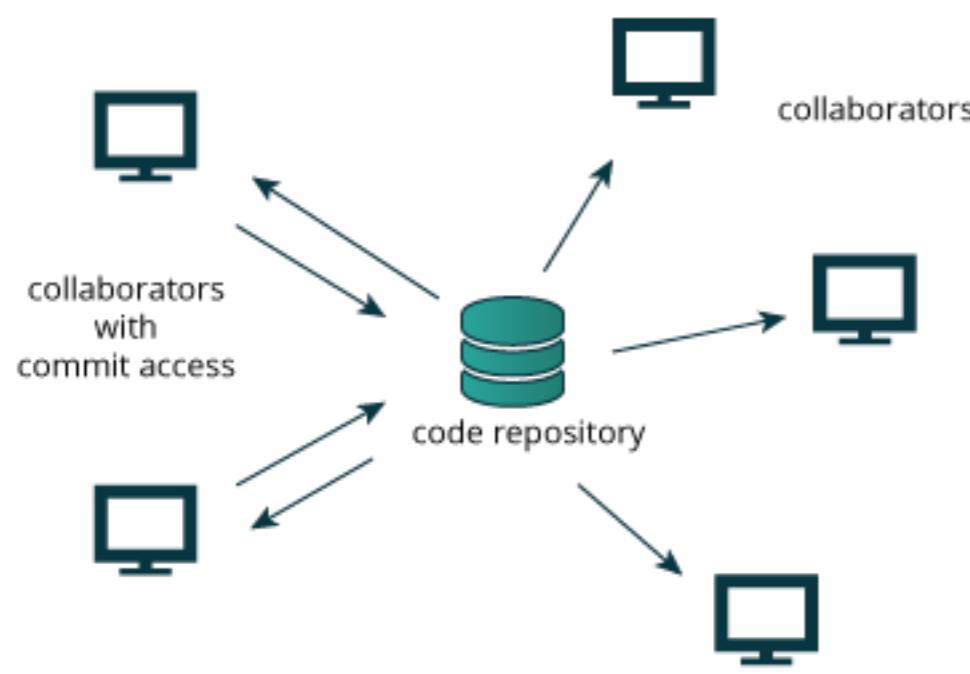
Version control systems (VCS) record changes to a file or set of files over time so that you can recall specific versions later.

Client-server	Free/open-source	<a href="#">CVS</a> (1986, 1990 in C) · <a href="#">CVSNT</a> (1998) · <a href="#">QVCS Enterprise</a> (1998) · <a href="#">Subversion</a> (2000)
	Proprietary	<a href="#">Software Change Manager</a> (1970s) · <a href="#">Panvalet</a> (1970s) · <a href="#">Endevor</a> (1980s) · <a href="#">Dimensions CM</a> (1980s) · <a href="#">DSEE</a> (1984) · <a href="#">Synergy</a> (1990) · <a href="#">ClearCase</a> (1992) · <a href="#">CMVC</a> (1994) · <a href="#">Visual SourceSafe</a> (1994) · <a href="#">Perforce</a> (1995) · <a href="#">StarTeam</a> (1995) · <a href="#">Integrity</a> (2001) · <a href="#">Surround SCM</a> (2002) · <a href="#">AccuRev SCM</a> (2002) · <a href="#">SourceAnywhere</a> (2003) · <a href="#">Vault</a> (2003) · <a href="#">Team Foundation Server</a> (2005) · <a href="#">Team Concert</a> (2008)
Distributed	Free/open-source	<a href="#">GNU arch</a> (2001) · <a href="#">Darcs</a> (2002) · <a href="#">DCVS</a> (2002) · <a href="#">ArX</a> (2003) · <a href="#">Monotone</a> (2003) · <a href="#">SVK</a> (2003) · <a href="#">Codeville</a> (2005) · <a href="#">Bazaar</a> (2005) · <a href="#">Git</a> (2005) · <a href="#">Mercurial</a> (2005) · <a href="#">Fossil</a> (2007) · <a href="#">Veracity</a> (2010)
	Proprietary	<a href="#">TeamWare</a> (1990s?) · <a href="#">Code Co-op</a> (1997) · <a href="#">BitKeeper</a> (1998) · <a href="#">Plastic SCM</a> (2006)

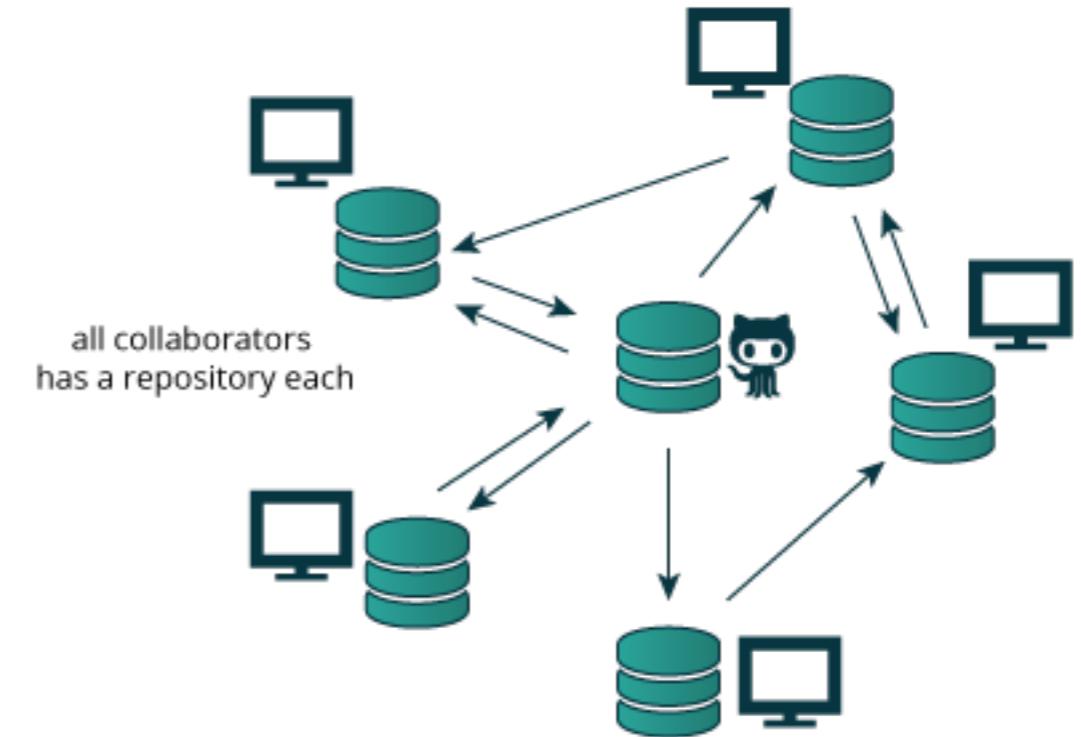
There are many VCS available, see:

[https://en.wikipedia.org/wiki/Revision\\_control](https://en.wikipedia.org/wiki/Revision_control)

# Client-Server vs Distributed VCS



**Client-server approach**



**Distributed approach**

Distributed version control systems (DCVS) allows multiple people to work on a given project without requiring them to share a common network.

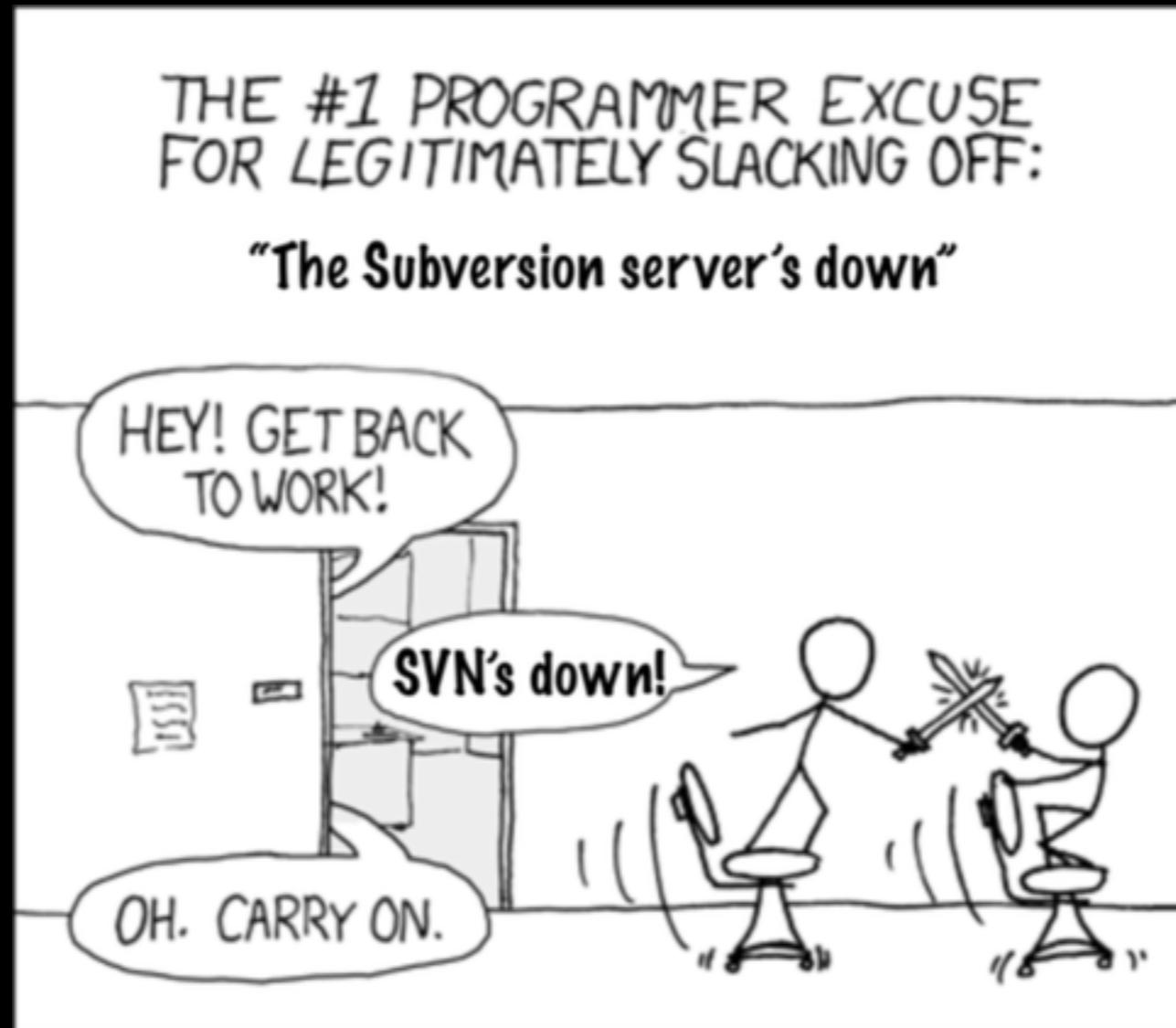
THE #1 PROGRAMMER EXCUSE  
FOR LEGITIMATELY SLACKING OFF:

"The Subversion server's down"



<http://tinyurl.com/distributed-advantages>

# Git is now the most popular free VCS!



- Git offers:**
- Speed
  - Backups
  - Off-line access
  - Small footprint
  - Simplicity\*
  - Social coding

<http://tinyurl.com/distributed-advantages>

# Why use Git?

Q. Would you write your Lab book in pencil, then erase and overwrite it every day with new content?

**Q. Would you write your lab book in pencil, then erase and overwrite it every day with new content?**

Version control is the lab notebook of the digital world: it's what professionals use to keep track of what they've done and to collaborate with others.

# Why use Git?

- Provides ‘**snapshots**’ of your project during development and provides a full record of project **history**.
- Allows you to easily **reproduce** and **rollback** to past versions of analysis and compare differences. (N.B. Helps fix software regression bugs!)
- Keeps **track of changes** to code you use from others such as fixed bugs & new features
- Provides a mechanism for sharing, updating and collaborating (like a social network)
- Helps keep your work and software organized and available

# Obtaining Git

# Obtaining Git

<https://git-scm.com/downloads>

# Configuring Git

# Configuring Git

(**RStudio > Tools > Shell**)

*# First tell Git who you are*

```
> git config --global user.name "Barry Grant"  
> git config --global user.email "bjgrant@ucsd.edu"
```

# Using Git

# Using Git

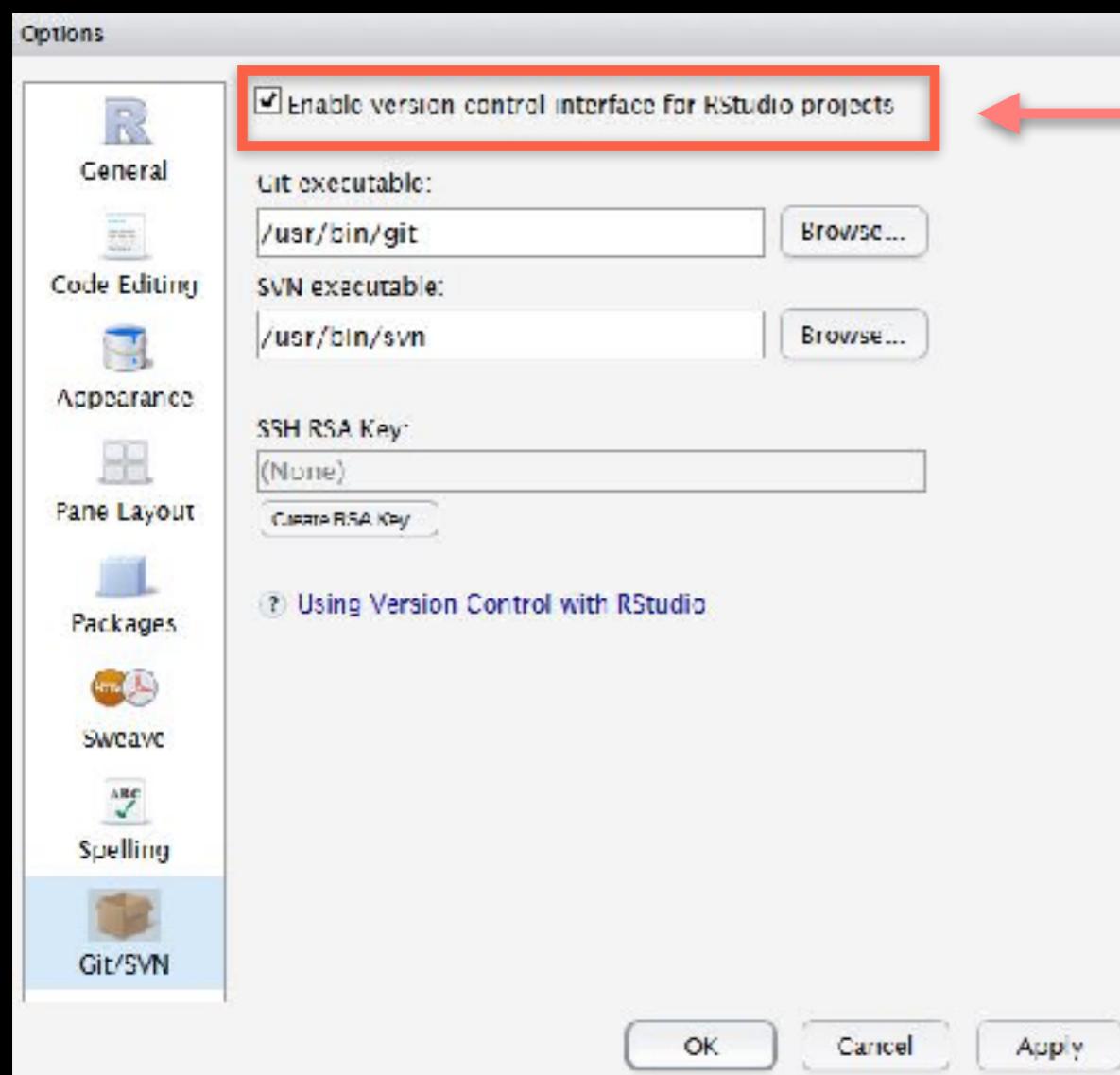
1. Initiate a Git repository.
2. Edit content (i.e. change some files).
3. Store a 'snapshot' of the current file state.\*

# Using Git with RStudio

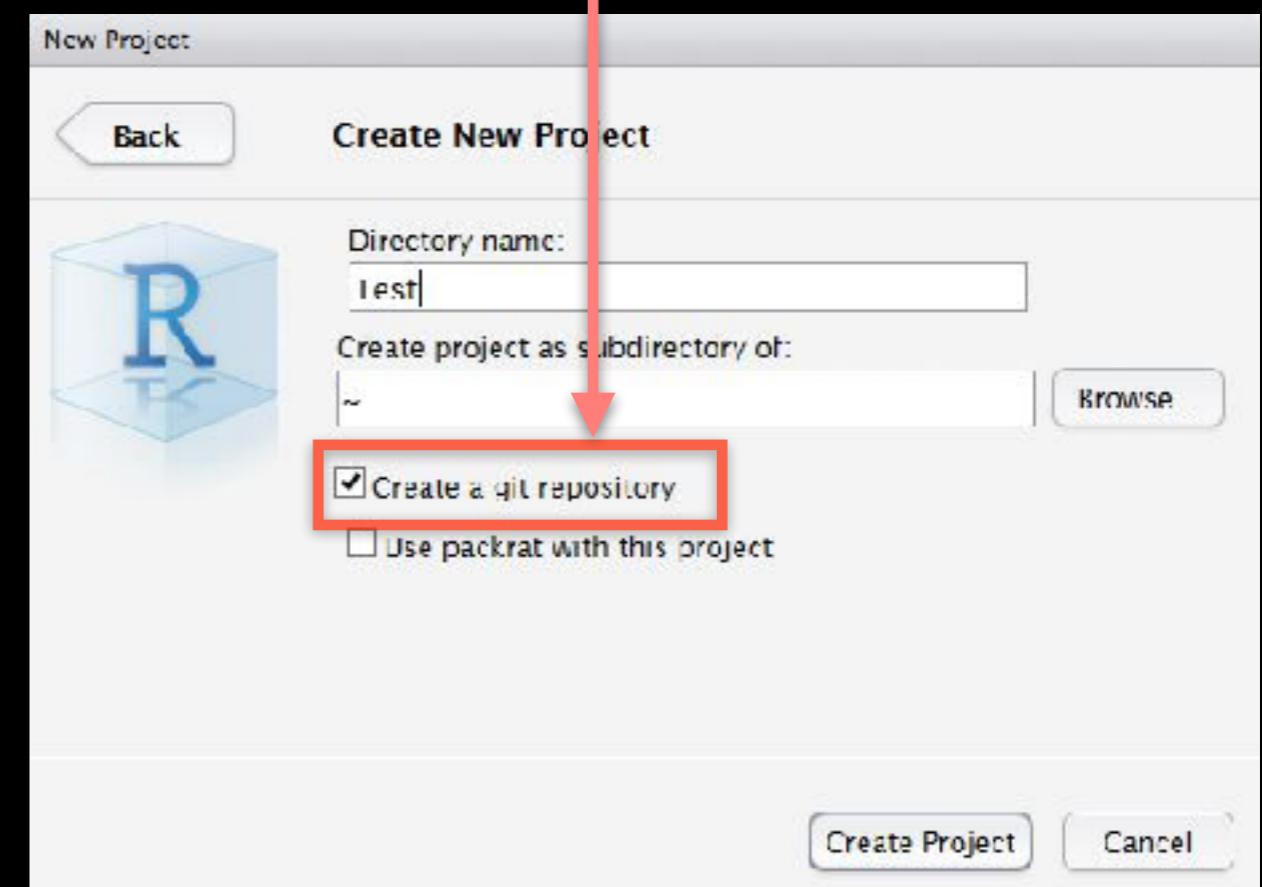
***Two initial steps:***

**1: RStudio > Tools > Global Options > Git/SVN**

**2: File > New Project > New Directory > Empty Project**



Make sure these are ticked!



1

2

# Your Turn:

<http://tinyurl.com/rclass-github>

Step 3-4 only please!

(We have already done steps 1 & 2)

# GitHub & Bitbucket

**GitHub** and **Bitbucket** are two popular hosting services for Git repositories. These services allow you to share your projects and collaborate with others using both '**public**' and '**private**' repositories\*.

The screenshot shows the GitHub dashboard. At the top, there's a 'GitHub Bootcamp' guide with four steps: 1. Set up Git, 2. Create repositories, 3. Fork repositories, and 4. Work together. Below the guide, a message says 'remills pushed to gh-pages at bioboot/web-2015'. On the right, there's a section for 'Your repositories' with a 'New repository' button and a list showing 'web-2015'.

<https://github.com>

The screenshot shows the Bitbucket dashboard. It features a 'Dashboard' section with tabs for Overview, Pull requests, Issues, and Snippets. A sidebar on the left lists repositories: 'Grantleb / bld3d', 'lariss / cheminf', 'Gr...', 'juli...', and 'bjgrant / test'. On the right, there's a 'New to Git?' section with a link to 'Learn more'. Below it, a list of recent activity includes a merge request from 'le33t65' to 'Grantleb/bld3d' and a push from 'XinQiuYao'.

<https://bitbucket.org>

wsj.com

Nikkel 17893.73 0.49% Hang Seng 21404.96 0.72% U.S. 10 Yr -0.32 Yield 2.074% Crude Oil 39.17 -0.36% Yen 119.16 0.26% EXPAND

# THE WALL STREET JOURNAL.

Subscribe Now | Sign In  
\$12 FOR 12 WEEKS

Home World U.S. Politics Economy Business Tech Markets Opinion Arts Life Real Estate

<  Workers Get New Tools for Airing Their Grips  Cell Carriers Battle for Wi-Fi Airwaves  Snapchat Names ex-Mattel Exec Vollero Its Finance Chief >

YOU ARE READING A PREVIEW OF A PAID ARTICLE. [SUBSCRIBE NOW](#) TO GET MORE GREAT CONTENT.

TECH

## GitHub Raises \$250 Million at \$2 Billion Valuation

Capital raise puts company's total funding at \$350 million



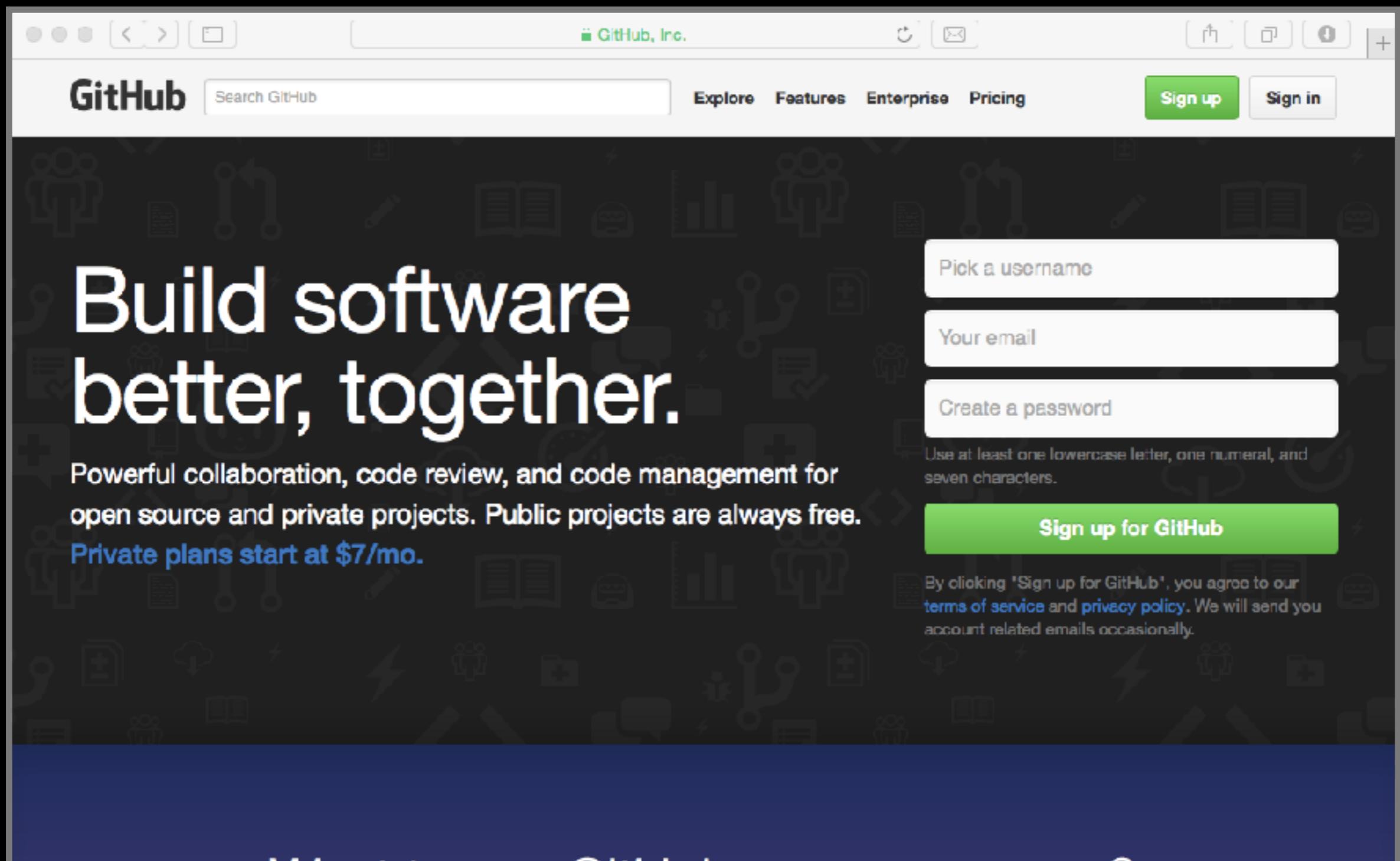
Analytics  
How does your organization's talent measure up to its technology?  
[Read the MIT Sloan report](#)

# What is the big deal?

- At the simplest level GitHub and Bitbucket offer **backup** of your projects history and a centralized mechanism for **sharing** with others by putting **your Git repo online**.
  - GitHub in particular is often referred to as the “nerds FaceBook and LinkedIn combined”.
- At their core both services **offer a new paradigm for open collaborative project development**, particularly for software.
  - In essence they allow anybody to contribute to any public project and get acknowledgment.

# First sign up for a GitHub account

**<https://github.com>**



# Pick the FREE plan!

The screenshot shows the GitHub welcome screen. At the top, there are three steps: 'Completed Set up a personal account', 'Step 2: Choose your plan' (which is highlighted), and 'Step 3: Go to your dashboard'. Below this, a section titled 'Choose your personal plan' displays five plan options:

Plan	Cost	Private repositories	Action
Large	\$50/month	50	<a href="#">Choose</a>
Medium	\$22/month	20	<a href="#">Choose</a>
Small	\$12/month	10	<a href="#">Choose</a>
Micro	\$7/month	5	<a href="#">Choose</a>
Free	\$0/month	0	<a href="#">Chosen</a>

A red circle highlights the 'Chosen' button for the Free plan. To the right, a box lists 'Each plan includes:' features: Unlimited collaborators, Unlimited public repositories, Free setup, HTTPS Protection, Email support, and Wikis, Issues, Pages, & more.

Charges to your account will be made in US Dollars. Converted prices are provided as a convenience and are only an estimate based on current exchange rates. Local prices will change as the exchange rate fluctuates.  
Don't worry, you can cancel or upgrade at any time.

# Your GitHub homepage

Check your email for verification request

The screenshot shows a GitHub user profile for 'biobootStudent'. The profile features a large green 'H' logo composed of squares. The user joined on Aug 28, 2015, and has 0 followers, 0 starred repositories, and 0 repositories followed. A 'Contributions' section displays a grid where each square represents a commit; the grid is mostly empty with a few small squares. A 'Pro tip' message encourages updating the profile with name, location, and a picture. A 'Read the Hello World guide' button is at the bottom.

GitHub, Inc.

Search GitHub

Pull requests Issues Gist

Edit profile

biobootStudent

Joined on Aug 28, 2015

Followers 0   Starred 0   Following 0

Contributions

Pro tip: updating your profile with your name, location, and a profile picture helps other GitHub users get to know you.

Contributions

Less More

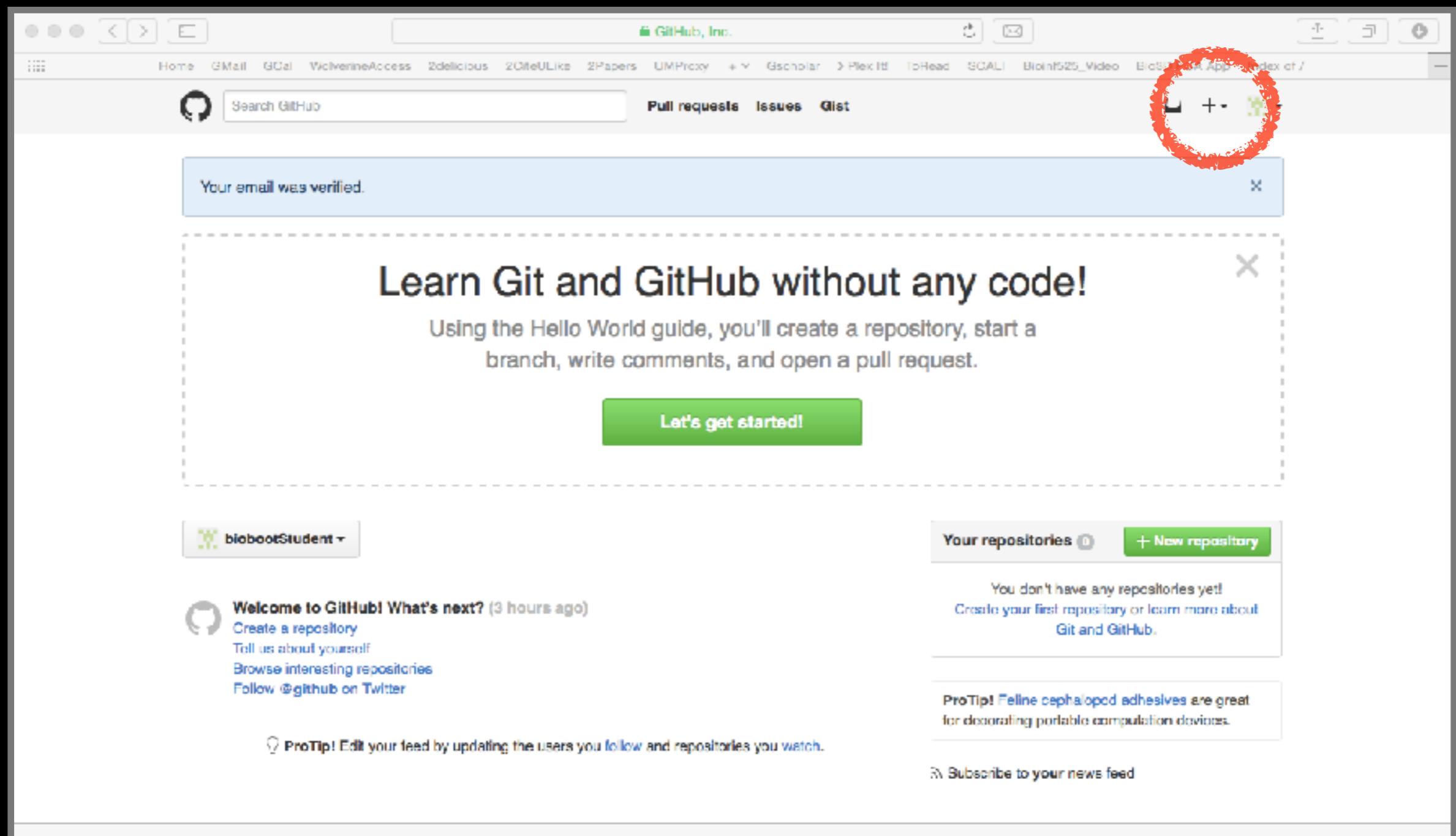
This is your contribution graph. When you make a commit to a repository, you'll get a ■ for that day. Make more contributions and you'll get a darker green square. Over time, your chart might start looking something like this.

We have a quick guide that will show you how to create your first repository. You'll also make a commit and earn your first green square!

Read the Hello World guide

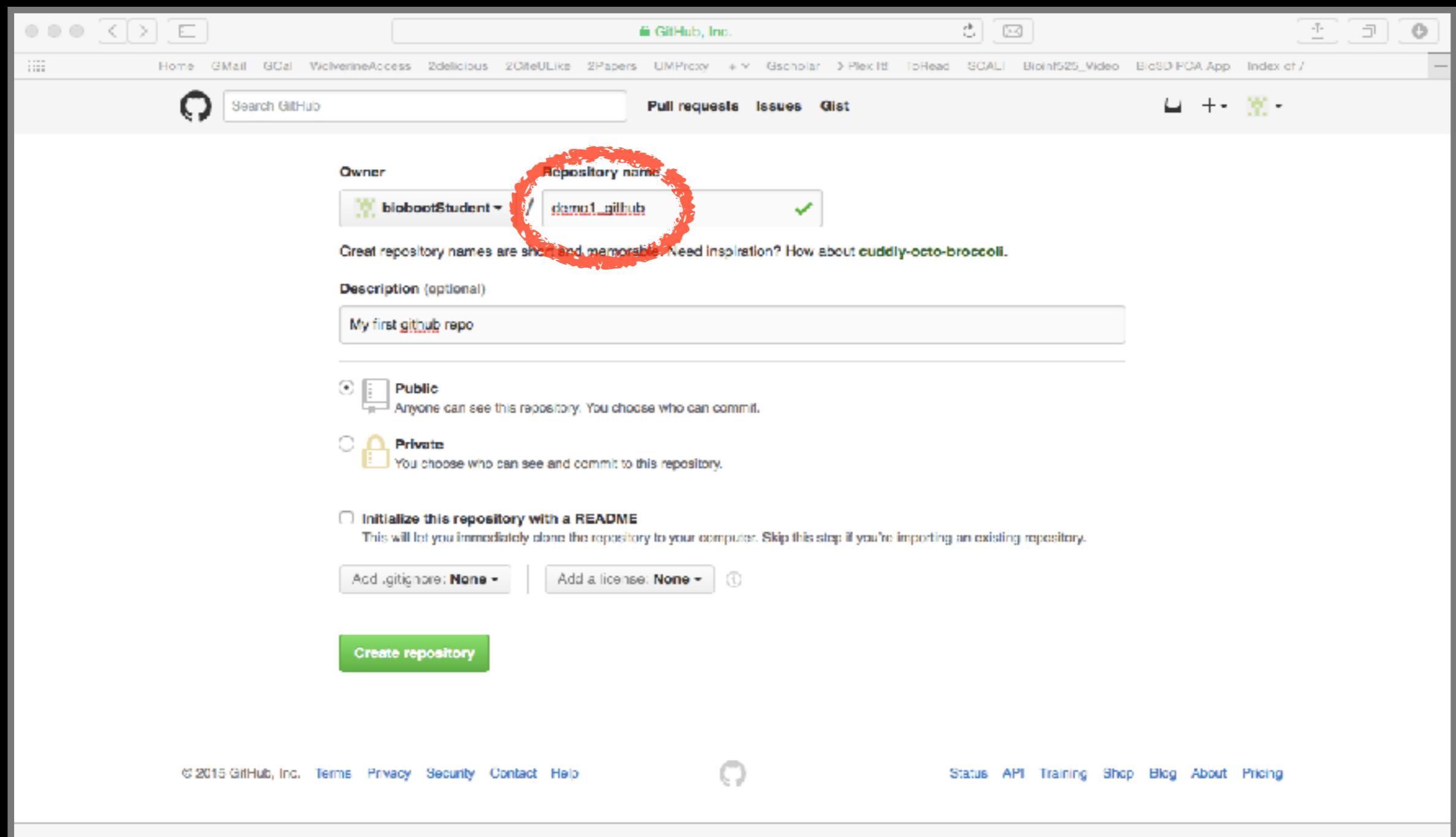
# Skip the hello-world tutorial

<https://guides.github.com/activities/hello-world/>



# Name your repo

## test



A screenshot of a web browser showing the GitHub repository creation interface. The URL in the address bar is `https://github.com/new`. The page title is "Create a new repository". The "Repository name" field contains the text "demo1\_github", which is circled in red. The "Owner" dropdown is set to "blobootStudent". Below the repository name, there is a note: "Great repository names are short and memorable. Need inspiration? How about cuddly-octo-broccoli." The "Description (optional)" field contains the text "My first github repo". Under "Visibility", the "Public" radio button is selected, with the note "Anyone can see this repository. You choose who can commit.". The "Private" radio button is also present with its own note. There is a checkbox for "Initialize this repository with a README", which is unchecked. At the bottom, there are buttons for "Add .gitignore: None" and "Add a license: None", followed by a "Create repository" button.

Owner: blobootStudent / Repository name: demo1\_github

Great repository names are short and memorable. Need inspiration? How about [cuddly-octo-broccoli](#).

Description (optional): My first [github](#) repo

Public Anyone can see this repository. You choose who can commit.

Private You choose who can see and commit to this repository.

Initialize this repository with a README  
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: None Add a license: None

Create repository

© 2015 GitHub, Inc. Terms Privacy Security Contact Help

Status API Training Shop Blog About Pricing

# Your Turn:

<http://tinyurl.com/rclass-github>

Step 6 only please!

(We have already done steps 1 to 5)

# Side-note: How to edit online

Specifically lets add some Markdown content

A screenshot of a web browser showing a GitHub repository page. The URL in the address bar is [https://github.com/biobootStudent/demo1\\_github](https://github.com/biobootStudent/demo1_github). The page displays the `demo1_github / README` file. The content of the README is:

```
1 # My first Git repo is now online
2 This is a first line of text.
3 This is a 2nd line of text.
4
5 I am going to use **markdown** syntax from now on because it is _cool_!
6 I am a student in [bioboot camp](http://bioboot.github.io/web-2015/)
```

The right side of the screen shows a vertical sidebar with various icons. A red circle highlights the edit icon (a pencil symbol) located in the top right corner of the sidebar.

At the bottom of the page, there are links for GitHub footer: © 2015 GitHub, Inc. Terms Privacy Security Contact Help, Status API Training Shop Blog About Pricing.

# Summary

- Git is a popular ‘distributed’ version control system that is lightweight and free
- GitHub and BitBucket are popular hosting services for git repositories that have changed the way people contribute to open source projects
- Introduced basic git and GitHub usage within RStudio and encouraged you to adopt these ‘best practices’ for your future projects.

# Learning Resources

- **Set up Git.** If you will be using Git mostly or entirely via **GitHub**, look at these how-tos.  
< <https://help.github.com/categories/bootcamp/> >
- **Getting Git Right.** Excellent **Bitbucket** git tutorials  
< <https://www.atlassian.com/git/> >
- **Pro Git.** A complete, book-length guide and reference to Git, by Scott Chacon and Ben Straub.  
< <http://git-scm.com/book/en/v2> >
- **StackOverflow.** Excellent programming and developer Q&A.  
< <http://stackoverflow.com/questions/tagged/git> >

# Learning git can be painful!

However in practice it is not nearly as crazy-making as the alternatives:

- Documents as email attachments
- Hair-raising ZIP archives containing file salad
- Am I working with the most recent data?
- Archaeological “digs” on old email threads and uncertainty about how/if certain changes have been made or issues solved

Finally Please remember that **GitHub** and **BitBucket** are **PUBLIC** and that you should cultivate your professional and scholarly profile with intention!

# Side-Note: Changing your default git text editor

- You can configure the default text editor that will be used when Git needs you to type in a message.  
  
    > `git config --global core.editor nano`
- If not configured, Git uses your system's default editor, which is generally Vim.

# Using Command Line Git

1. Initiate a Git repository.
2. Edit content (i.e. change some files).
3. Store a 'snapshot' of the current file state.\*

# Initiate a Git repository

# Initiate a Git repository

```
> cd ~/Desktop  
> mkdir git_class # Make a new directory  
> cd git_class    # Change to this directory  
> git init       # Our first Git command!  
> ls -a           # what happened?
```

# Side-Note: The .git/ directory

- Git created a ‘hidden’ **.git/** directory inside your current working directory.
- You can use the ‘**ls -a**’ command to list (*i.e.* see) this directory and its contents.
- This is where Git stores all its goodies - **this is Git!**
- You should not need to edit the contents of the **.git** directory for now but do feel free to poke around.

# Important Git commands

```
> git status      # report on content changes
```

```
> git add <filename>    # stage/track a file  
> git commit -m "message"  # snapshot
```

# Important Git commands

```
> git status      # report on content changes
```

```
> git add <filename>    # stage/track a file
```

```
> git commit -m "message"  # snapshot
```

You will use these three commands again and again in your Git workflow!

# Git TRACKS your directory content

- To get a report of changes (since last commit) use:

> **git status**

- You tell Git which files to track with:

> **git add <filename>**

This adds files to a so called **STAGING AREA** (akin to a “shopping cart” before purchasing).

- You tell Git when to take an historical **SNAPSHOT** of your staged files (*i.e.* record their current state) with:

> **git commit -m ‘Your message about changes’**

# Example Git workflow



Eva creates a README text file  
(this starts as untracked)



Adds file to STAGING AREA\*  
(tracked and ready to take a snapshot)



Commit changes\*  
(records snapshot of staged files!)

# Example Git workflow



Eva creates a README text file



Adds file to STAGING AREA\*



Commit changes\*



Eva modifies README and adds a ToDo text file



Adds both to STAGING AREA\*



Commit changes\*

# 1. Eva creates a README file

```
> # cd ~/Desktop/git_class  
> # git init  
  
> echo "This is a first line of text." > README  
> git status      # Report on changes  
# On branch master  
#  
# Initial commit  
#  
# Untracked files:  
#   (use "git add <file>..." to include in what will be committed)  
#  
#     README  
#  
# nothing added to commit but untracked files present (use "git add" to track)
```

## 2. Adds to ‘staging area’

```
> git add README      # Add README file to staging area  
> git status          # Report on changes  
  
# On branch master  
#  
# Initial commit  
#  
# Changes to be committed:  
#   (use "git rm --cached <file>..." to unstage)  
#  
#       new file: README  
#
```

### 3. Commit changes

```
> git commit -m "Create a README file" # Take snapshot  
# [master (root-commit) 8676840] Create a README file  
# 1 file changed, 1 insertion(+)  
# create mode 100644 README
```

```
> git status # Report on changes  
# On branch master  
# nothing to commit, working directory clean
```

# 4. Eva modifies README file and adds a ToDo file

```
> echo "This is a 2nd line of text." >> README
> echo "Learn git basics" >> ToDo

> git status      # Report on changes
# On branch master
#
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified: README
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       ToDo
#
# no changes added to commit (use "git add" and/or "git commit -a")
```

## 5. Adds both files to ‘staging area’

```
> git add README ToDo      # Add both files to ‘staging area’  
> git status                 # Report on changes  
  
# On branch master  
# Changes to be committed:  
#   (use "git reset HEAD <file>..." to unstage)  
#  
#       modified: README  
#       new file: ToDo  
#
```

# 6. Commits changes

```
> git commit -m "Add ToDo and modify README"
```

```
# [master 7b679fa] Add ToDo and modify README
# 2 files changed, 2 insertions(+)
# create mode 100644 ToDo
```

```
> git status
```

```
# On branch master
# nothing to commit, working directory clean
```

# Example Git workflow

1.  Eva creates a README text file
2.  Adds file to STAGING AREA\*
3.  Commit changes\*
  
4.  Eva modifies README and adds a ToDo text file  

5.  Adds both to STAGING AREA\*
6.  Commit changes\*

...But, how do we see the history of our project changes?

# git log: Timeline history of snapshots (*i.e.* commits)

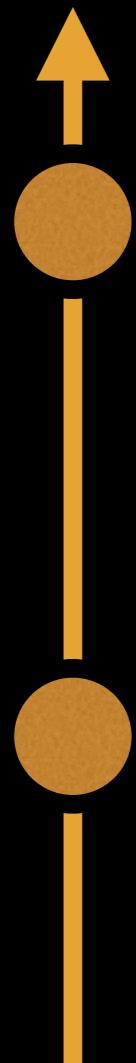
```
> git log
```

```
# commit 7b679fa747e8640918fcaad7e4c3f9c70c87b170
# Author: Barry Grant <bjgrant@umich.edu>
# Date: Thu Jul 30 11:43:40 2015 -0400
#
#   Add ToDo and finished README
#
# commit 86768401610770ae32e2fd4faee07d1d5c68619c
# Author: Barry Grant <bjgrant@umich.edu>
# Date: Thu Jul 30 11:26:40 2015 -0400
#
#   Create a README file
#
```

# git log: Timeline history of snapshots (*i.e.* commits)

```
> git log
```

```
# commit 7b679fa747e8640918fcaad7e4c3f9c70c87b170 -----
```



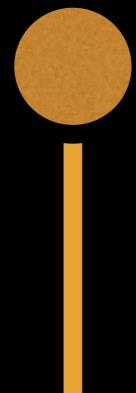
```
# Date: Thu Jul 30 11:43:40 2015 -0400
```

```
#
```

```
#   Add ToDo and finished README
```

```
#
```

```
# commit 86768401610770ae32e2fd4faee07d1d5c68619c -----
```



```
# Date: Thu Jul 30 11:26:40 2015 -0400
```

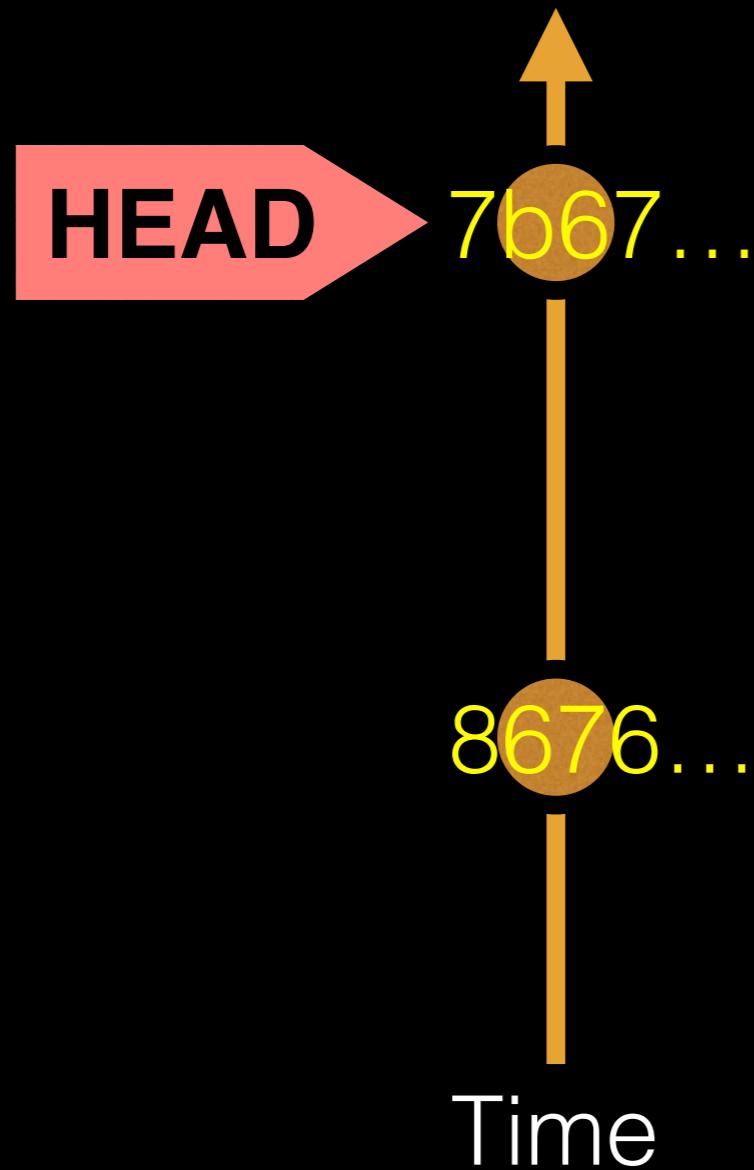
```
#
```

```
#   Create a README file
```

```
#
```

Past

# Side-Note: Git history is akin to a graph

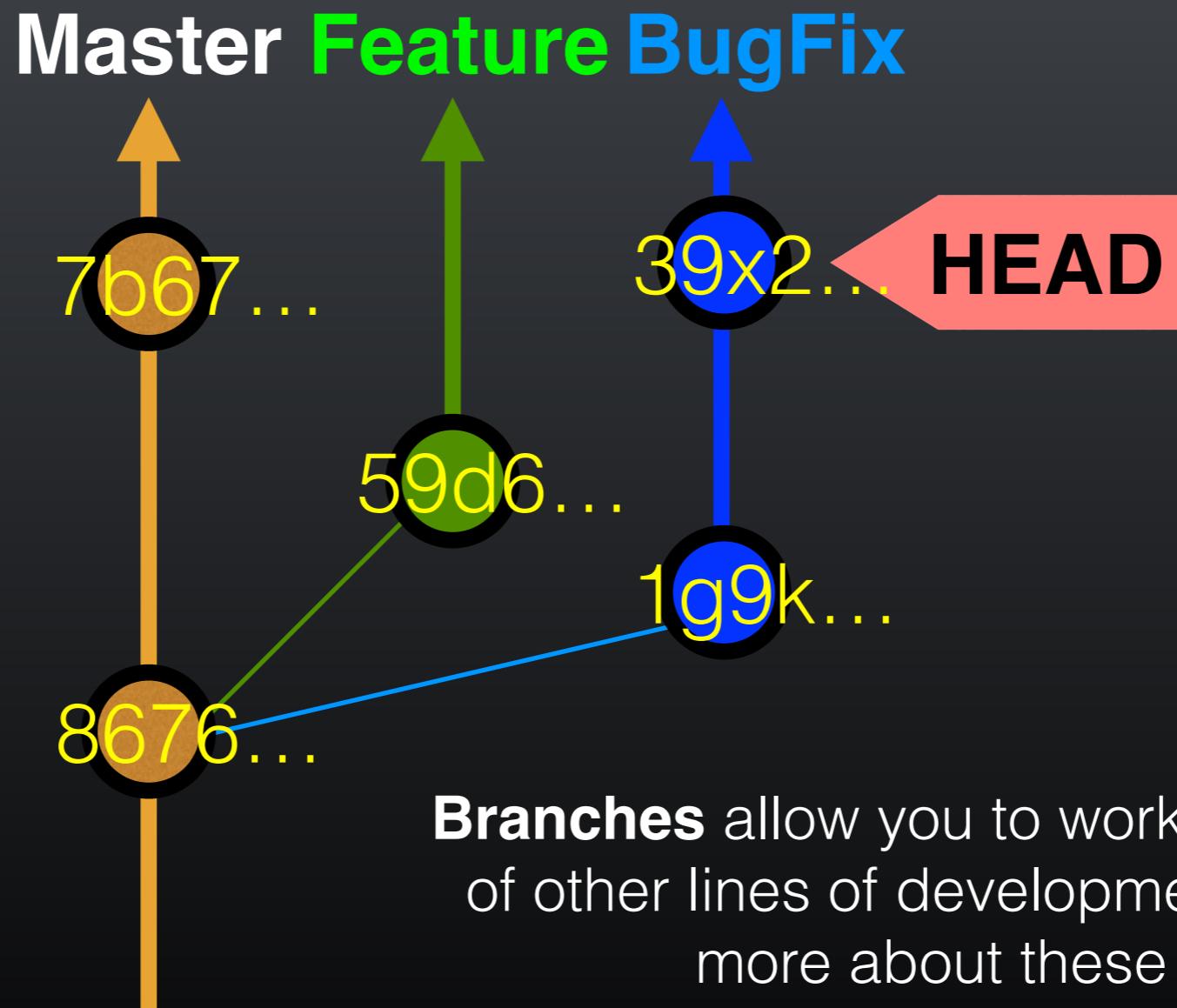


Nodes are **commits** labeled by their unique '**commit ID**'.

(This is a **CHECKSUM** of the commits author, time, commit msg, commit content and previous commit ID).

**HEAD** is a reference (or '**pointer**') to the currently checked out commit (typically the most recent commit).

# Projects can have complicated graphs due to **branching**



## Key Points:

You explicitly and iteratively tell git what files to track (“**git add**”) and snapshot (“**git commit**”).

Git keeps an historical log “(**git log**)” of the content changes (and your comments on these changes) at each past commit.

It is good practice to regularly check the status of your working directory, staging arena repo (“**git status**”)

# Break

# Summary of key Git commands:

```
> git status          # Get a status report of changes since last commit
```

```
> git add <filename>      # Tell Git which files to track/stage
```

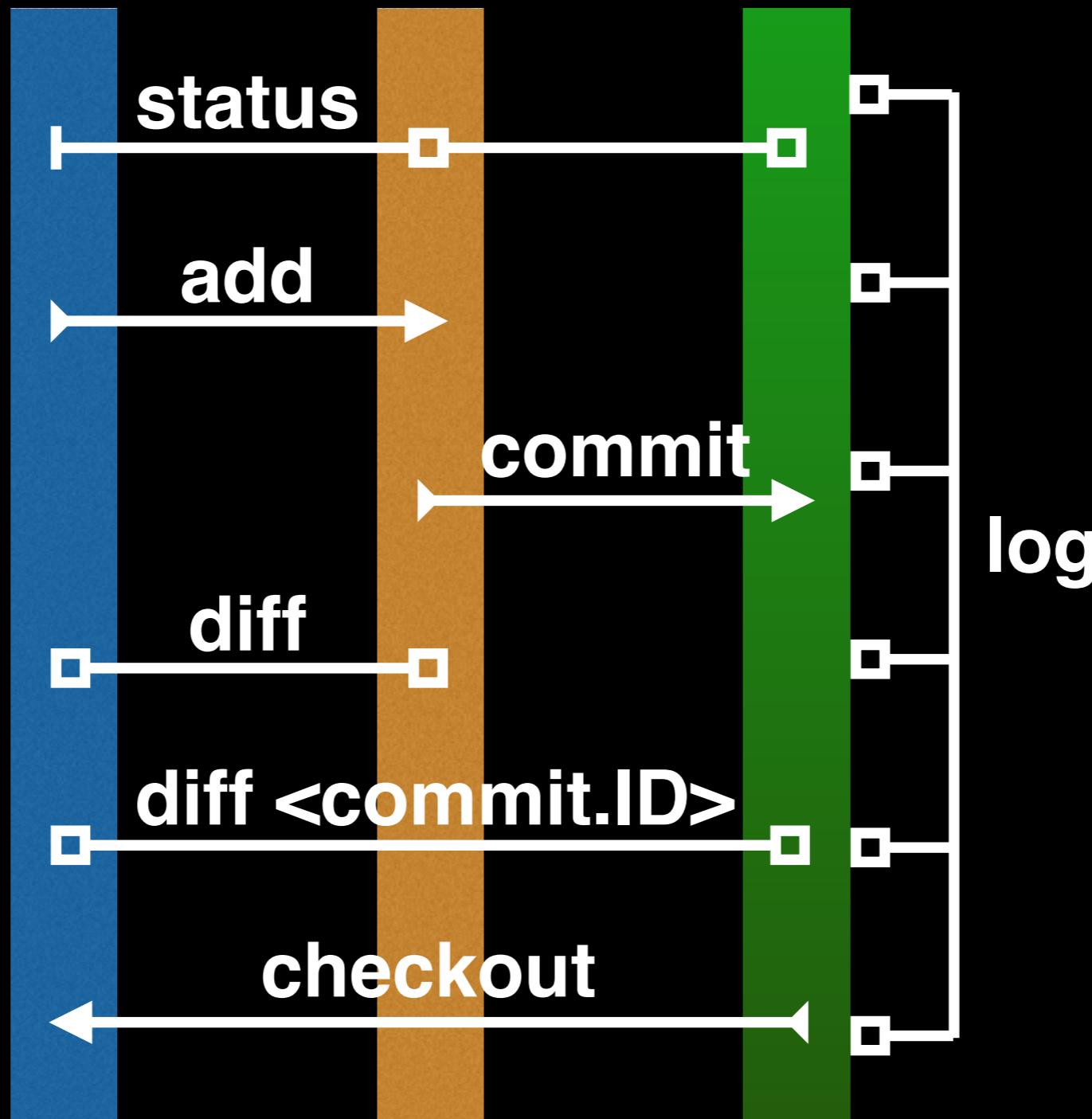
```
> git commit -m 'Your message'    # Take a content snapshot!
```

```
> git log            # Review your commit history
```

```
> git diff <commit.ID> <commit.ID> # Inspect content differences
```

```
> git checkout <commit.ID> # Navigate through the commit history
```

Your 'Staging Area' Local Repository

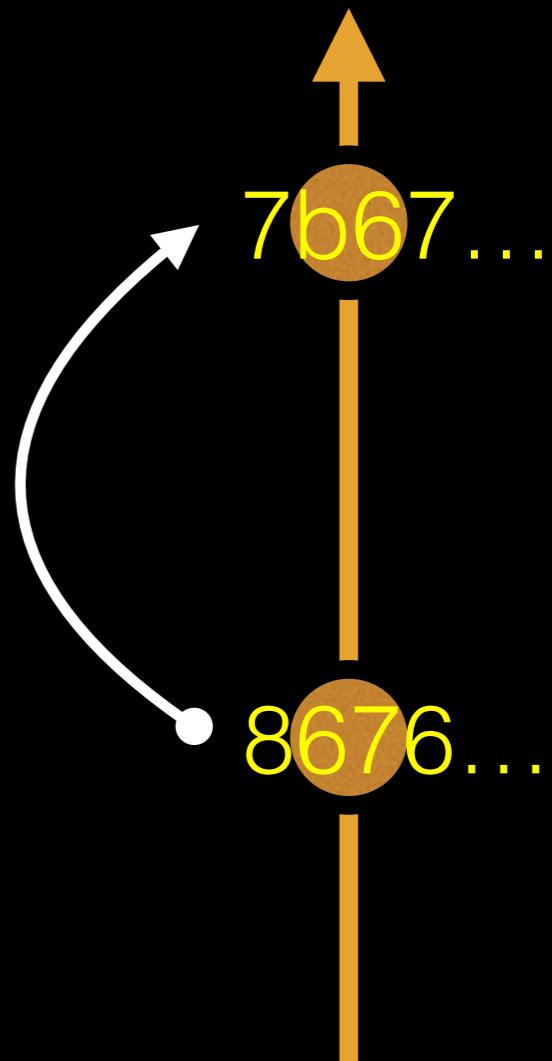


# git diff: Show changes between commits

> **git diff 8676 7b67**

```
# diff --git a/README b/README
# index 73bc85a..67bd82c 100644
# --- a/README
# +++ b/README
# @@ -1 +1,2 @@
# This is a first line of text.
# +This is a 2nd line of text.

# diff --git a/ToDo b/ToDo
# new file mode 100644
# index 0000000..14fdbd56
# --- /dev/null
# +++ b/ToDo
# @@ -0,0 +1 @@
# +Learn git basics
```

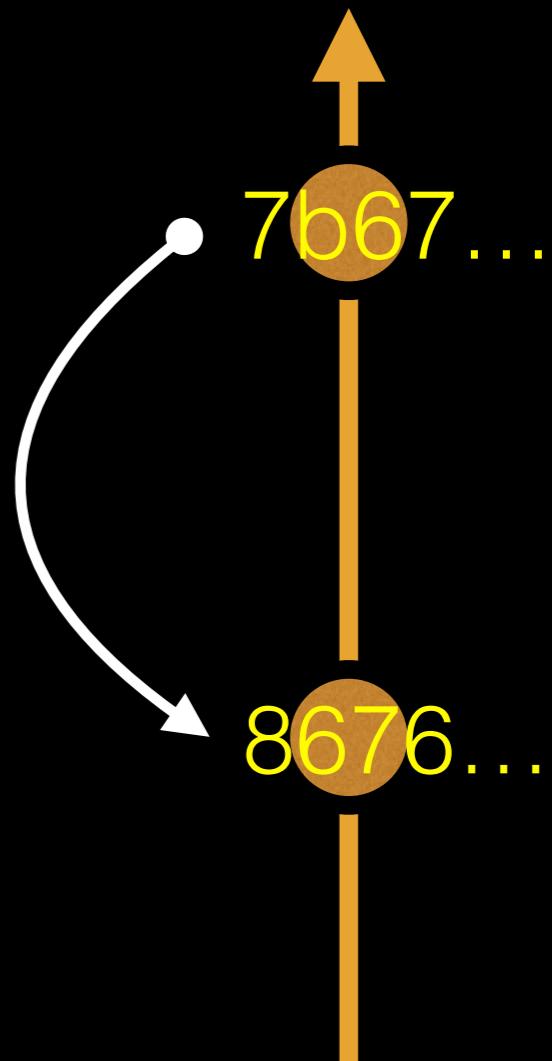


# git diff: Show changes between commits

> **git diff 7b67 8676**

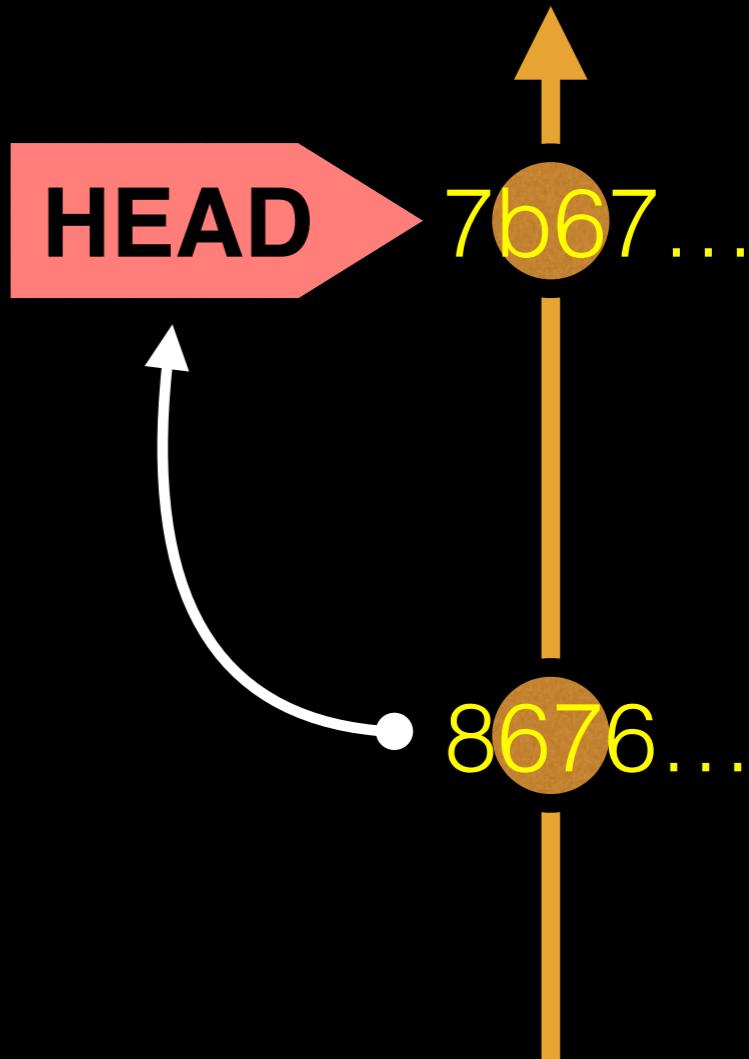
```
# diff --git a/README b/README
# index 67bd82c..73bc85a 100644
# --- a/README
# +++ b/README
# @@ -1,2 +1 @@
# This is a first line of text.
# -This is a 2nd line of text.

# diff --git a/ToDo b/ToDo
# deleted file mode 100644
# index 14fb56..0000000
# --- a/ToDo
# +++ /dev/null
# @@ 1 +0,0 @@
# -Learn git basics
```

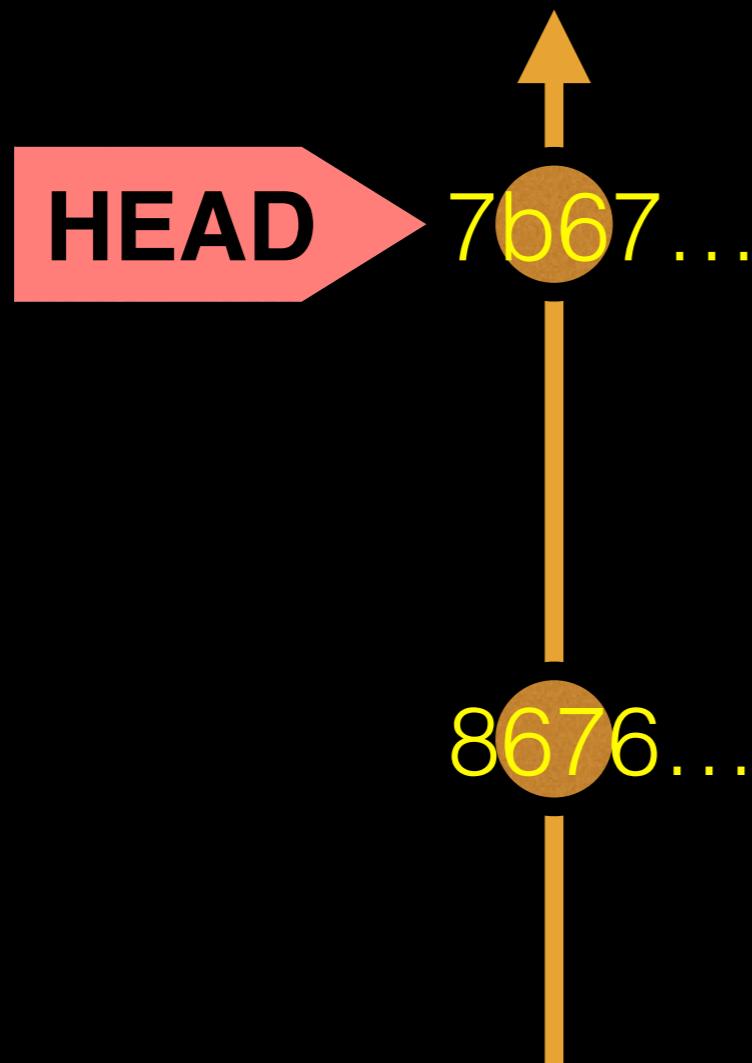


# git diff: Show changes between commits

```
> git diff 8676          ## Difference to current HEAD position!  
  
# diff --git a/README b/README  
# index 73bc85a..67bd82c 100644  
# --- a/README  
# +++ b/README  
# @@ -1 +1,2 @@  
# This is a first line of text.  
# +This is a 2nd line of text.  
  
# diff --git a/ToDo b/ToDo  
# new file mode 100644  
# index 0000000..14fdbd56  
# --- /dev/null  
# +++ b/ToDo  
# @@ -0,0 +1 @@  
# +Learn git basics
```



# **HEAD** advances automatically with each new commit



To move **HEAD** (back or forward) on the Git graph (and retrieve the associated snapshot content) we can use the command:

> `git checkout <commit.ID>`

# git checkout: Moves HEAD

> **more README**

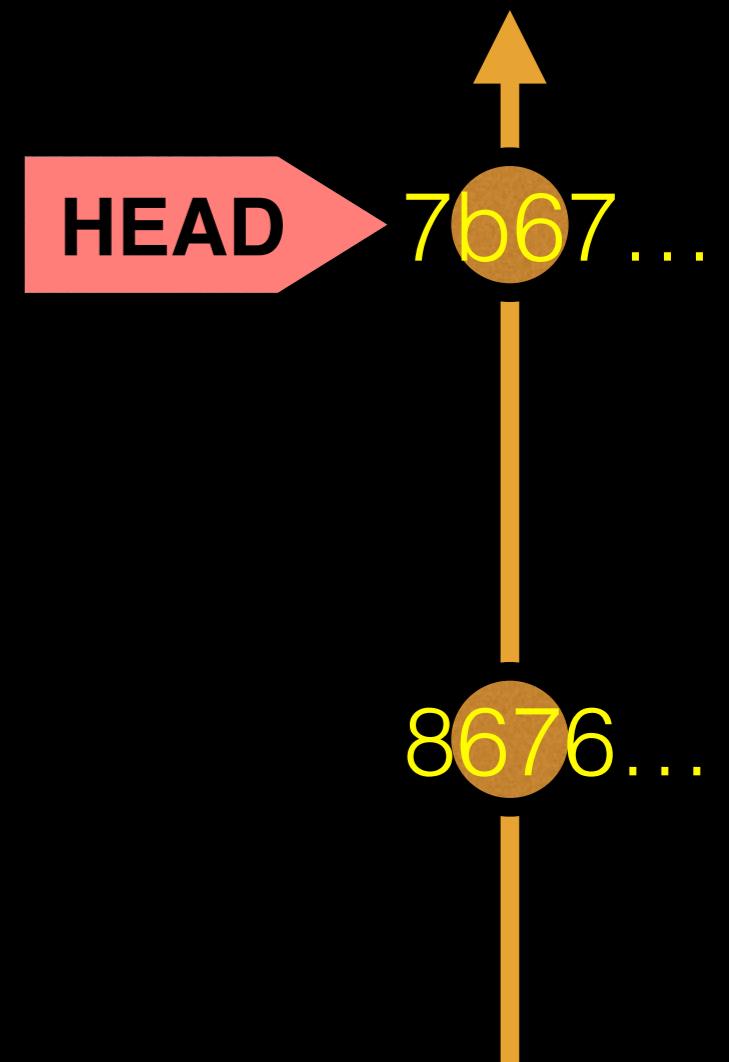
This is a first line of text.

This is a 2nd line of text.

> **git log --oneline**

# 7b679fa Add ToDo and finished README

# 8676840 Create a README file



# git checkout: Moves HEAD (e.g. back in time)

> **more README**

This is a first line of text.

This is a 2nd line of text.

> **git log --oneline**

# 7b679fa Add ToDo and finished README

# 8676840 Create a README file

> **git checkout 8676840**

# You are in 'detached HEAD' state...<cut>...

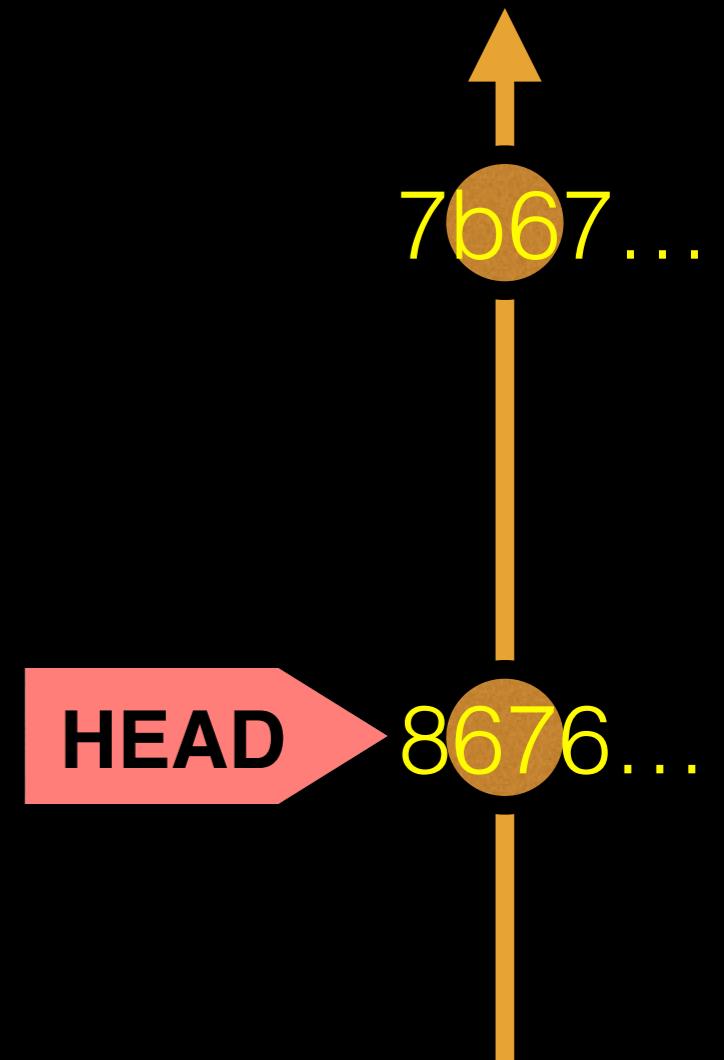
# HEAD is now at 8676840... Create a README file

> **more README**

This is a first line of text.

> **git log --oneline**

# 8676840 Create a README file



# git checkout: Moves HEAD (e.g. back to the future!)

> **git checkout master**

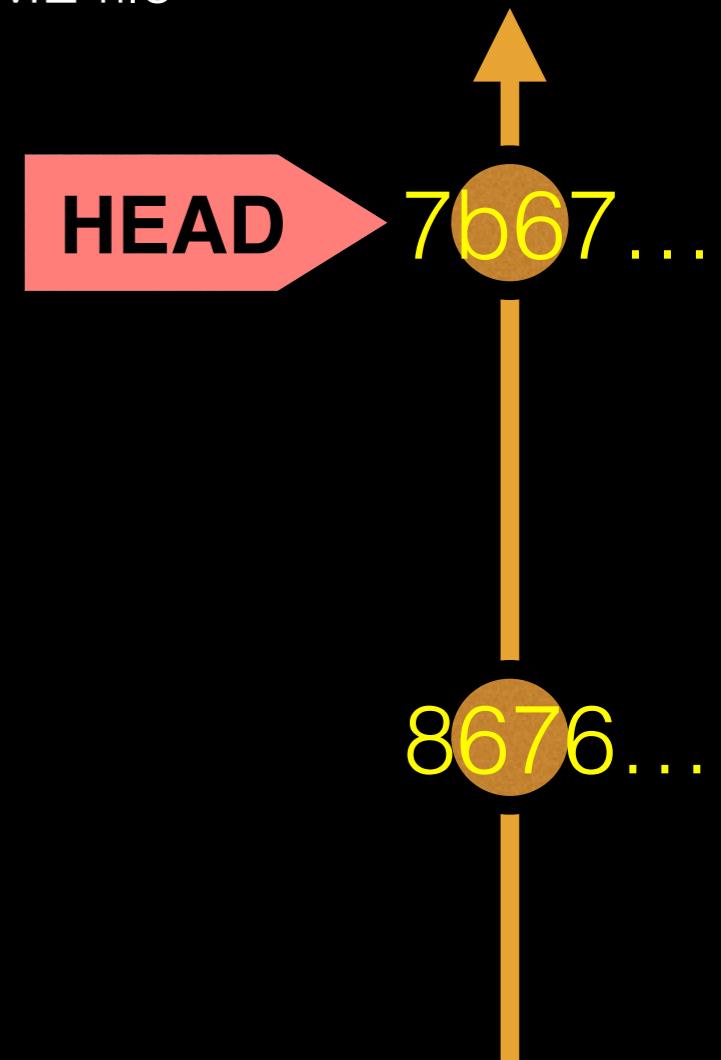
```
# Previous HEAD position was 8676840... Create a README file
# Switched to branch 'master'
```

> **git log --oneline**

```
# 7b679fa Add ToDo and finished README
# 8676840 Create a README file
```

> **more README**

```
This is a first line of text.
This is a 2nd line of text.
```



# Side-Note: There are two\* main ways to use **git checkout**

- Checking out a **commit** makes the entire working directory match that commit. This can be used to view an old state of your project.

> `git checkout <commit.ID>`

- Checking out a **specific file** lets you see an old version of that particular file, leaving the rest of your working directory untouched.

> `git checkout <commit.ID> <filename>`

# You can discard revisions with **git revert**

- The **git revert** command undoes a committed snapshot.
- But, instead of removing the commit from the project history, it figures out how to **undo the changes** introduced by the commit and **appends a new commit** with the resulting content.  
  
`> git revert <commit.ID>`
- This prevents Git from losing history!

# Removing untracked files with **git clean**

- The **git clean** command removes untracked files from your working directory.
  - Like an ordinary **rm** command, **git clean** is not undoable, so make sure you really want to delete the untracked files before you run it.
- > `git clean -n` # dry run display of files to be ‘cleaned’
- > `git clean -f` # remove untracked files

# GUIs

**Tower** (Mac only)

**GitHub\_Desktop** (Mac, Windows)

**SourceTree** (Mac, Windows)

**SmartGit** (Linux)

**RStudio**

<https://git-scm.com/downloads/guis>