



Class 14
Git & GitHub

Barry Grant
UC San Diego
<http://thegrantlab.org>



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

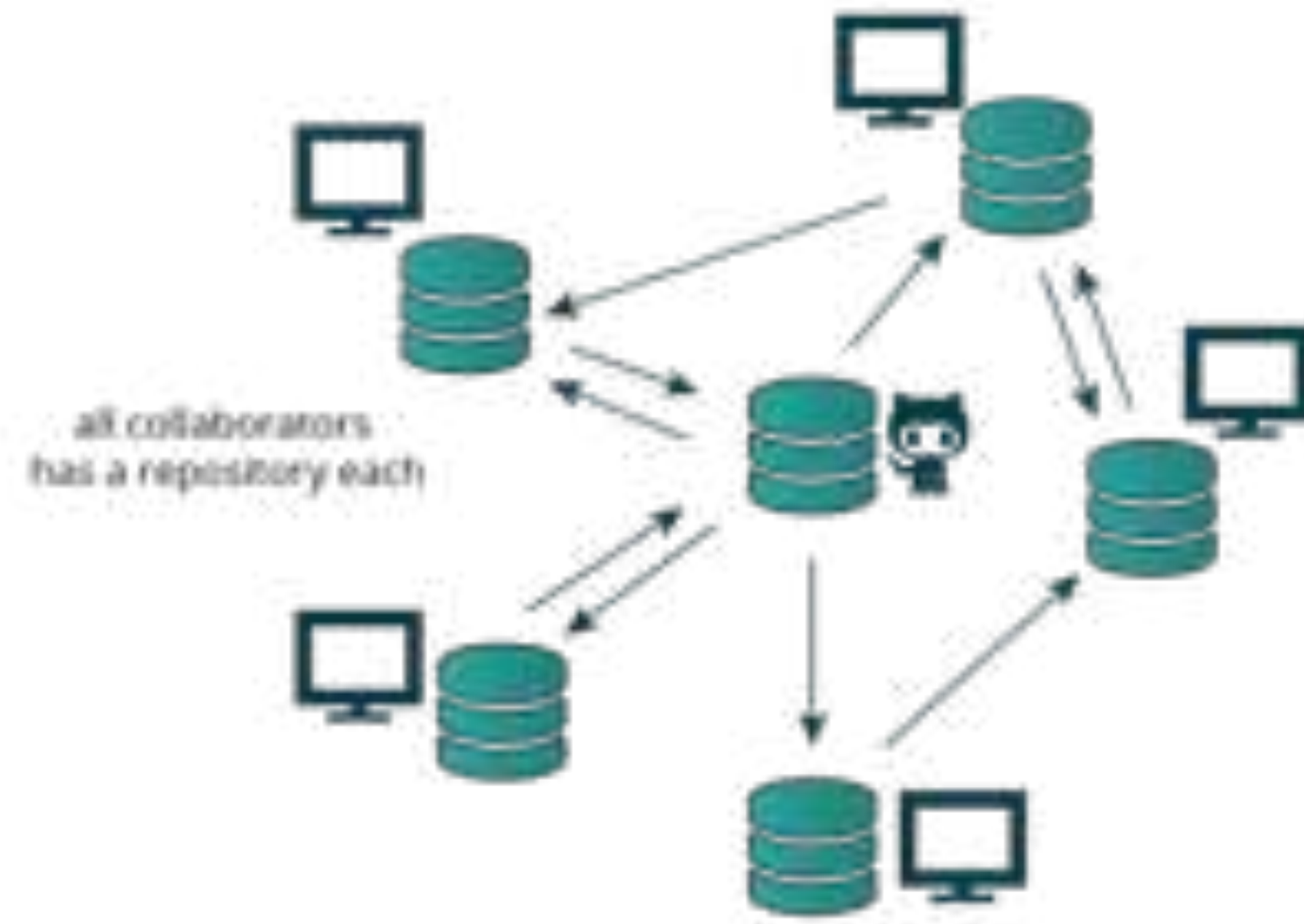
There are many VCS available, see:

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.

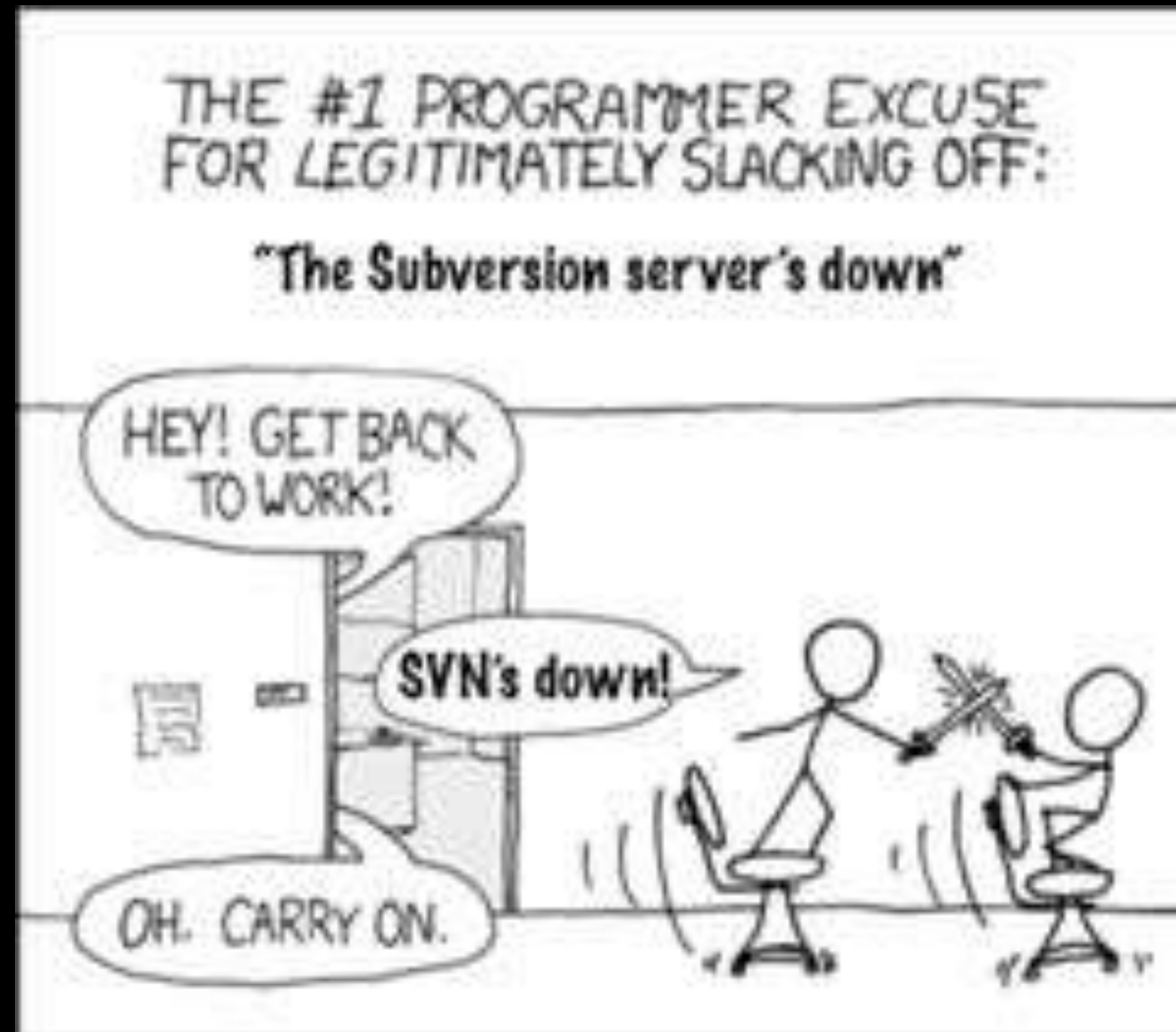
THE #1 PROGRAMMER EXCUSE
FOR LEGITIMATELY SLACKING OFF:

"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>

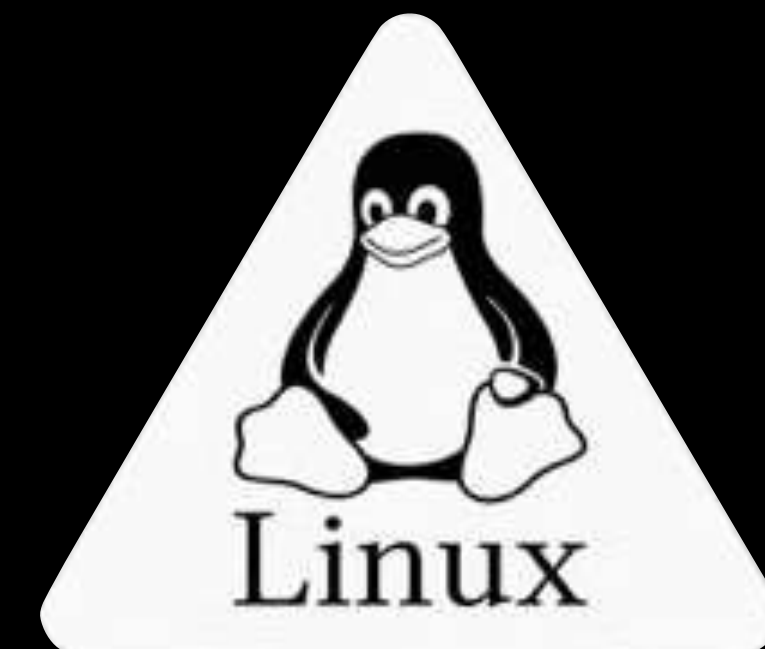
Where did Git come from?

Written initially by Linus Torvalds to support Linux kernel and OS development.

Meant to be distributed, fast and more natural.

Capable of handling large projects.

Now the most popular free VCS!





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

Note: You hopefully already have git installed!
To check open the “Terminal” tab in RStudio and type:

① `which git`

② `git --version`

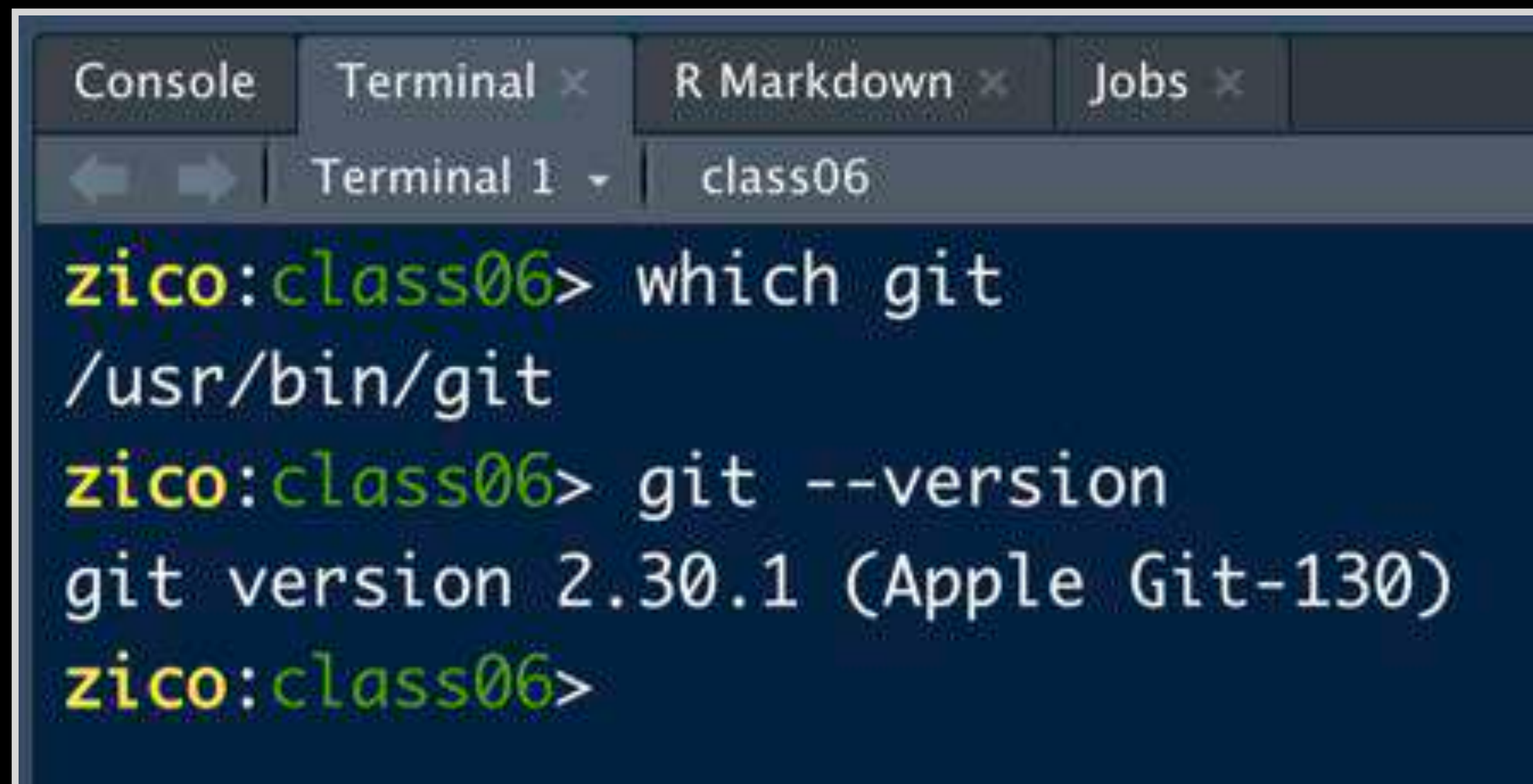
Obtaining Git

Note: You hopefully already have git installed!
To check open the “Terminal” tab in RStudio and type:

① **which git**

② **git --version**

Obtaining Git



```
Console Terminal x R Markdown x Jobs x  
Terminal 1 class06  
zico:class06> which git  
/usr/bin/git  
zico:class06> git --version  
git version 2.30.1 (Apple Git-130)  
zico:class06>
```


Note: You might already have git installed
To check open the “Terminal” tab in RStudio and type:

① **which git**

② **git --version**

Obtaining Git

Windows Only (if you have problems)

If the “**which git**” command did not work, try:

where git

If this works see next slide. If not then you need to install **GitBash**, instructions here:

Class [Computer Setup Page](#)

Mac Only (if you have problems)

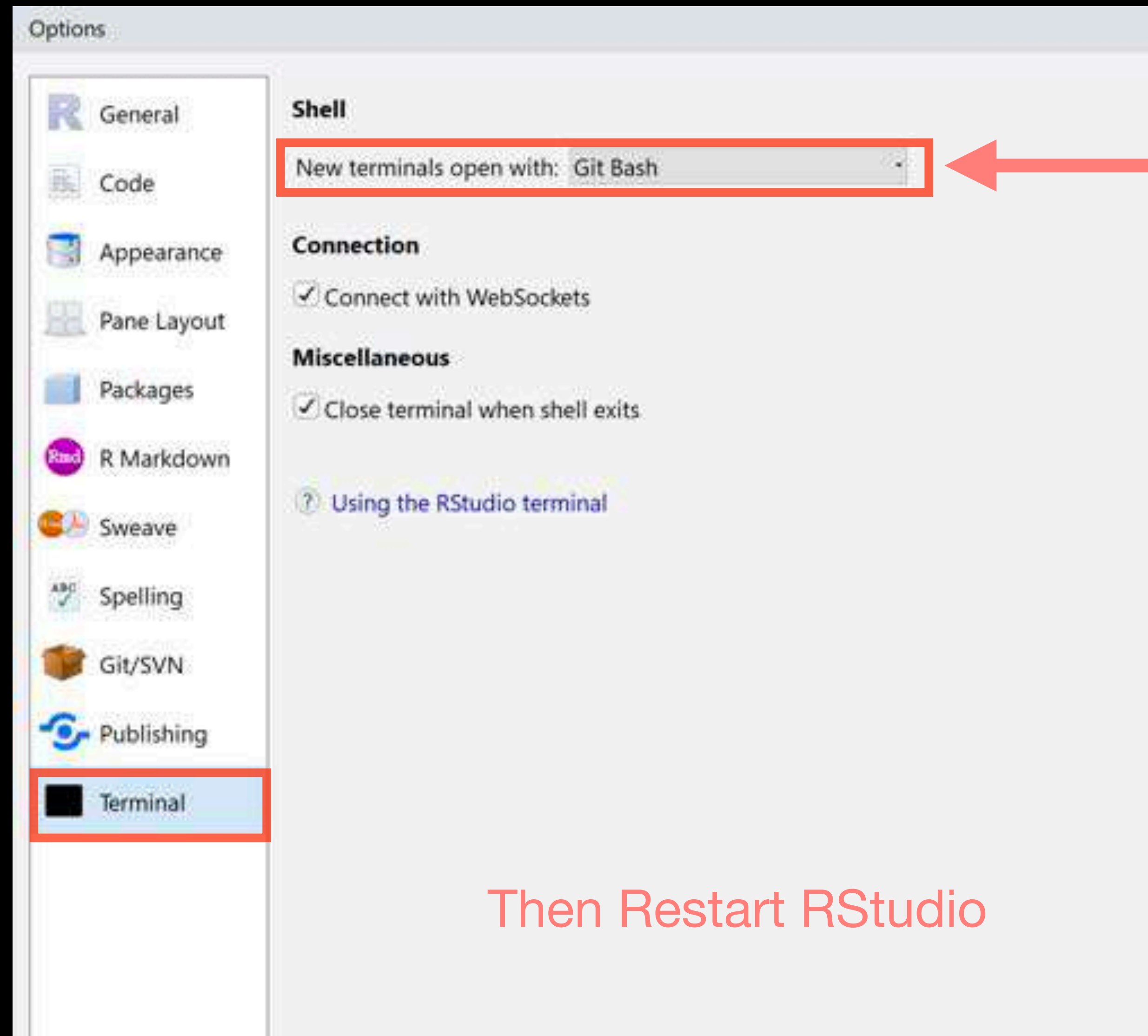
If the “**which git**” command did not work, you may need to install select developer tools.

In your Terminal type:

xcode-select --install

On a Windows PC Only!

Go to: RStudio > Tools > Global Options > Terminal



**Make sure
Git Bash is
selected!**

Then Restart RStudio

Do it Yourself!

Note: You might already have git installed
To check open the “Terminal” tab in RStudio and type:

- 1 **which git**
- 2 **git --version**

Installing Git

Windows (if you have problems)

Follow the GitBash instructions here:

[Class Computer Setup Page](#)

Mac (if you have problems)

In the **Terminal** instal select developer tools

xcode-select --install

Configuring Git

Do it Yourself!

Configuring Git

(RStudio Terminal Tab)
(...or *RStudio* > *Tools* > *Shell*)

First tell Git who you are

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

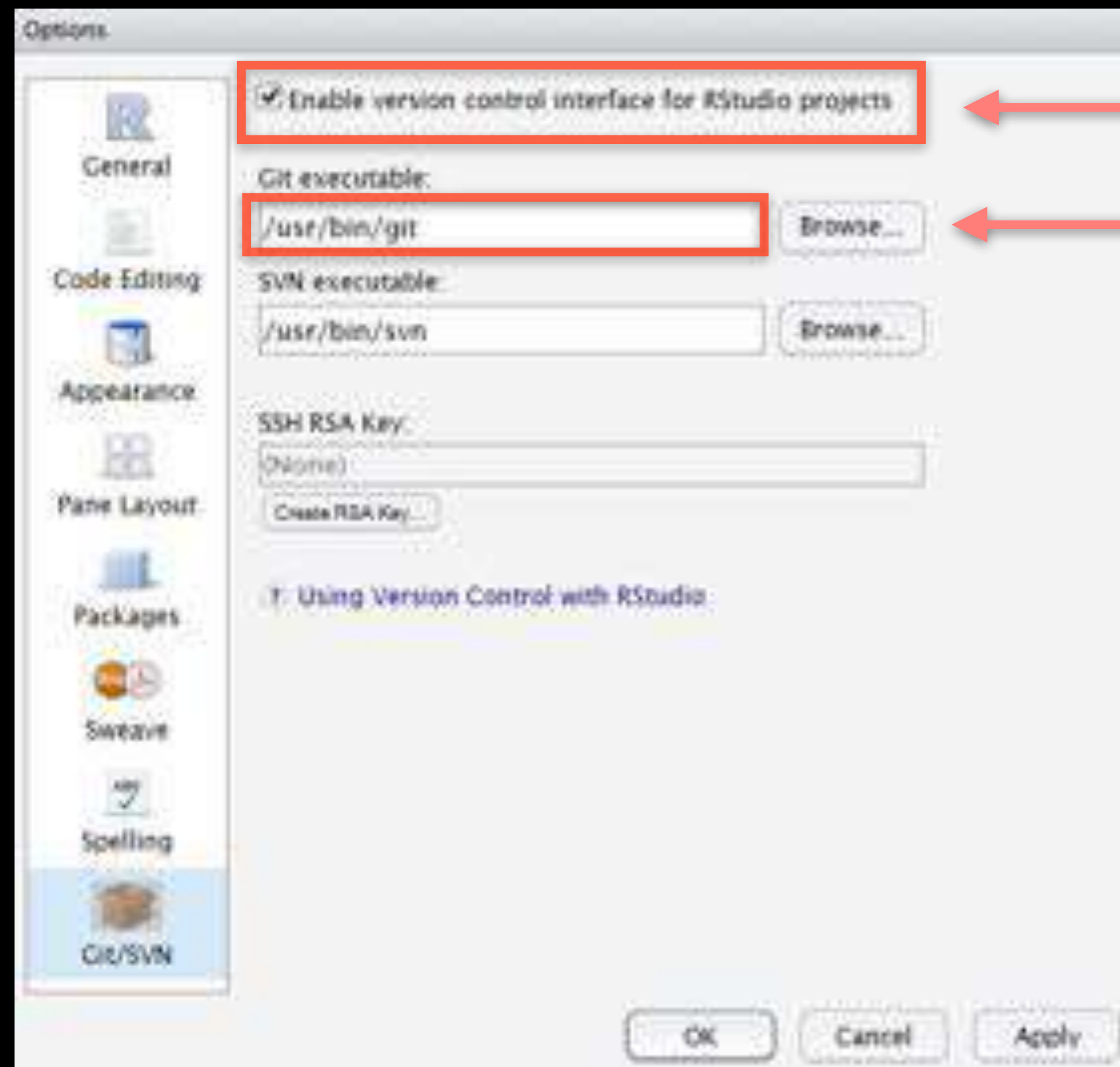
Configuring RStudio

For Mac & Linux

(PC on next slide)

Do it Yourself!

Go to: RStudio > Tools > Global Options > Git/SVN



- 1 Make sure this is **ticked!**
- 2 Make sure this is **correct!**

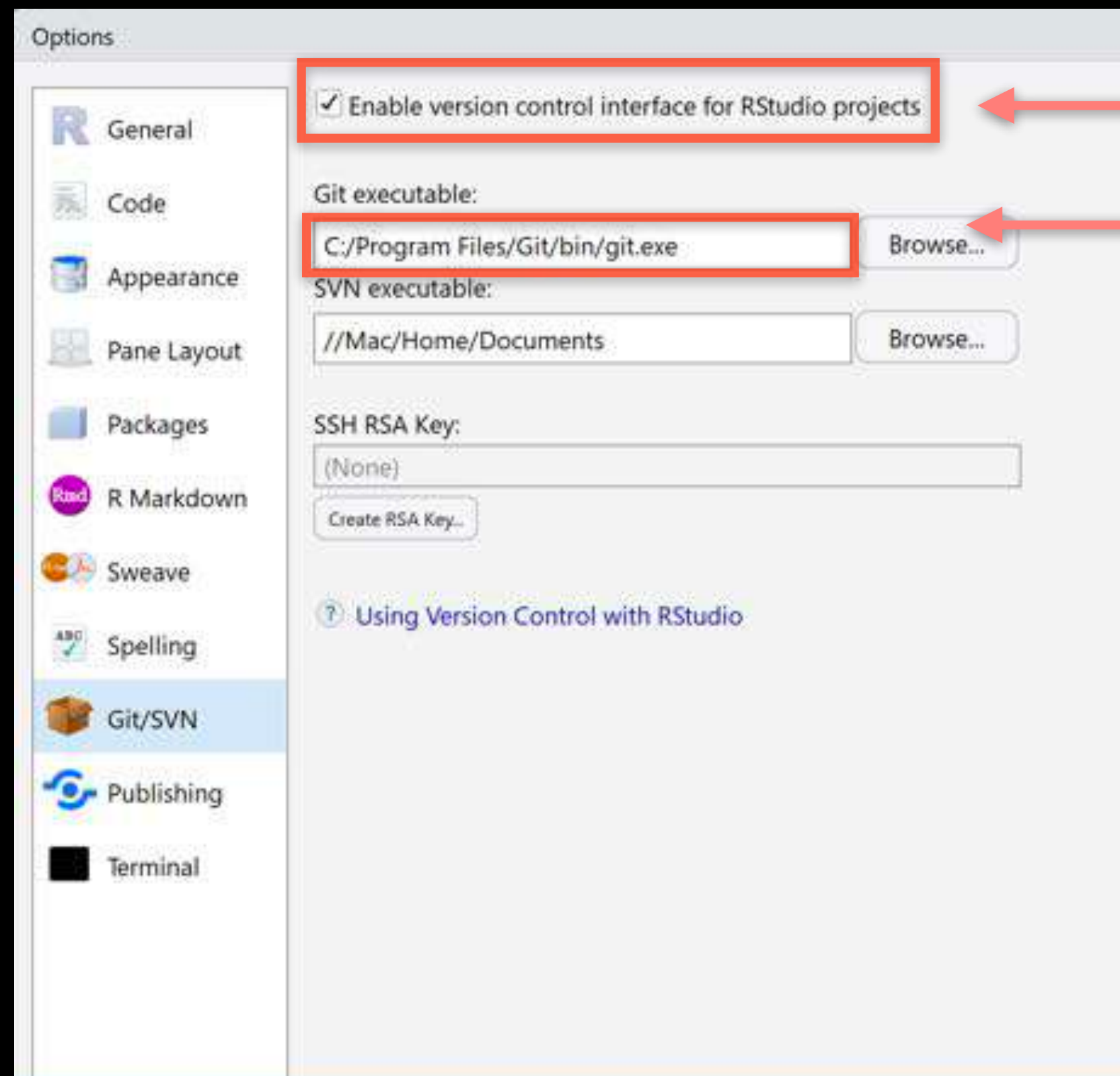
Check in your RStudio "Terminal" tab:

```
blitz:another> which git
/usr/local/bin/git
blitz:another>
```

Do it Yourself!

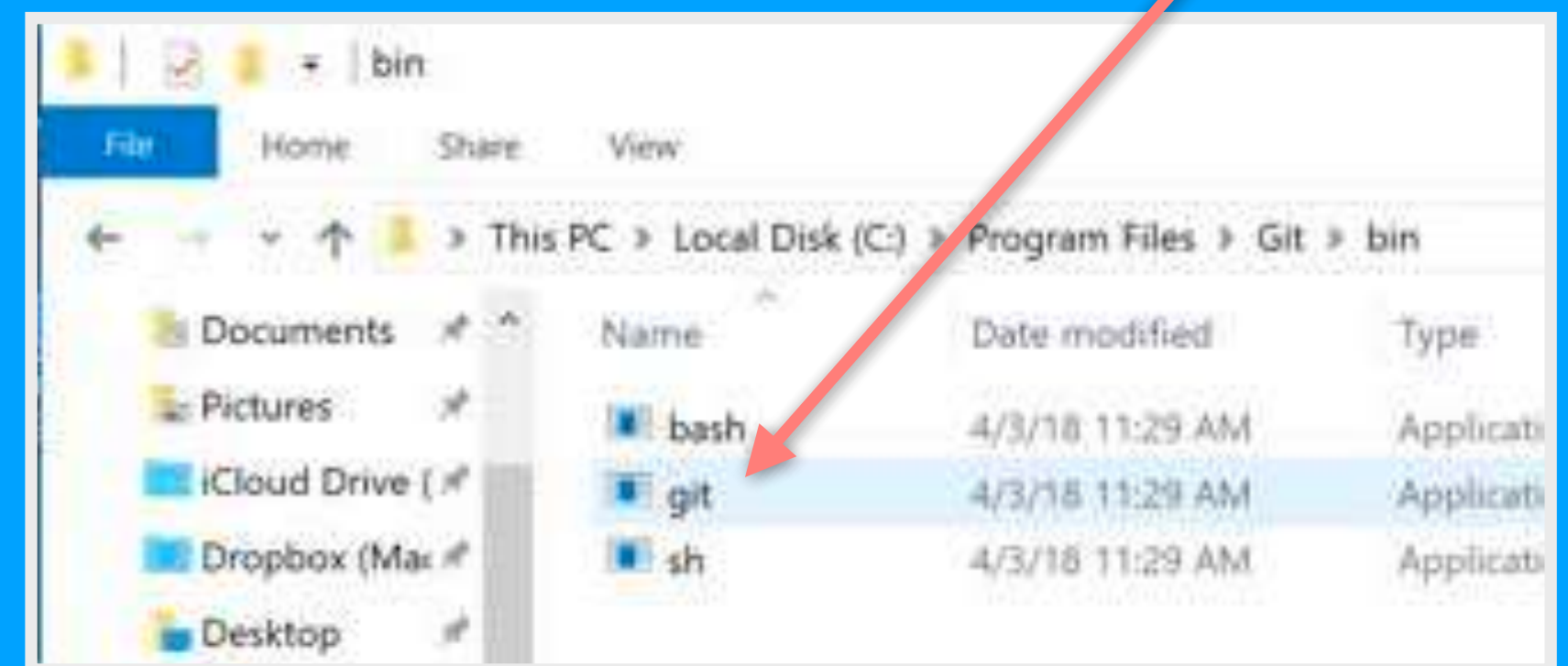
On a PC!

Go to: RStudio > Tools > Global Options > Git/SVN



- 1 Make sure this is **ticked!**
- 2 This is the PATH for **PC!**

Check in your Windows File Explorer:



Restart RStudio!

Using Git

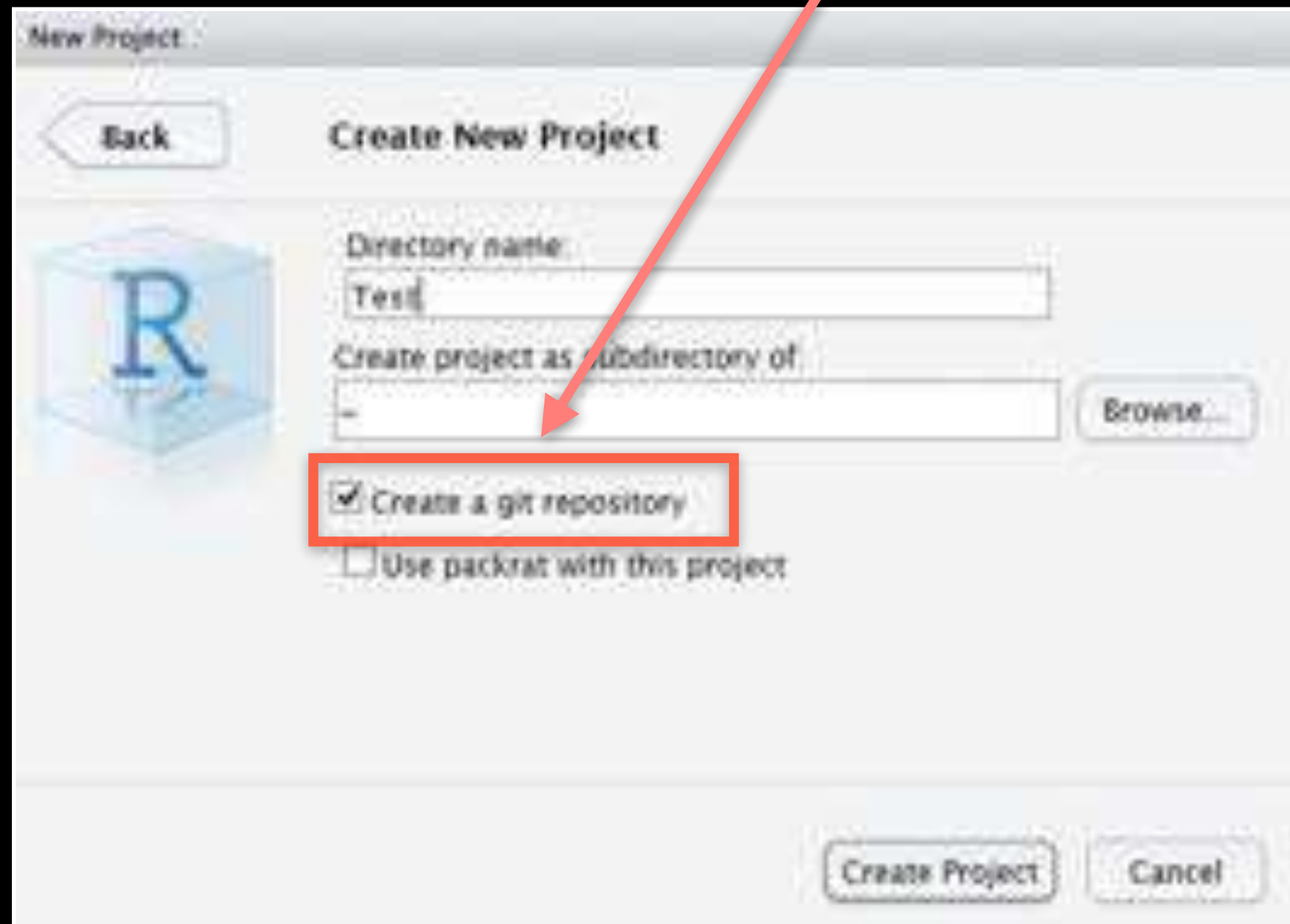
Using Git

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

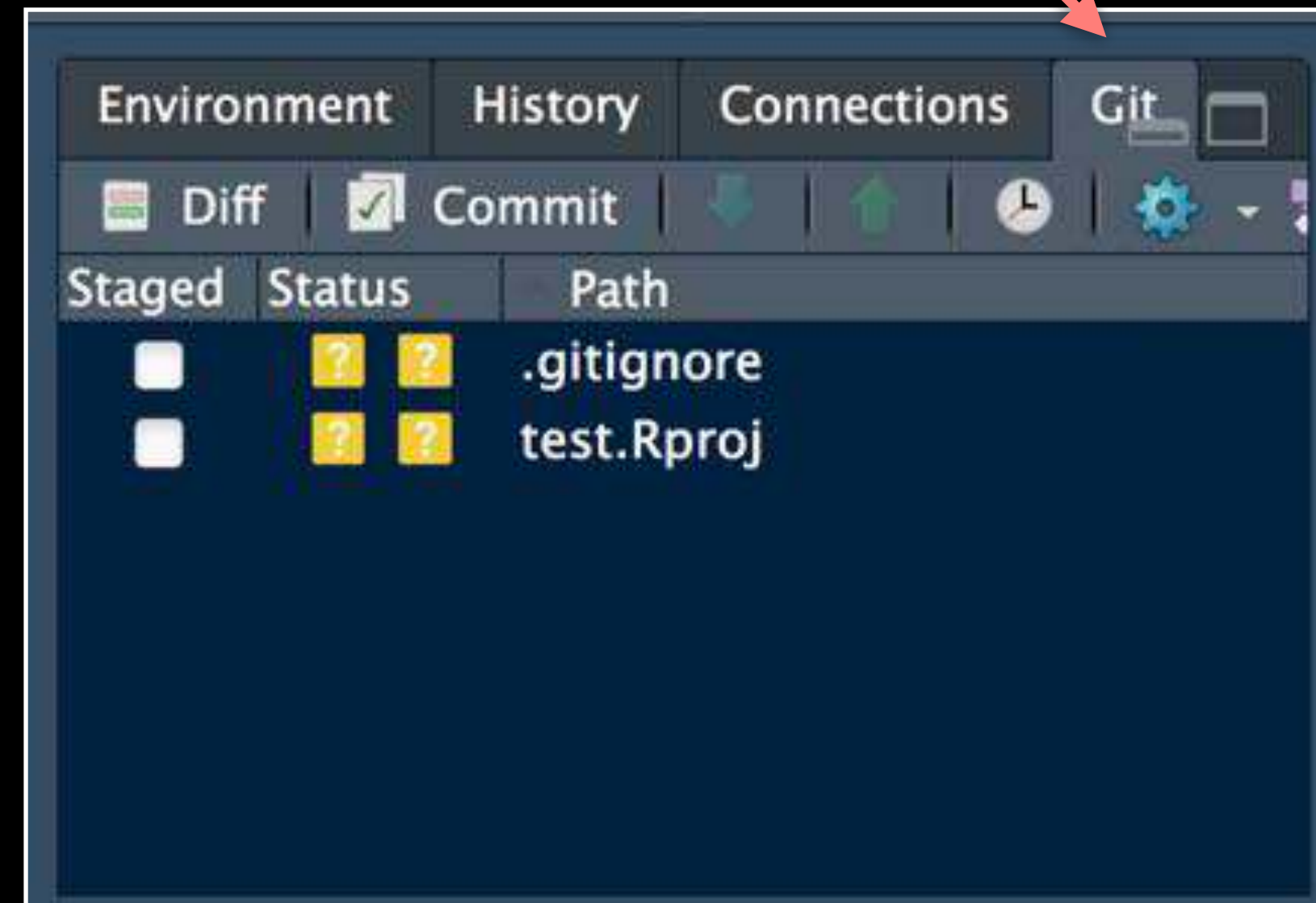
Do it Yourself!

Create a new **Test** RStudio project

1 New option to create a Git repository...



2 New Git tab...



Check if new Git options appear in RStudio?

Using Git in RStudio

1. **Initiate** a git repository for an RStudio Project
2. Do your work and edit content as normal
3. Store a 'snapshot' of the current file state
 - (a) Periodically **add** important files to git "Staging Area"
 - (b) **Commit** changes to your "git repository"

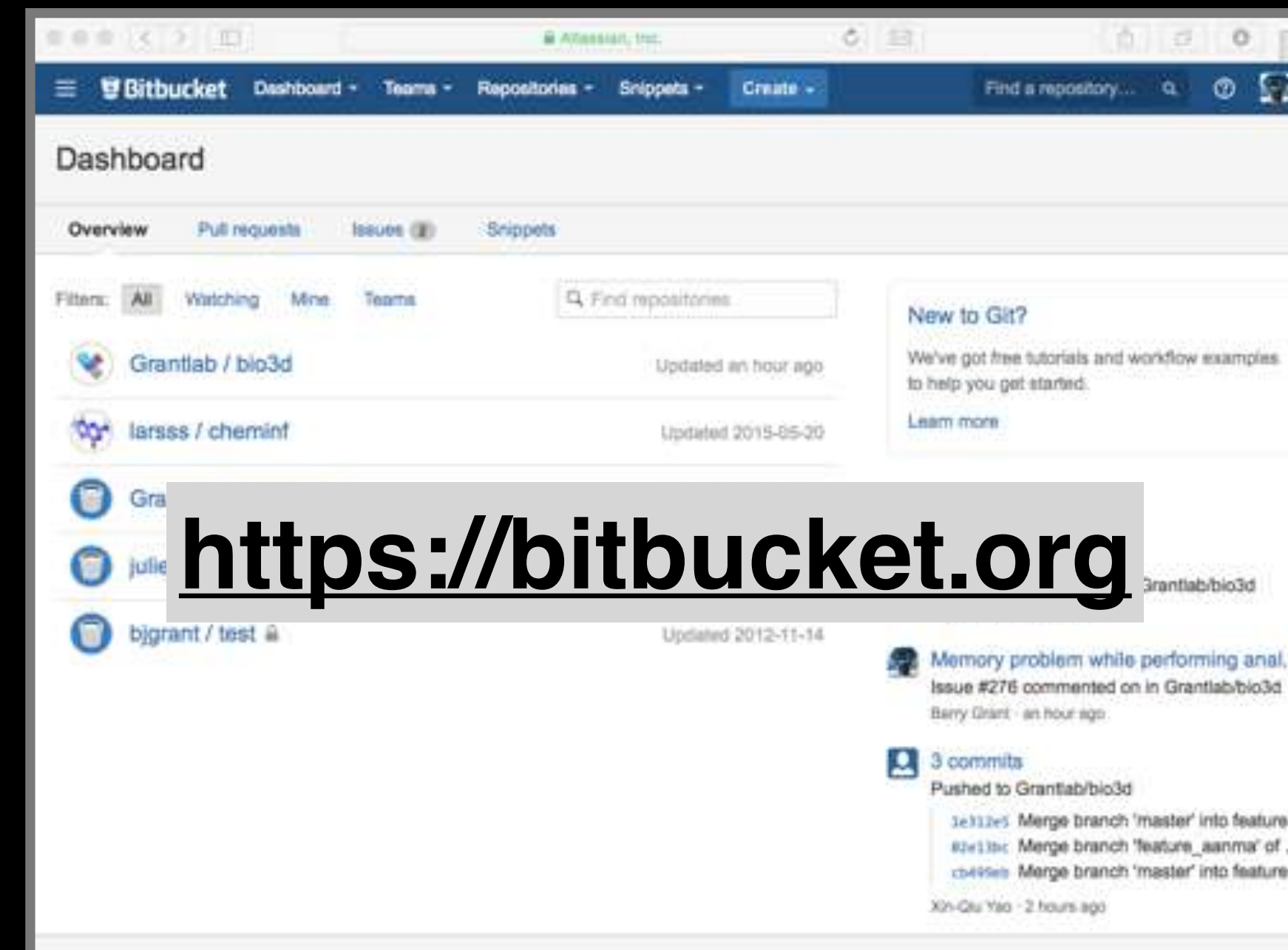
Rinse and repeat....

Follow along!

Demo:

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*.



↑ Nikkei
17893.73 0.49%

↑ Hang Seng
21404.96 0.72%

↓ U.S. 10 Yr
-Q/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



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



www.bbc.com/news/technology-44381711

Home Sport Weather Shop Real Travel More Search

NEWS

Home Video World US & Canada UK Business Tech Science Stories Entertainment & Arts Health

Microsoft buys Github code-sharing site for \$7.5bn

 **David Lee**
North America technology reporter

4 June 2018

 Share



Top Stories

Gangster 'Whitey' Bulger killed in prison
Bulger was severely beaten by one or more inmates shortly after arriving at the prison, sources say.
30 October 2018

Synagogue shooting victims' funerals start
30 October 2018

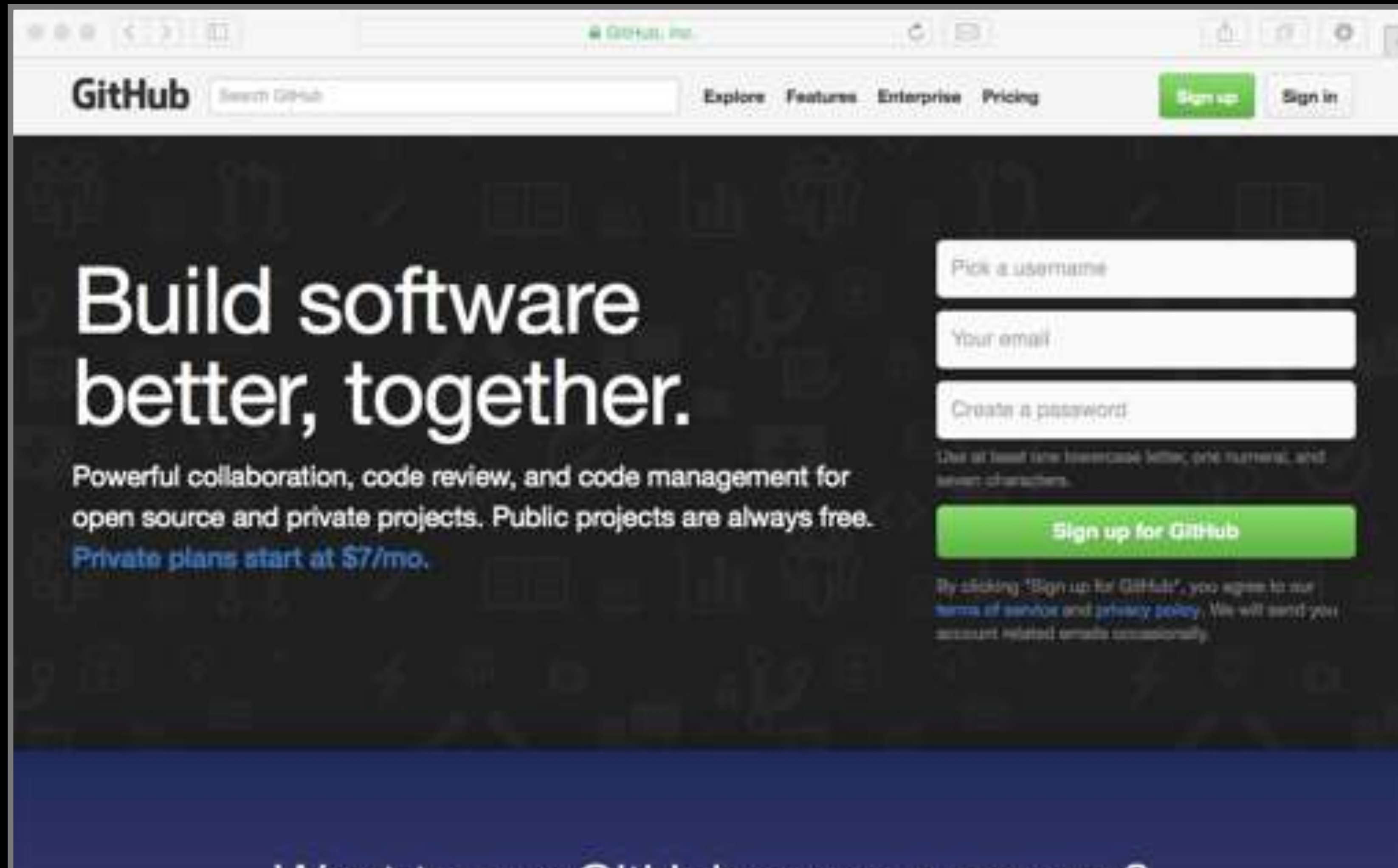
What do American voters care about?
30 October 2018

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>



The image shows a screenshot of the GitHub website's sign-up page. The browser's address bar shows the URL 'https://github.com'. The page features a dark background with white text. At the top, there is a navigation bar with the GitHub logo, a search bar, and links for 'Explore', 'Features', 'Enterprise', and 'Pricing'. On the right side of the navigation bar, there are 'Sign up' and 'Sign in' buttons. The main content area has a large heading 'Build software better, together.' followed by a sub-heading 'Powerful collaboration, code review, and code management for open source and private projects. Public projects are always free. Private plans start at \$7/mo.' To the right of this text is a sign-up form with three input fields: 'Pick a username', 'Your email', and 'Create a password'. Below the password field, there is a note: 'Use at least one lowercase letter, one numeral, and seven characters.' A prominent green 'Sign up for GitHub' button is located below the form. At the bottom of the form, there is a small disclaimer: 'By clicking "Sign up for GitHub", you agree to our terms of service and privacy policy. We will send you account related emails occasionally.'

Pick the **FREE** plan!

Completed: Set up a personal account

Step 2: Choose your plan

Step 3: Set up your business

Choose your personal plan

Plan	Cost	Private repositories	Choose
Large	\$00/month	50	Choose
Medium	\$00/month	20	Choose
Small	\$12/month	10	Choose
Micro	\$7/month	5	Choose
Free	\$0/month	0	Choose

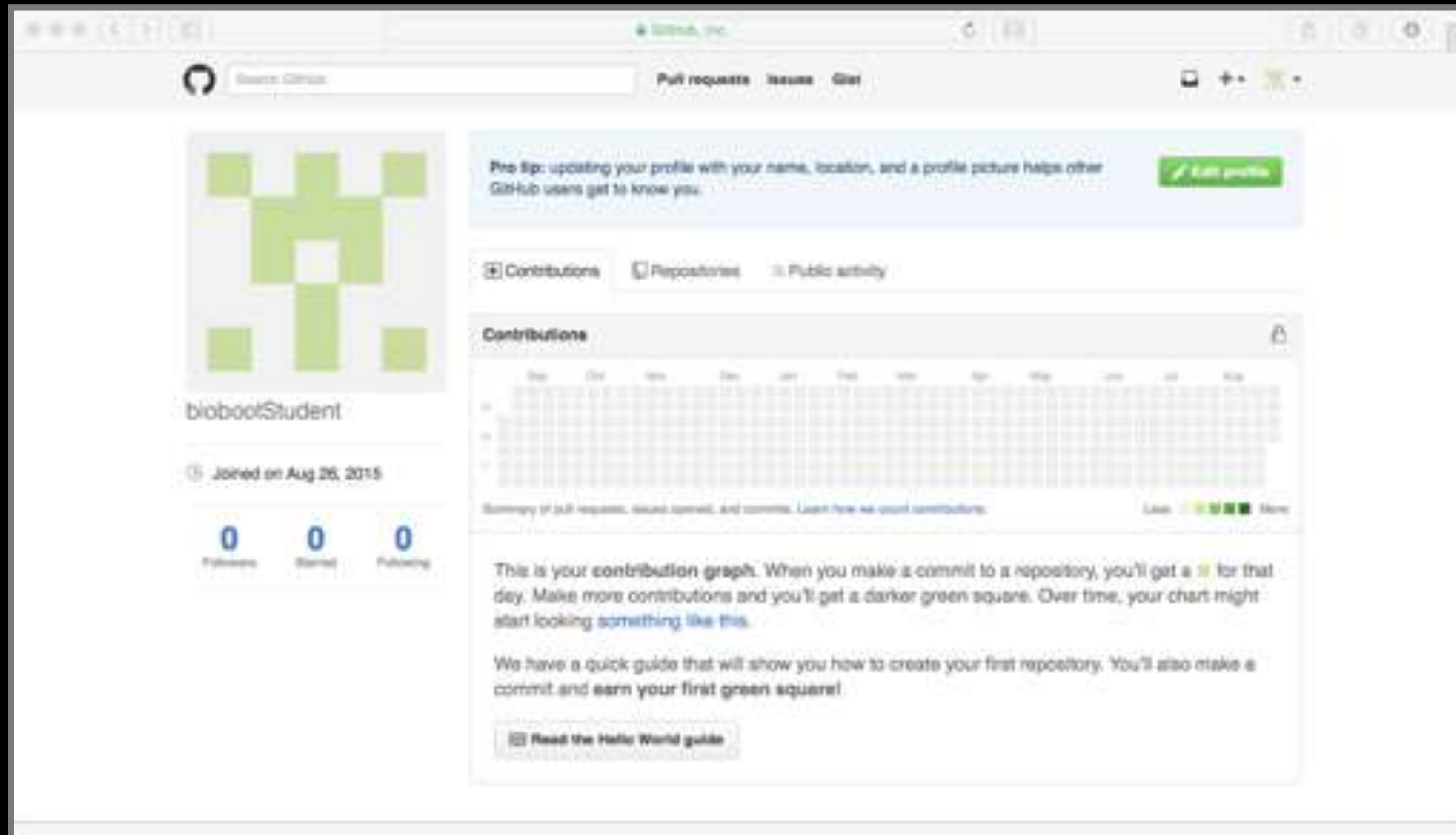
Each plan includes:

- Unlimited collaborators
- Unlimited public repositories
- ✓ Free setup
- ✓ HTTPS Protection
- ✓ Email support
- ✓ 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



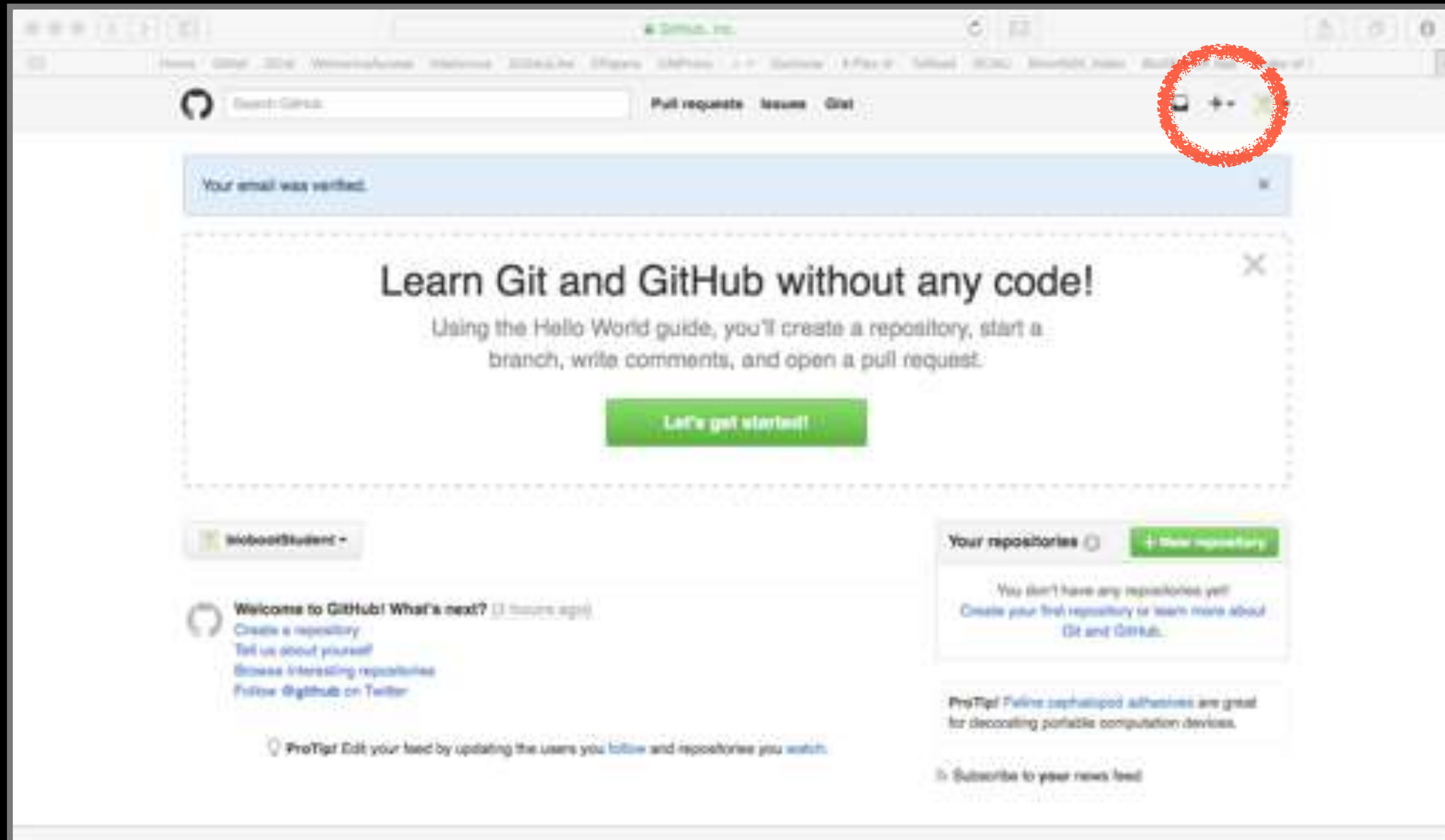
Connecting RStudio to GitHub

Create a **Personal Access Token** (PAT) on GitHub

See **section 4** of lab worksheet

Skip the hello-world tutorial

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



Name your repo

bimm143

Search GitHub Pull requests Issues Marketplace Explore

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner: bioboot - **bimm143** ✓

Repository name: **bimm143** ✓

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

Description (optional)

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

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

Add a README 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.

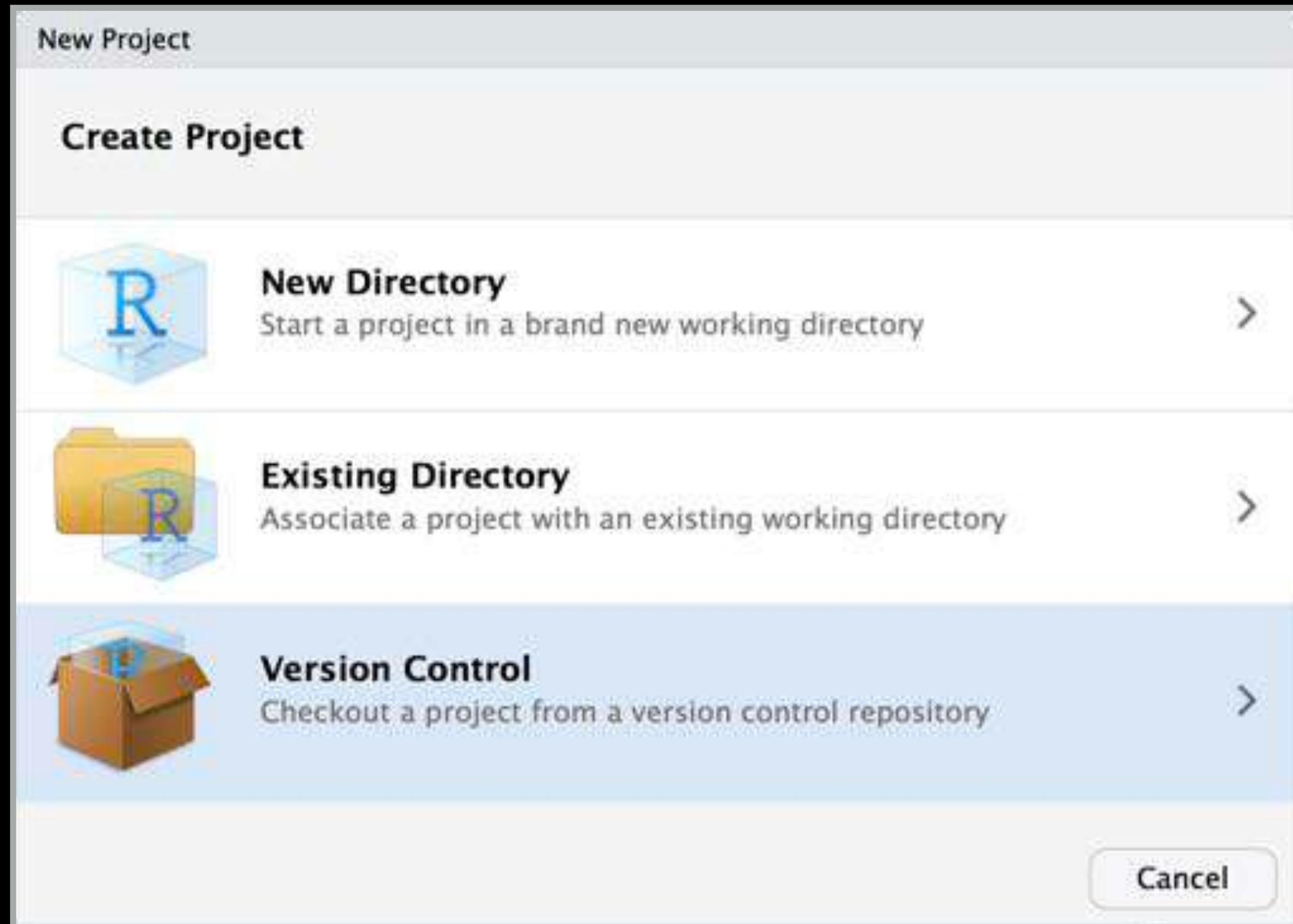
Add .gitignore: None ▾ Add a license: None ▾ ⓘ

Create repository **Create**

Copy the “**Clone**” HTTPS link

The screenshot shows the GitHub interface for the repository 'bioboot / bimm143'. At the top, there are navigation links for 'Pull requests', 'Issues', 'Marketplace', and 'Explore'. Below the repository name, there are buttons for 'Unwatch', 'Star', and 'Fork'. A secondary navigation bar includes 'Code', 'Issues', 'Pull requests', 'Projects', 'Wiki', 'Insights', and 'Settings'. The main content area shows 'No description, website, or topics provided.' and 'Add topics'. Below this, there are statistics for '1 commit', '1 branch', '0 releases', and '1 contributor'. A 'Clone or download' button is highlighted with a red circle. A dropdown menu is open, showing 'Clone with HTTPS' as the selected option, with the URL 'https://github.com/bioboot/bimm143.git' and a copy icon also circled in red. Other options in the dropdown include 'Use SSH', 'Open in Desktop', and 'Download ZIP'. The repository name 'bimm143' is displayed at the bottom of the page.


RStudio > New Project > **Version Control**



RStudio > New Project > Version Control

New Project

Back Clone Git Repository

 Repository URL: **GitHub Paste**

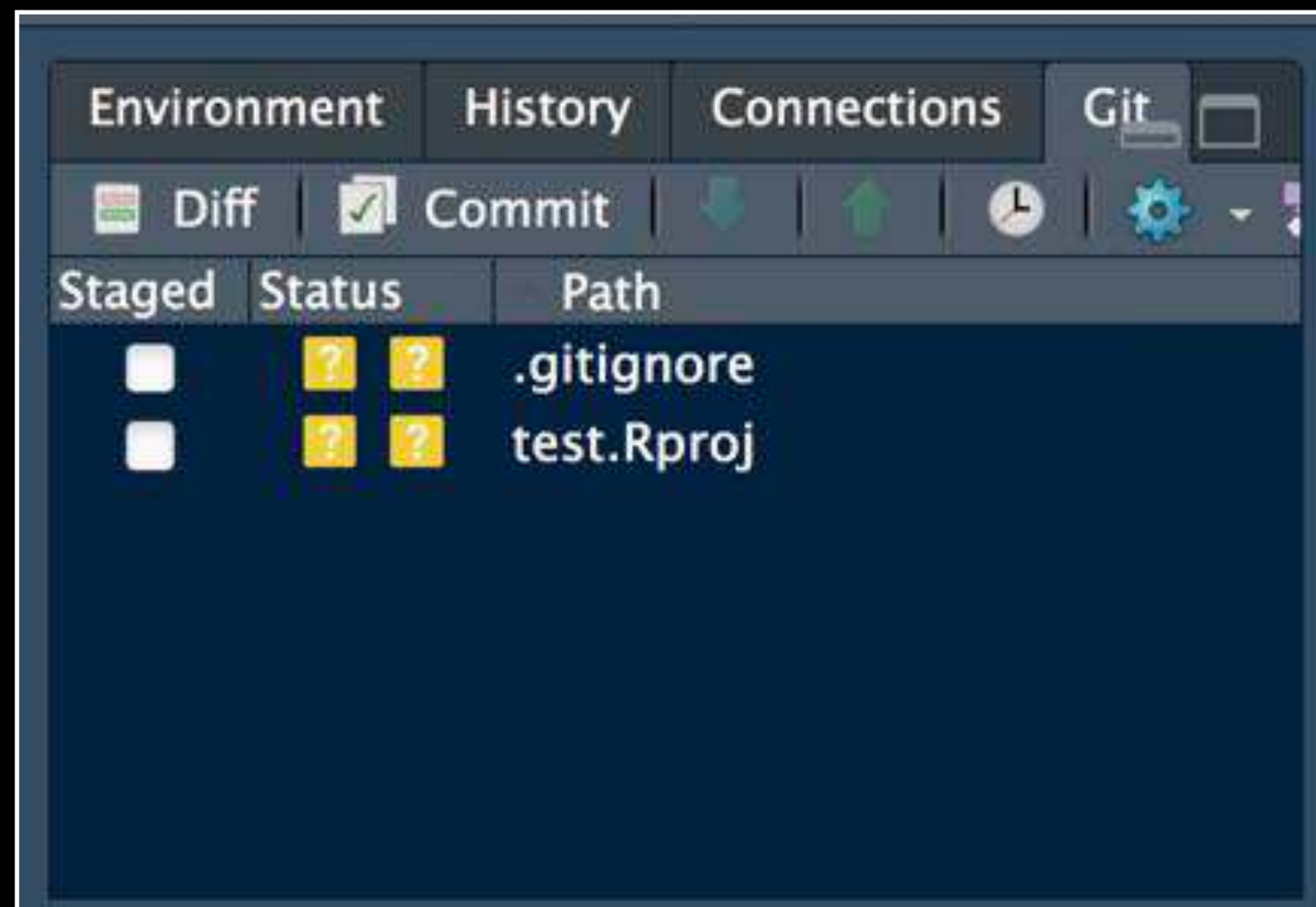
Project directory name:

Create project as subdirectory of:

Open in new session

Demo of *editing*, *adding* *committing* and *pushing*

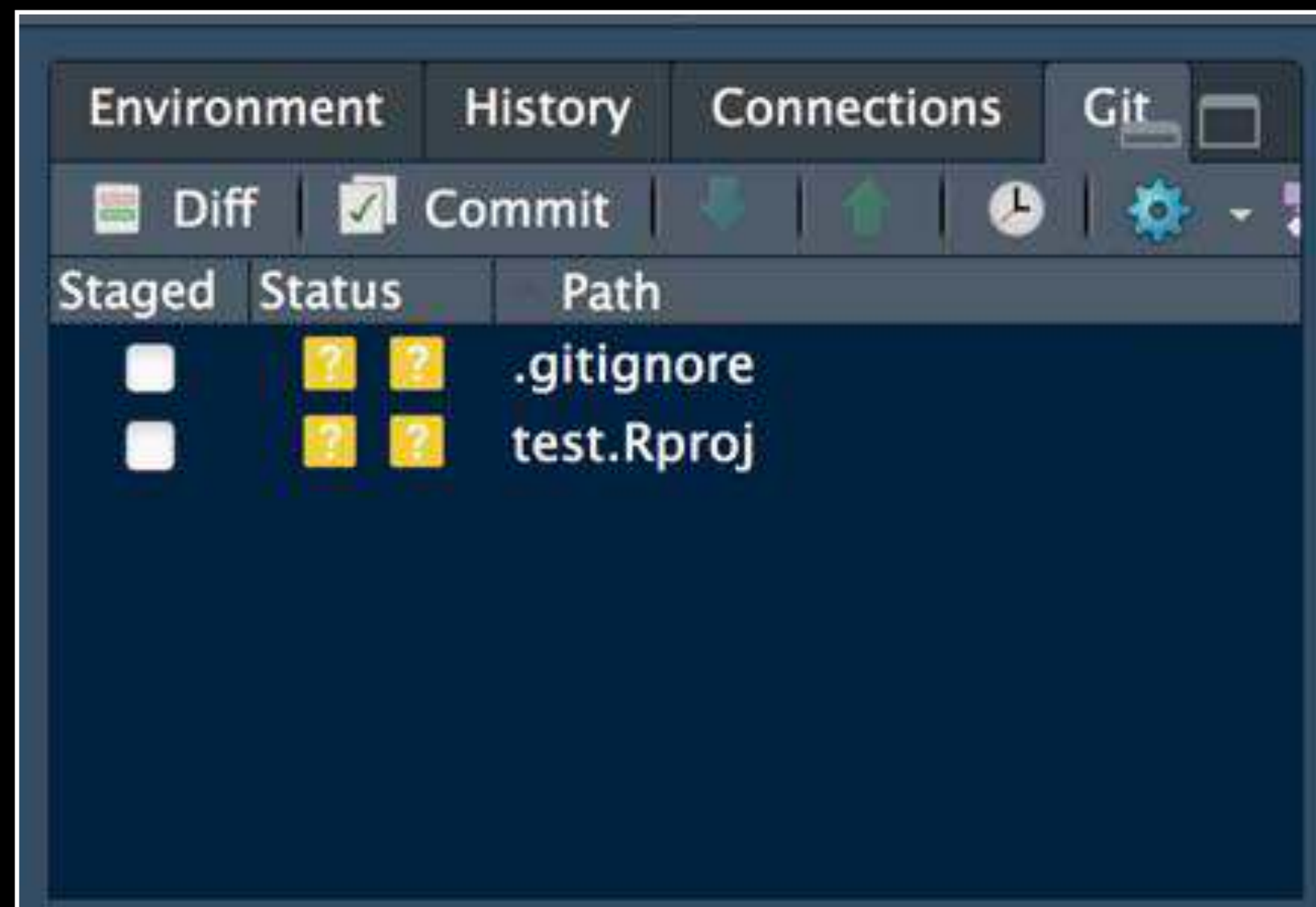
**Check if new Git tab
Appears in RStudio?**



**Now experiment editing the
README.md file in RStudio
and adding, committing and
pushing changes to GitHub
via this tab**

Demo of *editing*, *adding* *committing* and *pushing*

**Check if new Git tab
Appears in RStudio?**

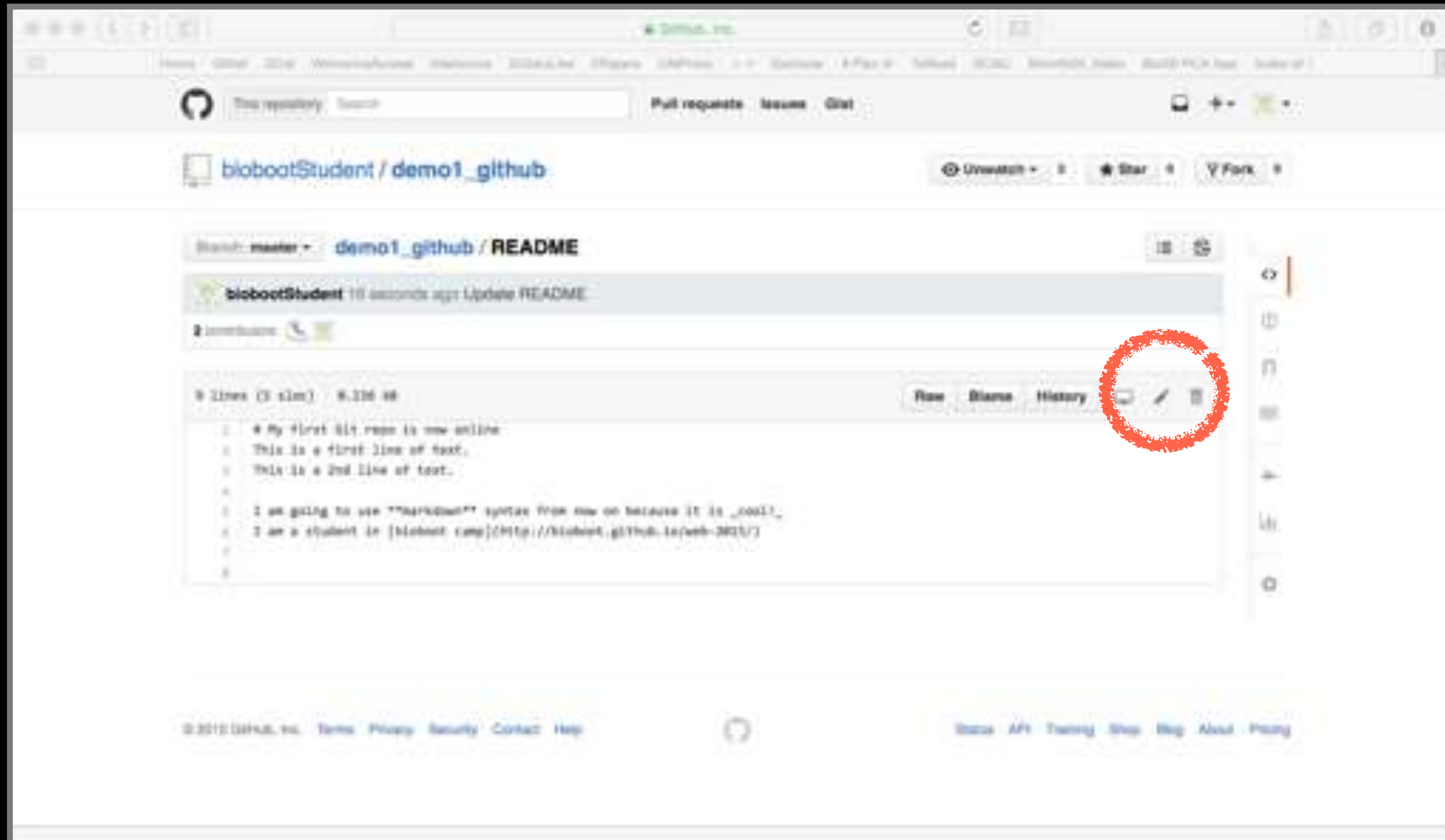


**Now experiment editing the
README.md file in RStudio
and adding, committing and
pushing changes to GitHub
via this tab**

**When you are ready copy your
different class directories/projects
to this new GitHub tracked folder**

Side-note: How to edit online

Specifically lets add some Markdown content



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.

The screenshot shows a GitHub repository page for 'Introduction to Bioinformatics'. The page features a blue header with the title 'Introduction to Bioinformatics' and a logo of a DNA double helix with a magnifying glass over the numbers '101' and '110'. Below the logo, there is a description: 'A demo site of students cool class web site'. A link 'View the Project on GitHub' points to 'bioboot/bioinformatics143-1'. The page also mentions 'This project is maintained by bioboot' and 'Hosted on GitHub Pages — Theme by undersitch'. On the right side, there is a section titled 'Bioinformatics Class' with a description: 'This is my repository for my Bioinformatics class from UC San Diego in STB'. Below this is an 'Index of Material' listing 14 classes, including 'Introductory Material: Working With R', 'Class 5 - Basic Data Exploration and Visualization in R HTML, MD, Rmd', 'Class 6 - Creating R Functions', 'Class 7 - R Packages, working with CRAN, and working with Bioconductor', 'Using R and Other Tools for Bioinformatics Analysis', 'Class 8 - An Introduction to Machine Learning (Hierarchical Clustering)', 'Class 9 - Analyzing High Dimensional Datasets and Unsupervised Learning', 'Class 11 - Structural Bioinformatics: Analyzing Protein Structure and Function', 'Class 12 - Drug Discovery: Techniques and Analysis', 'Class 13 - Genome Informatics and High Throughput Sequencing (NGS, RNA-Seq, and FastQC)', and 'Class 14 - Transcriptomics and RNA-Seq Analysis'.

Here I: (1) forked Serina's Repo, (2) Chose the "minimal" theme, (3) Edited _config.yml (adding logo and title)

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!



The End!

<http://thegrantlab.org>

A microscopic image showing clusters of small, spherical, multi-colored particles. The particles are primarily purple and pink, with some yellow and blue ones interspersed. They are arranged in dense, irregular groups against a dark, teal background.

Thank You!

<http://thegrantlab.org>

Reference Slides

Command Line GIT

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’**

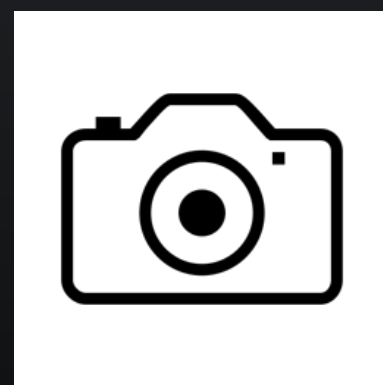
Example Git workflow



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

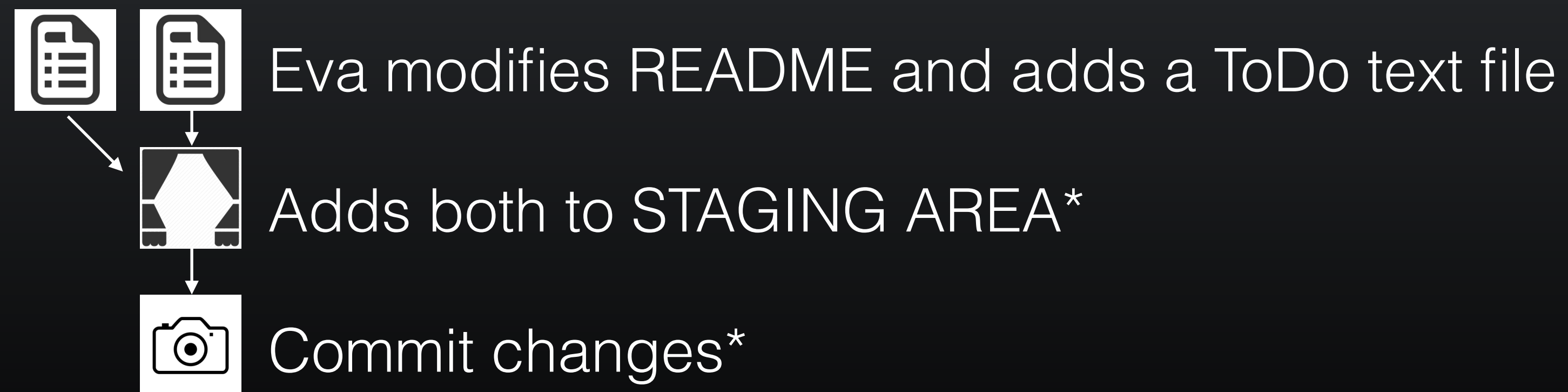
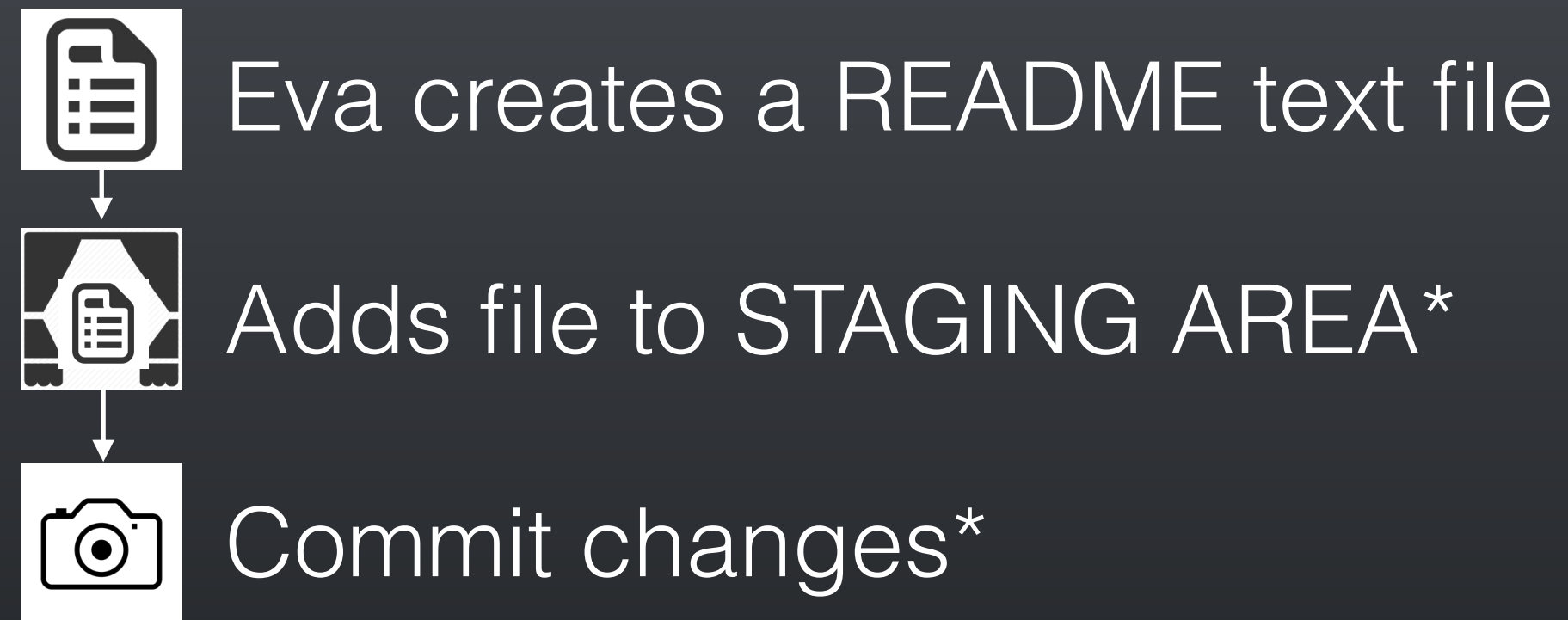


Adds file to STAGING AREA*
(tracked and ready to take a snapshot)



Commit changes*
(records snapshot of staged files!)

Example Git workflow



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(+)
```



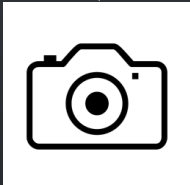
```
# create mode 100644 ToDo
```



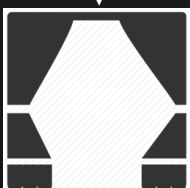
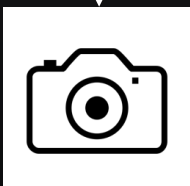
```
> git status
```

```
# 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)

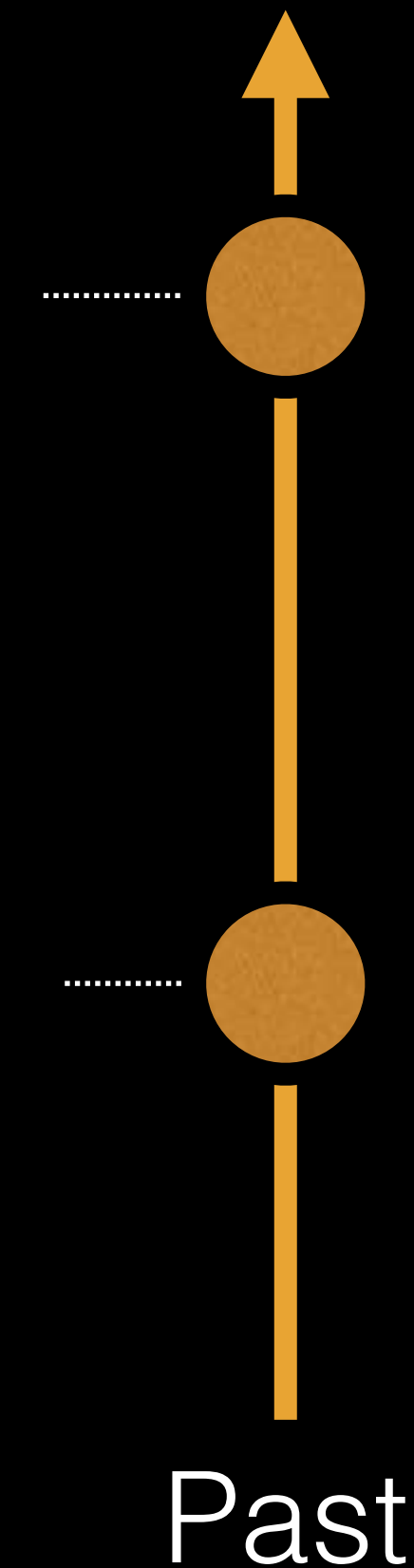
> **git log**

```
# 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
#
```

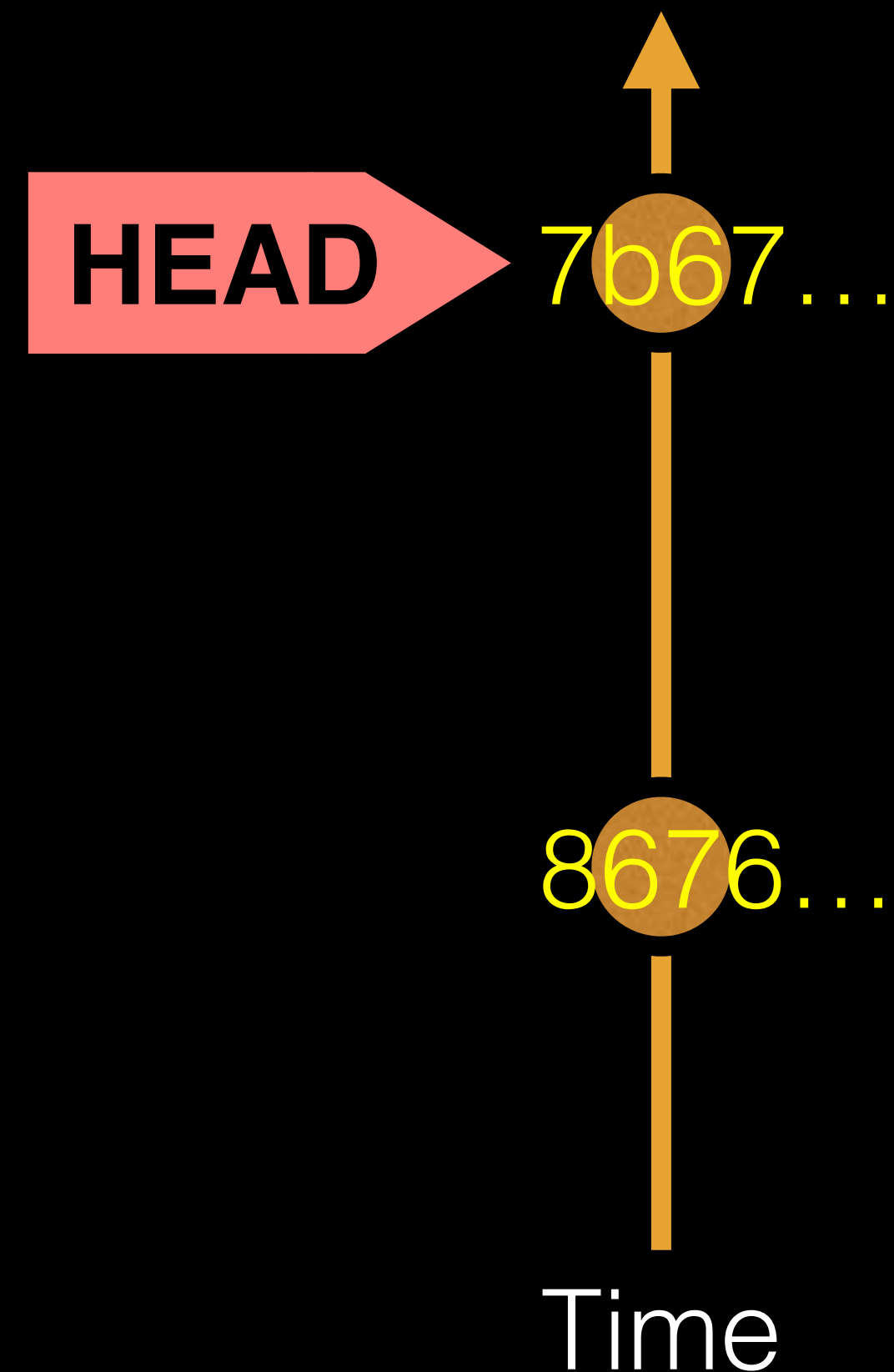
git log: Timeline history of snapshots (*i.e.* commits)

> **git log**

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



Side-Note: Git history is akin to a graph

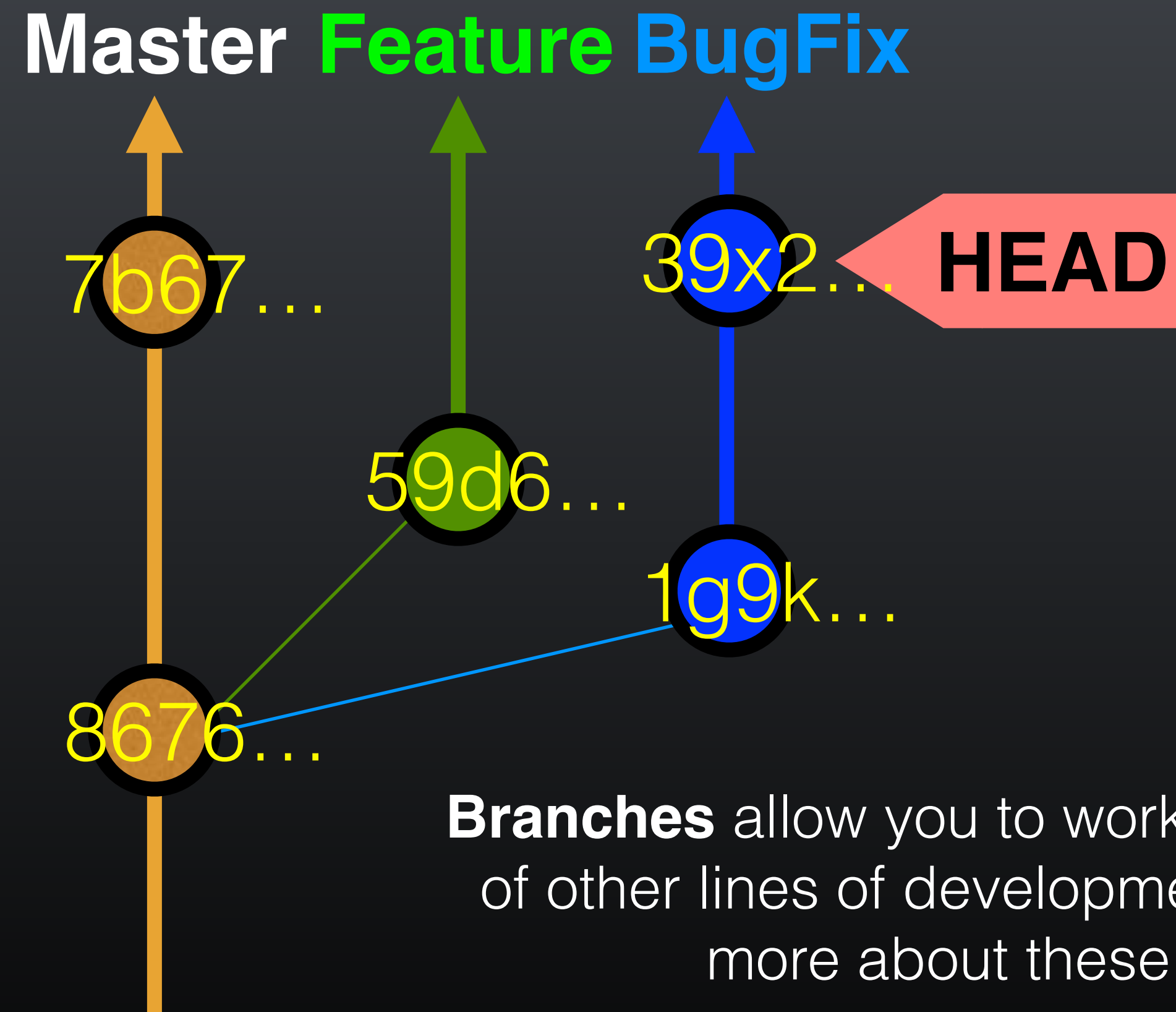


Nodes are **commits** labeled by their unique '**commit ID**'.

(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**



Branches allow you to work independently of other lines of development we will talk more about these later!

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

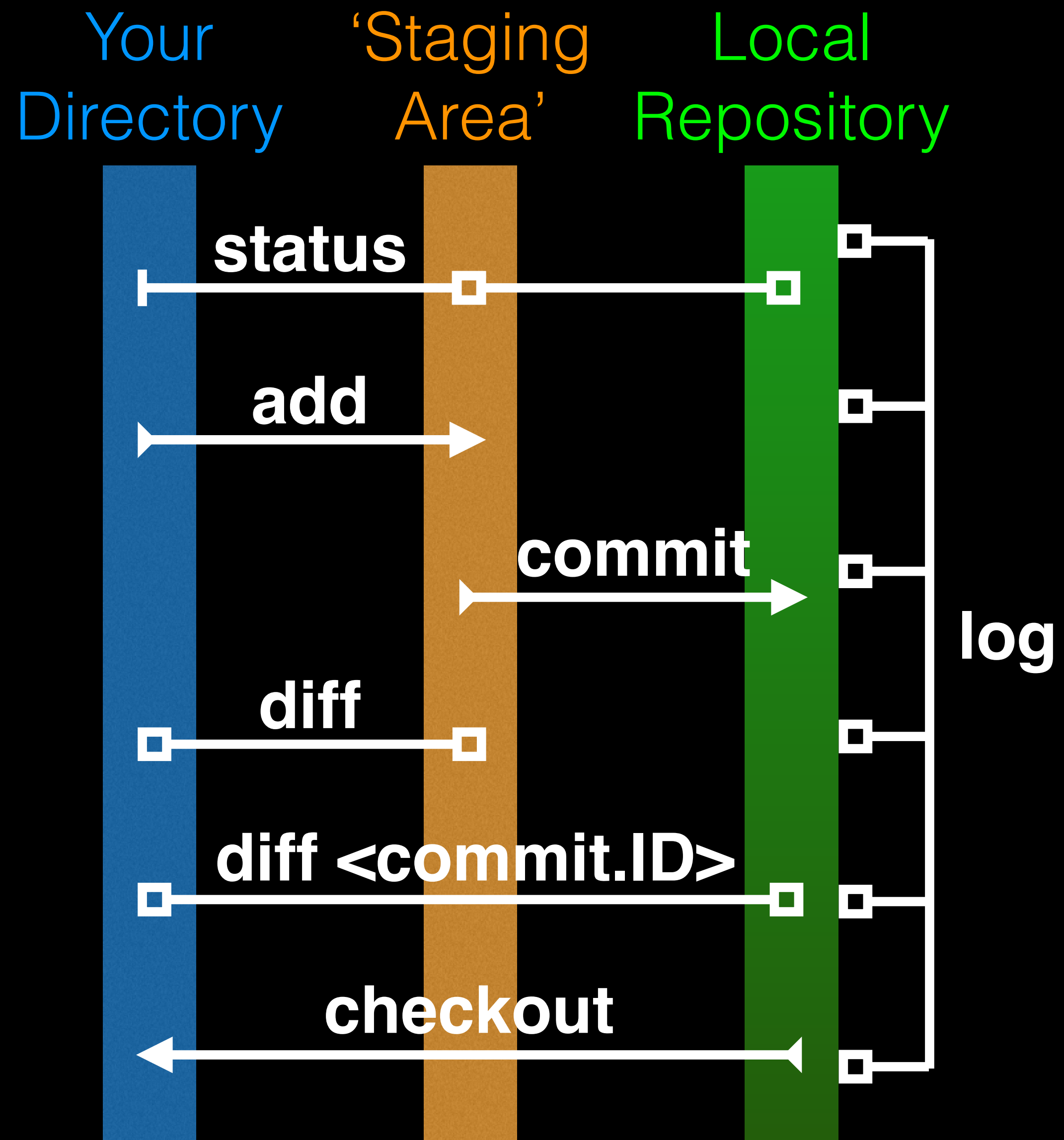
> **git add <filename>** # Tell Git which files to track/stage

> **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



git diff: Show changes between commits

```
> git diff 8676 7b67
```

```
# diff --git a/README b/README
# index 73bc85a..67bd82c 100644
# --- a/README
# +++ b/README
# @@ -1,2 @@
# 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..14fbd56
# --- /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 14fbd56..0000000
# --- a/ToDo
# +++ /dev/null
# @@ 1 +0,0 @@
# -Learn git basics
```

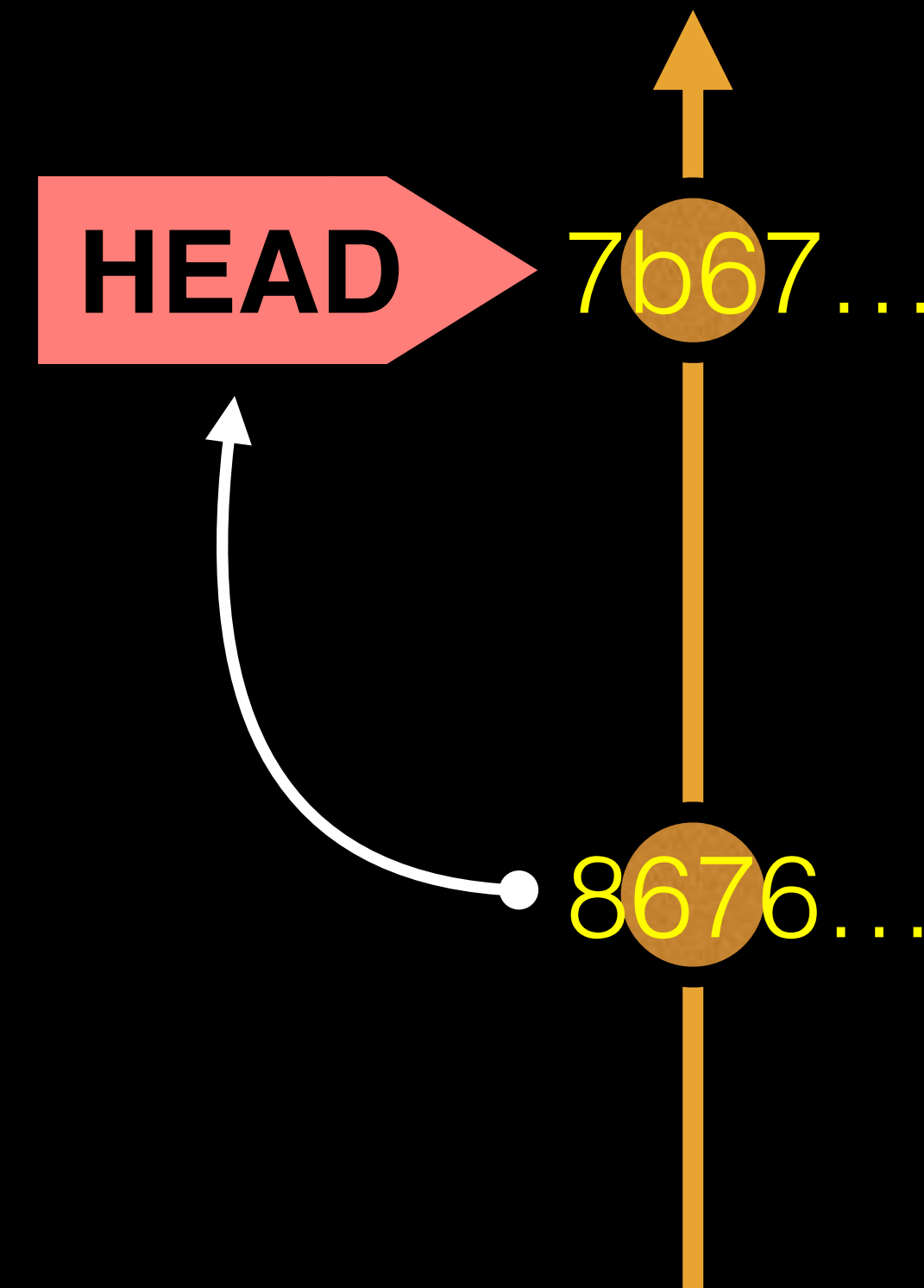


git diff: Show changes between commits

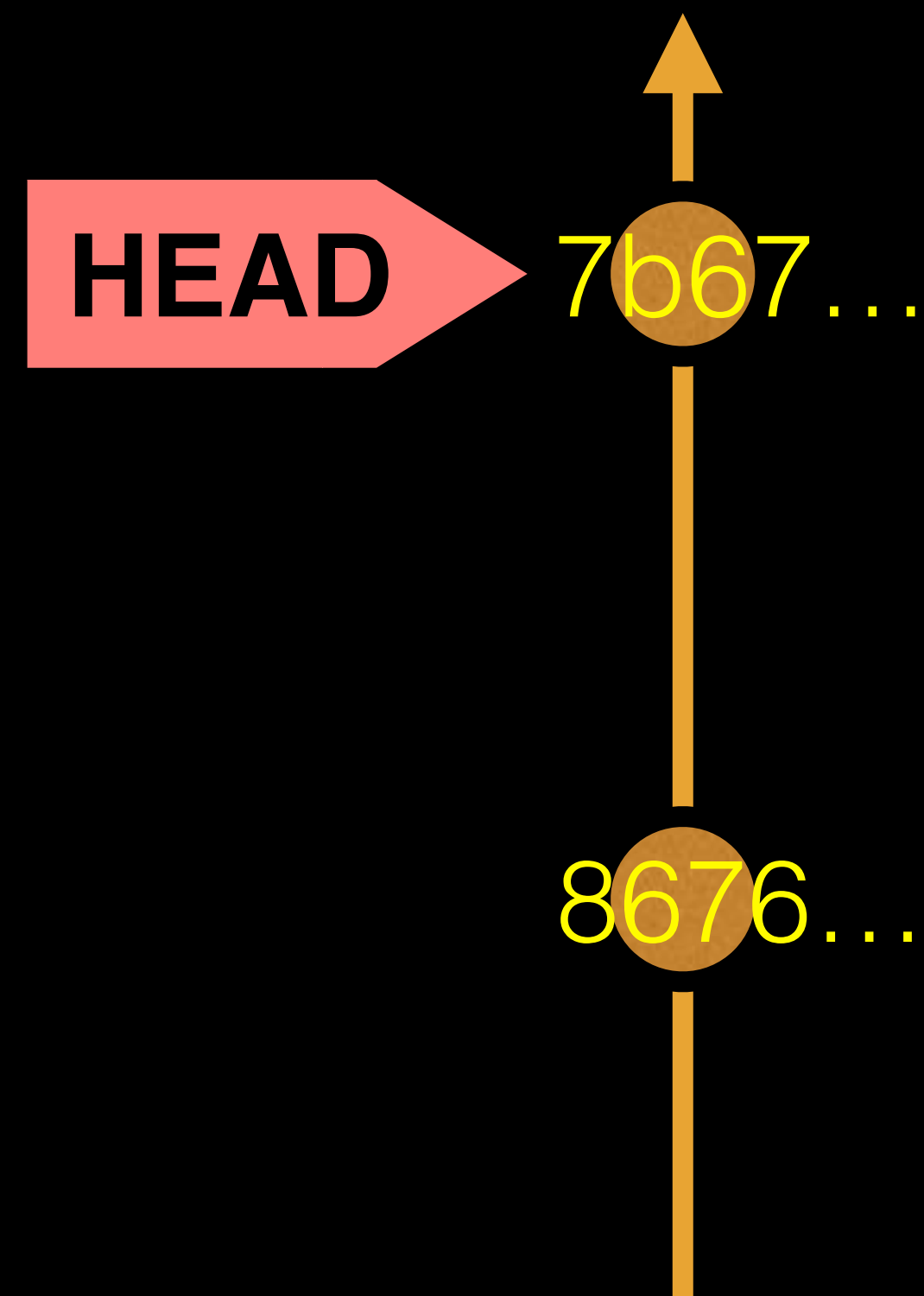
> **git diff 8676** **## Difference to current HEAD position!**

```
# 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..14fbd56
# --- /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>
```

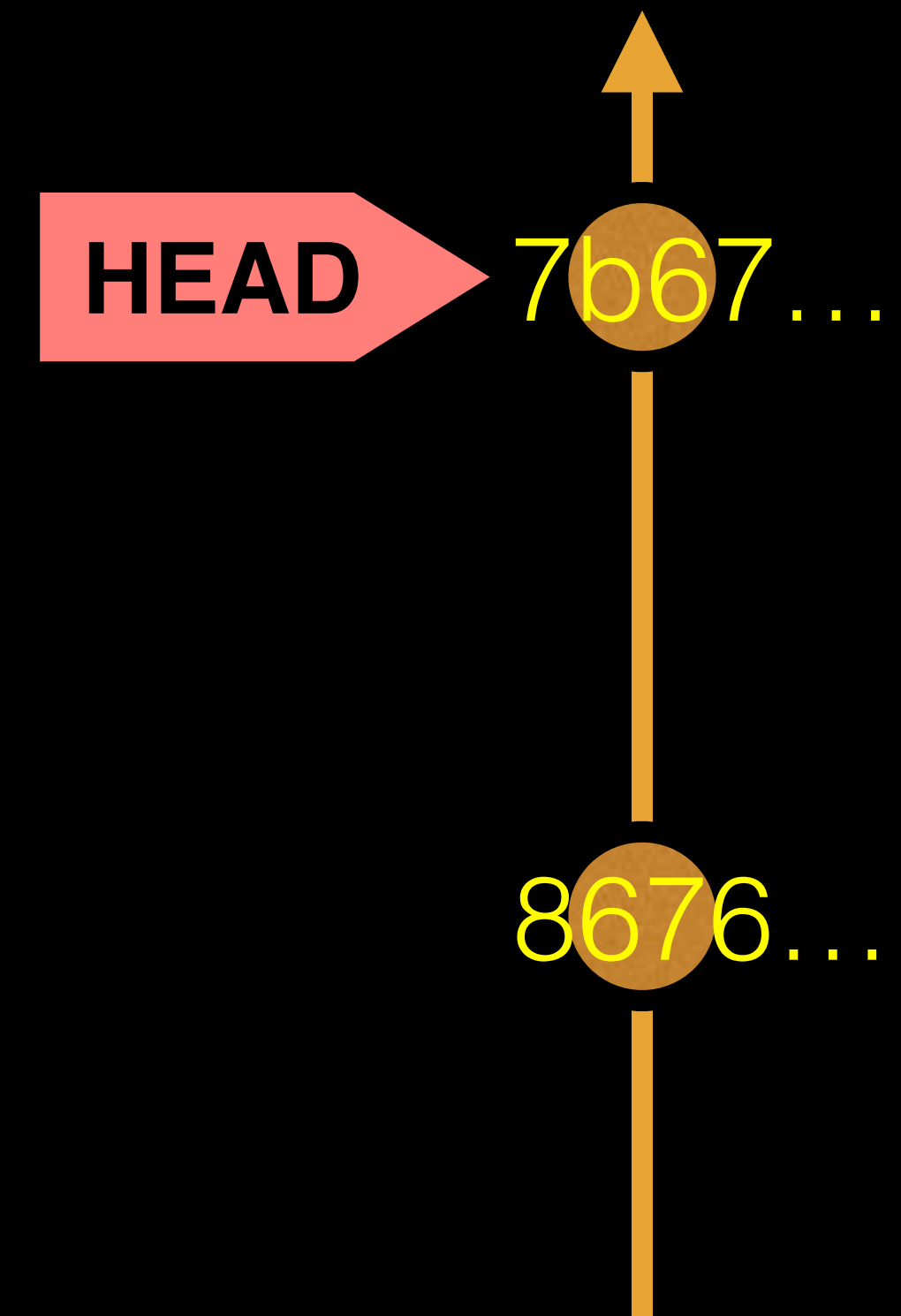
git checkout: Moves HEAD

> **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 86768**

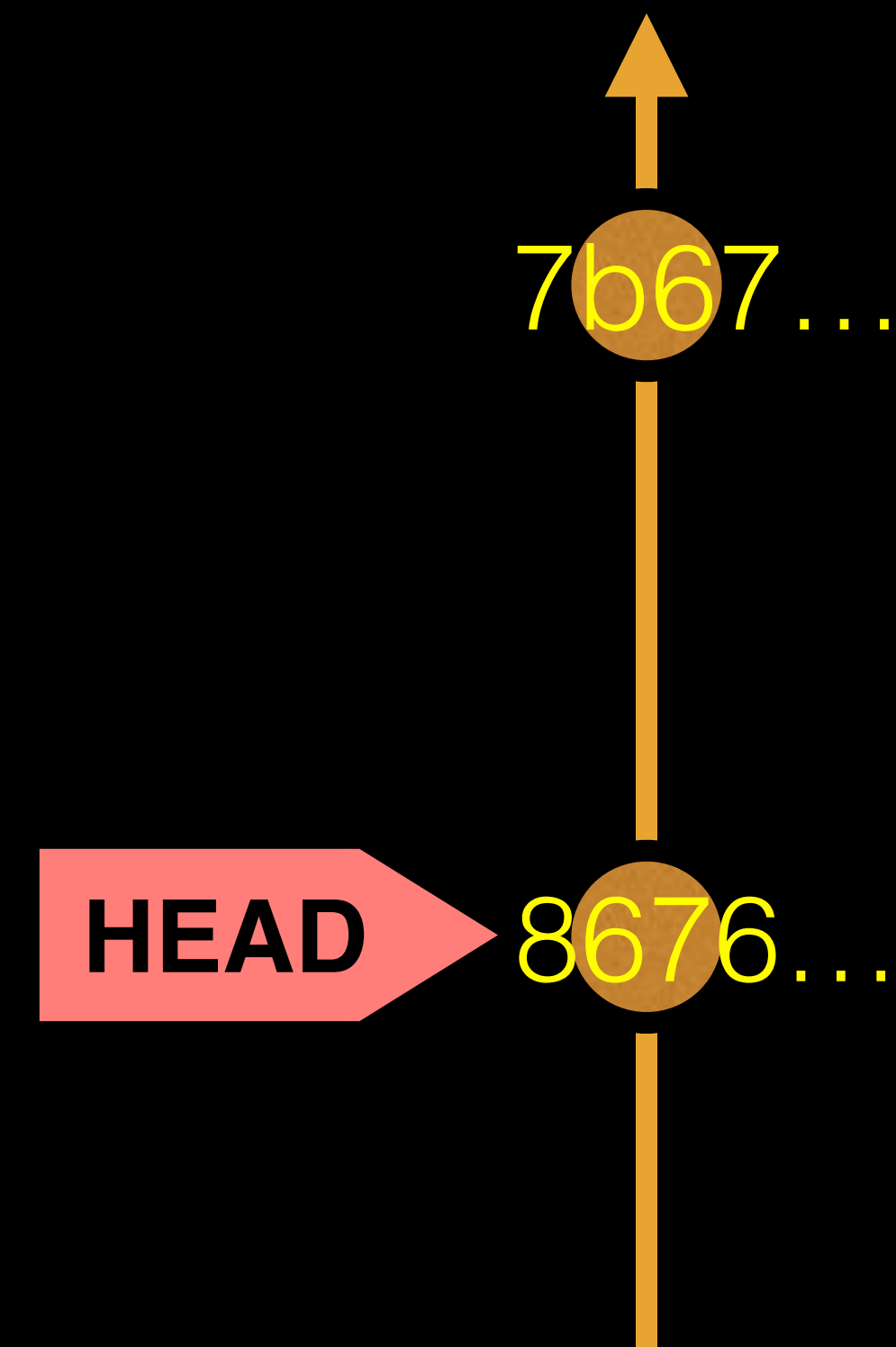
You are in 'detached HEAD' state...<cut>...
HEAD is now at 8676840... Create a README file

> **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**

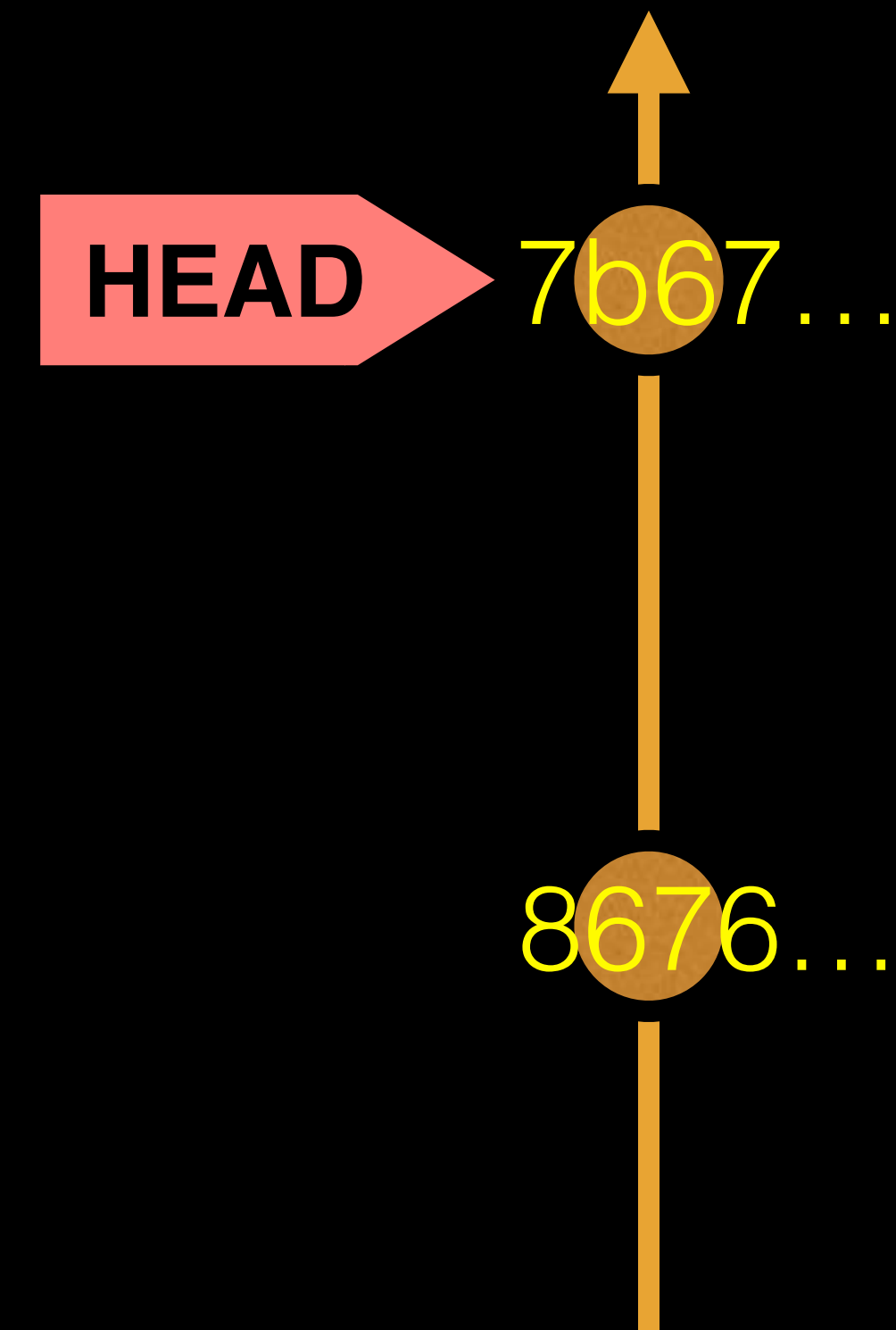
```
# Previous HEAD position was 8676840... Create a README file  
# Switched to branch 'master'
```

> **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>