

# Reproducible Research with R Markdown, Git, and Github

Kylie A. Bemis

Northeastern University  
Khoury College of Computer Sciences



Northeastern University

# Goals for this session

- R Markdown
- Git
- Github

# R MARKDOWN

# How to report reproducible analyses?

- Goal 1: Turn statistical analyses into reports
  - ◆ High-quality documents, presentations, dashboards, etc.
- Goal 2: Analyses should be *reproducible*
  - ◆ Update report results when analysis or data changes

# R + Markdown

## R

- R code chunks
- Inline results and plots
- Re-run to update report

## Markdown

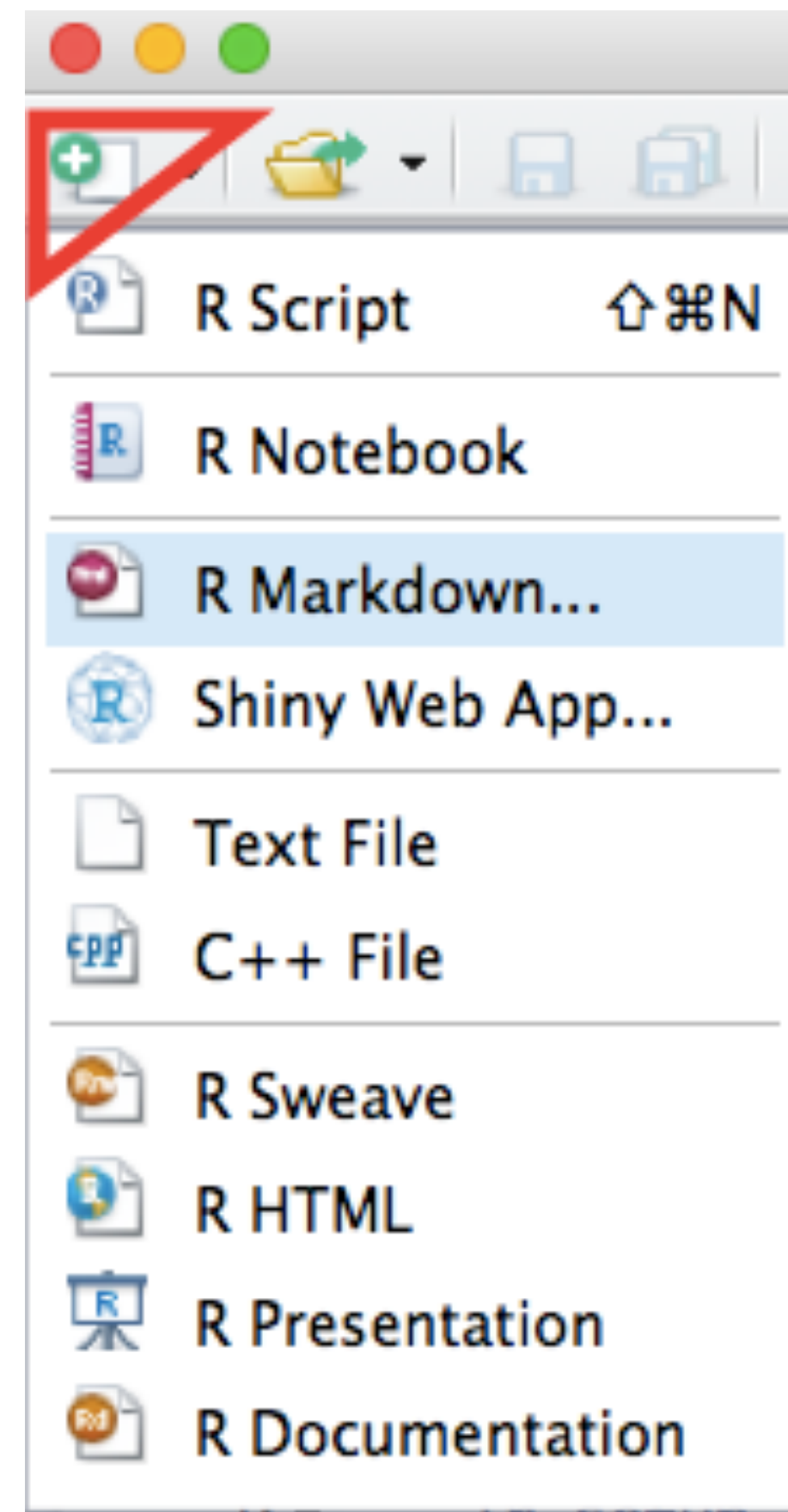
- Markup language
- Easily format text
- Create PDF, HTML, etc.

# R Markdown

- Turn R code into high-quality reports
  - ◆ Markdown-formatted text
  - ◆ R code chunks with inline results
  - ◆ Compile a professional-quality document
- Recompile to produce updated report
  - ◆ All code is re-run to create new report
  - ◆ Reports are always up-to-date with newest data

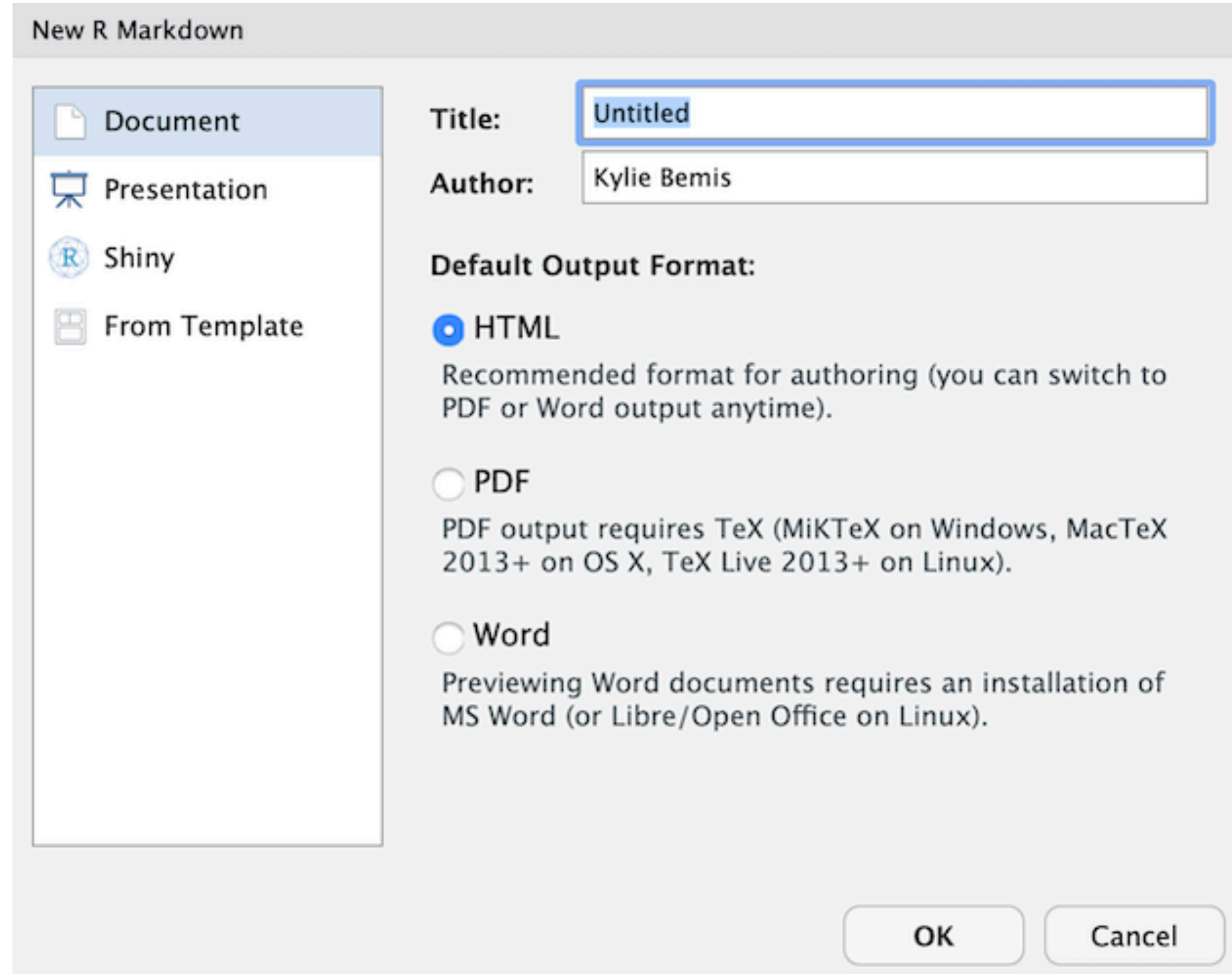
# Create an R Markdown source file

Easy to create and author R Markdown from RStudio



# Choose an R Markdown format

Select an output format and title (and author)



The screenshot shows the 'New R Markdown' dialog box. On the left, there is a list of options: 'Document' (selected), 'Presentation', 'Shiny', and 'From Template'. On the right, there are fields for 'Title' (containing 'Untitled') and 'Author' (containing 'Kylie Bemis'). Below these fields, the 'Default Output Format' is set to 'HTML', which is the recommended format for authoring. The 'PDF' and 'Word' options are also visible, each with a brief description of their requirements. At the bottom right, there are 'OK' and 'Cancel' buttons.

New R Markdown

☒ Document  
☐ Presentation  
☐ Shiny  
☐ From Template

Title:

Author:

Default Output Format:

☒ HTML  
Recommended format for authoring (you can switch to PDF or Word output anytime).

☐ PDF  
PDF output requires TeX (MiKTeX on Windows, MacTeX 2013+ on OS X, TeX Live 2013+ on Linux).

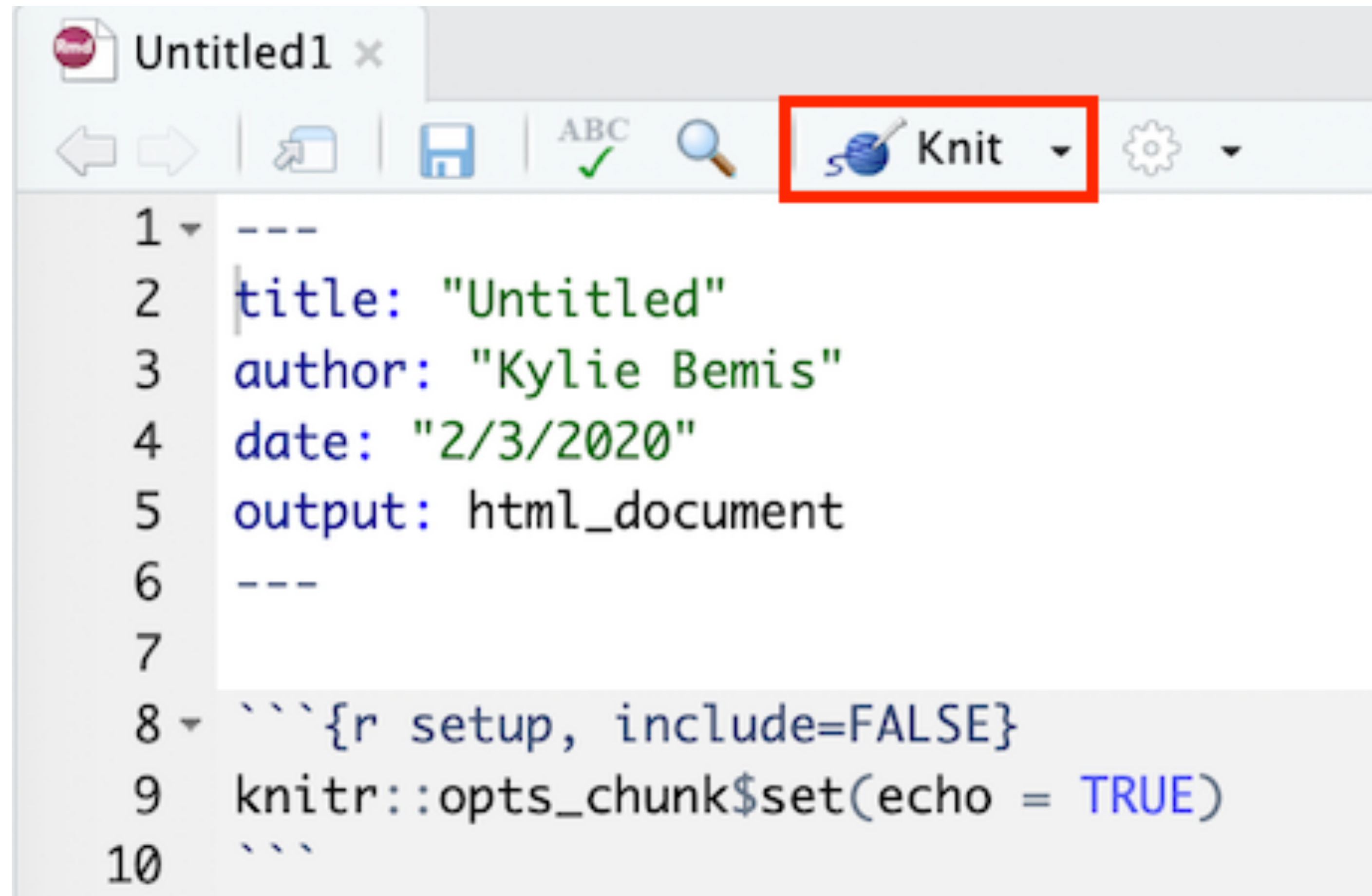
☐ Word  
Previewing Word documents requires an installation of MS Word (or Libre/Open Office on Linux).

OK Cancel



# Compile an R Markdown document

Click the "Knit" button to compile into a finished document



The screenshot shows an R Markdown editor window titled "Untitled1". The toolbar at the top includes icons for navigation, saving, and running. The "Knit" button, which features a blue globe icon and the text "Knit", is highlighted with a red rectangular box. Below the toolbar, the R Markdown document content is displayed, showing a YAML header and an R code chunk.

```
1 ---  
2 title: "Untitled"  
3 author: "Kylie Bemis"  
4 date: "2/3/2020"  
5 output: html_document  
6 ---  
7  
8 ```{r setup, include=FALSE}  
9 knitr::opts_chunk$set(echo = TRUE)  
10 ```
```

# Markdown formatting

## Text formatting

---

`*italic*` or `_italic_`  
`**bold**`    `__bold__`  
``code``  
superscript<sup>2</sup> and subscript<sub>2</sub>

## Headings

---

`# 1st Level Header`

`## 2nd Level Header`

`### 3rd Level Header`

## Lists

---

`* Bulleted list item 1`

`* Item 2`

`* Item 2a`

`* Item 2b`

`1. Numbered list item 1`

`1. Item 2. The numbers are incremented automatically in the output.`

## Text formatting

*italic* or *italic* **bold** **bold** code superscript<sup>2</sup> and subscript<sub>2</sub>

## Headings

### 1st Level Header

### 2nd Level Header

### 3rd Level Header

## Lists

- Bulleted list item 1

- Item 2

- Item 2a

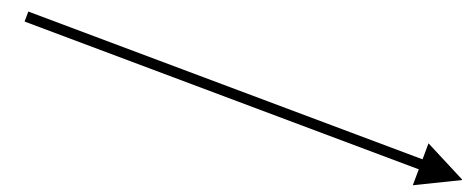
- Item 2b

1. Numbered list item 1

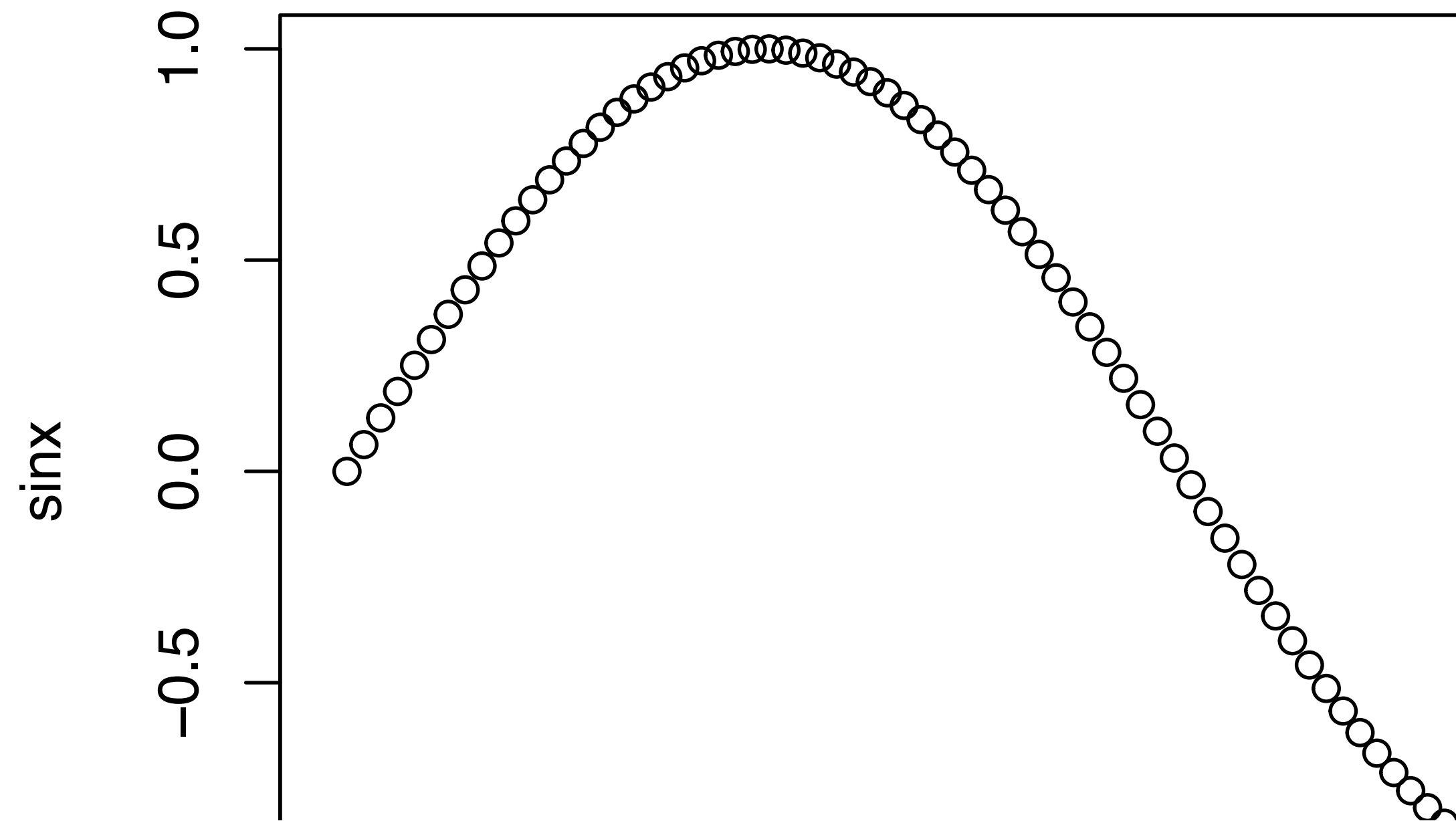
2. Item 2. The numbers are incremented automatically in the output.

# R code chunks

```
```{r}  
x <- seq(from=0, to=2*pi, length.out=100)  
sinx <- sin(x)  
plot(sinx ~ x)  
```
```



```
x <- seq(from=0, to=2*pi, length.out=100)  
sinx <- sin(x)  
plot(sinx ~ x)
```

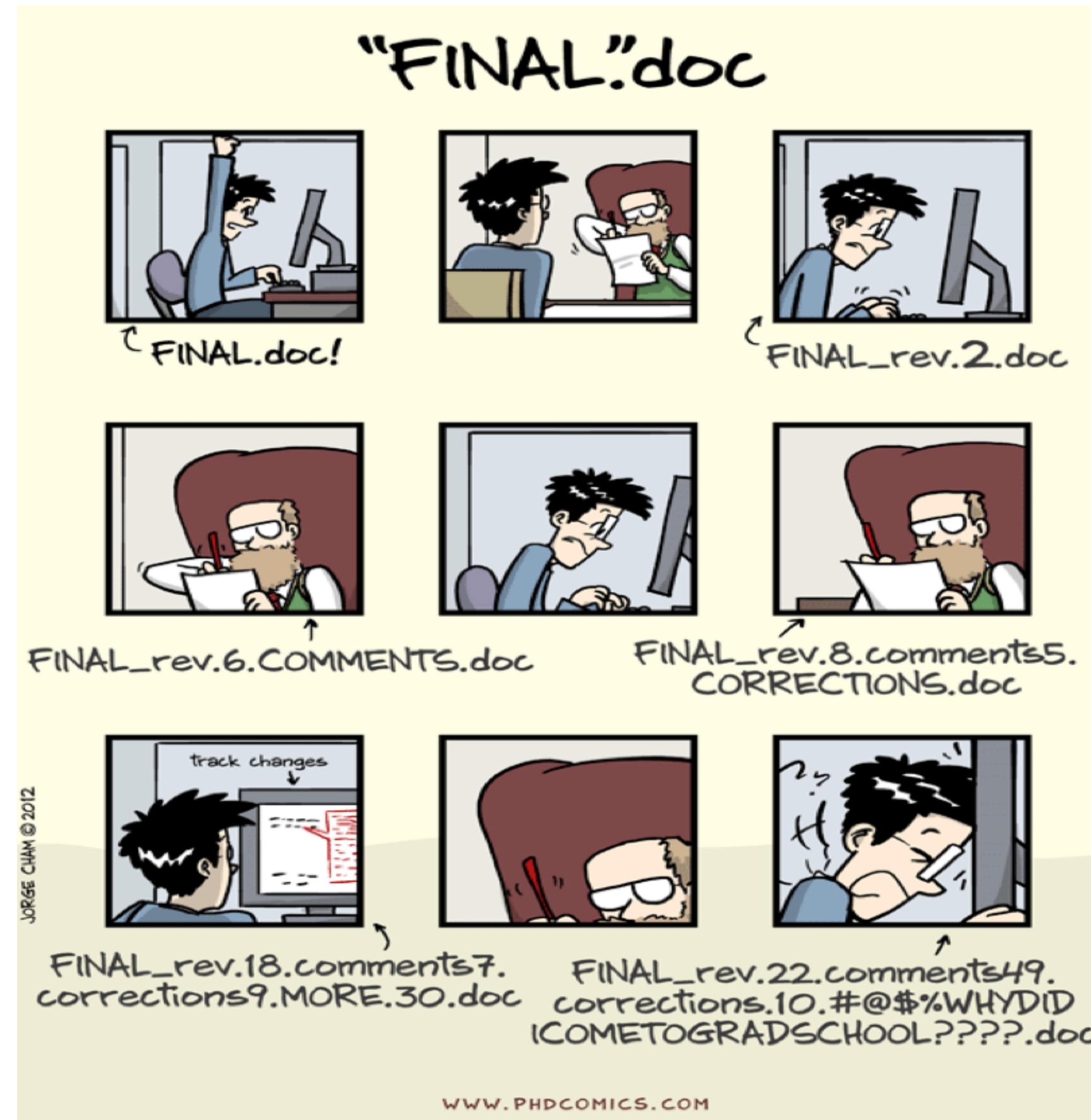


# Advanced R Markdown

- Customize code chunks
  - ◆ Hide echoed code or results
  - ◆ Set dimensions of plots
  - ◆ And much, much more...
- R Markdown cheat sheet:
  - ◆ <https://www.rstudio.com/wp-content/uploads/2016/03/rmarkdown-cheatsheet-2.0.pdf>

# GIT AND GITHUB

# How do you manage files in a project?

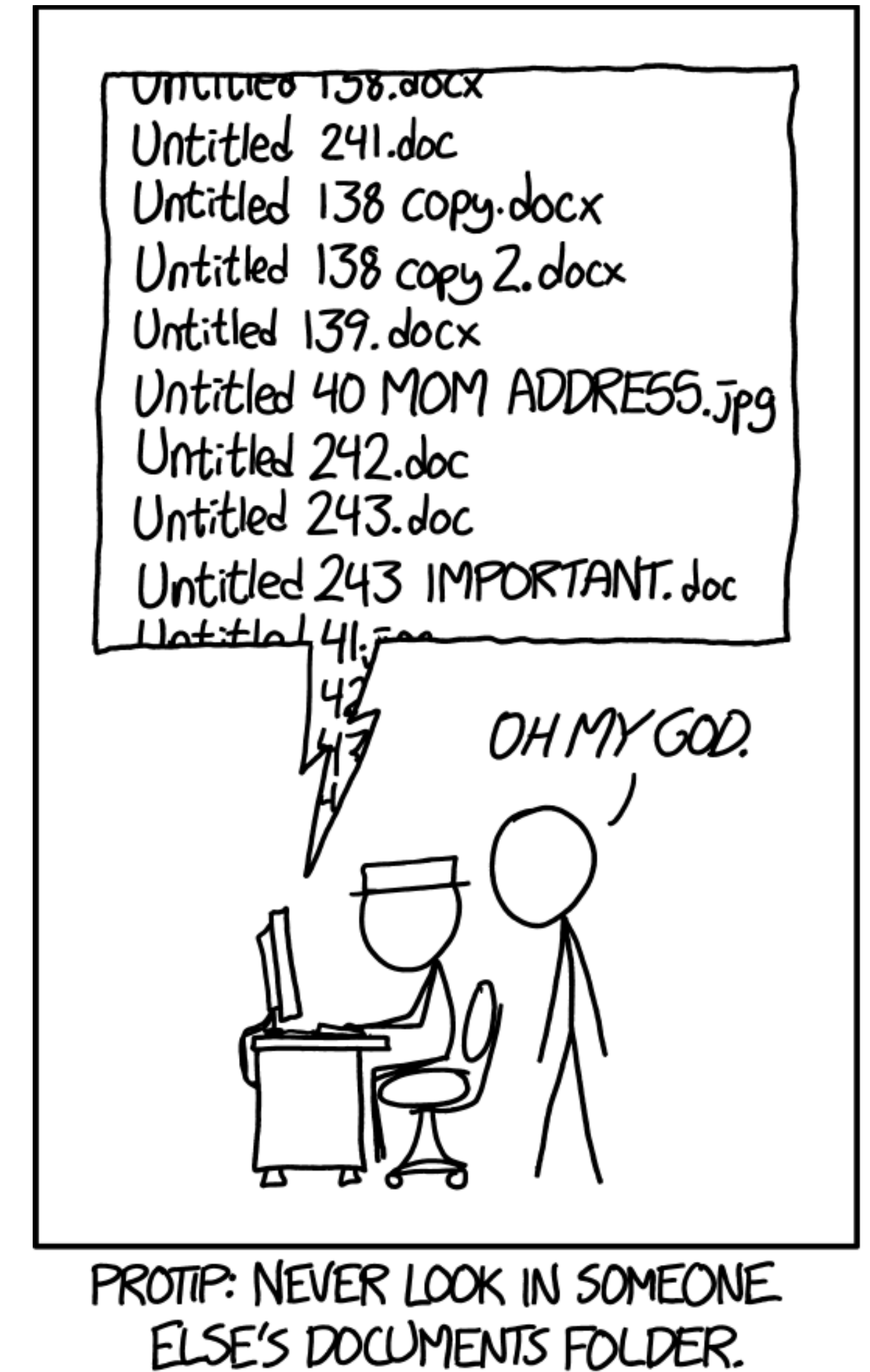


title: "notFinal.doc" - originally published 10/12/2012



# Version control

- Complex projects produce many files
- Need to **track changes** and **versions**
- Need to **share with collaborators**



# Goals of version control

- Track changes made to a project
  - ◆ Track **changes** across multiple files
  - ◆ Track **creation** and **deletion** of files
  - ◆ **Revert** and **merge** changes as necessary
- Allow multiple **branches** of progress
- **Synchronize work** with collaborators

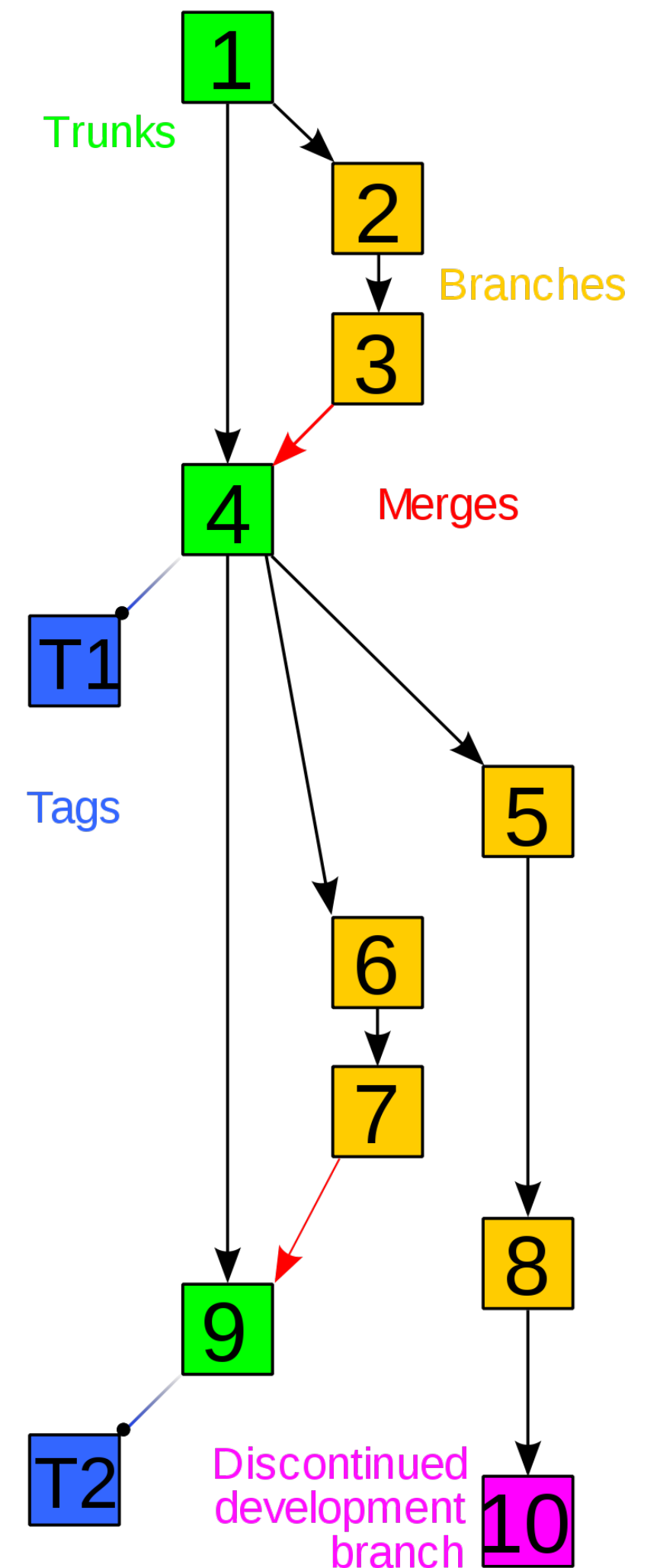


# Vocabulary

- A **repository** (“repo”) stores a project tracked by version control and its history
- A **commit** is a snapshot of a set of *changes*
- The project *head* is the most recent commit
- Changes can be **pushed** and **pulled** from one repository to another

# History and branches

- Revisions depend on earlier revisions
  - ◆ Each revision is linked to the revisions it depends on
- Progress may *fork* into separate **branches**
  - ◆ Develop new features or prepare bug patches
  - ◆ **Merge changes** back to the *trunk* or “main” branch
- Project history forms a *graph*



# Git and Github

- **Git** is a popular version control system
  - ◆ Distributed version control system
  - ◆ Repo is mirrored on each developer's machine
  - ◆ No need to rely on a central server
- **Github** hosts online Git repositories
  - ◆ Free hosting of open-source projects
  - ◆ Share work with collaborators

# Installing Git

- First check if Git is already installed
- **macOS** download:
  - ◆ <https://git-scm.com/download/mac>
- **Windows** download:
  - ◆ <https://gitforwindows.org>

# Setting up Git

- Git needs to know who is making changes
- Configure your credentials:
  - ◆ `git config --global user.name <your name>`
  - ◆ `git config --global user.email <your email>`
  - ◆ `git config --global --list`

# Using Git

- Any directory with a git history is a repo
- Initialize a git repo in a directory:
  - ◆ Navigate to a directory
  - ◆ `git init`
- Any files in the repo can now be tracked

# Using Git and Github

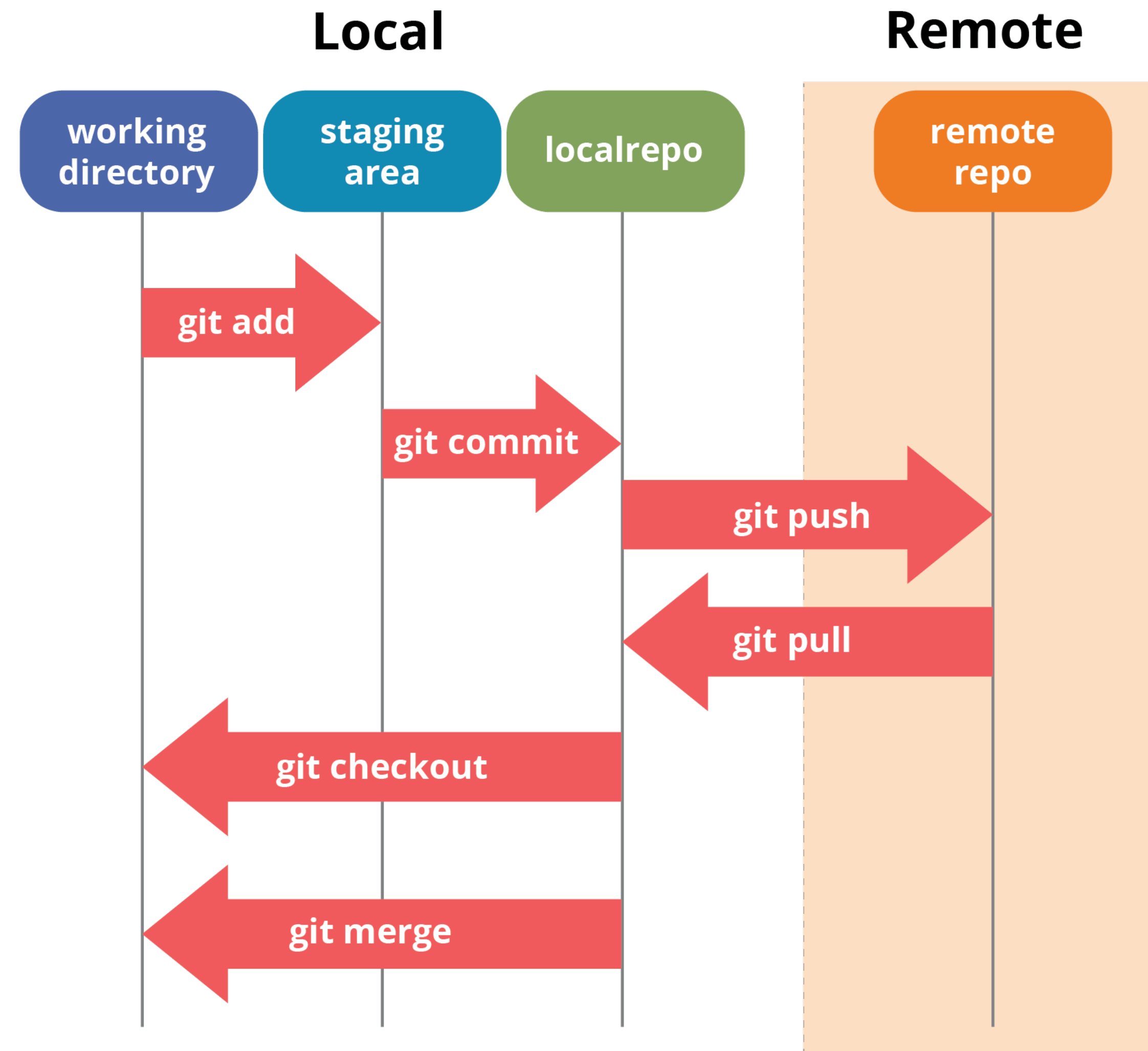
- Share work on a remote repository
- Create a new repo on Github
- Clone the repo locally
  - ◆ Copy the web URL from the Github repo page
  - ◆ `git clone <URL>`
- Work locally and push to Github

# Understanding Git

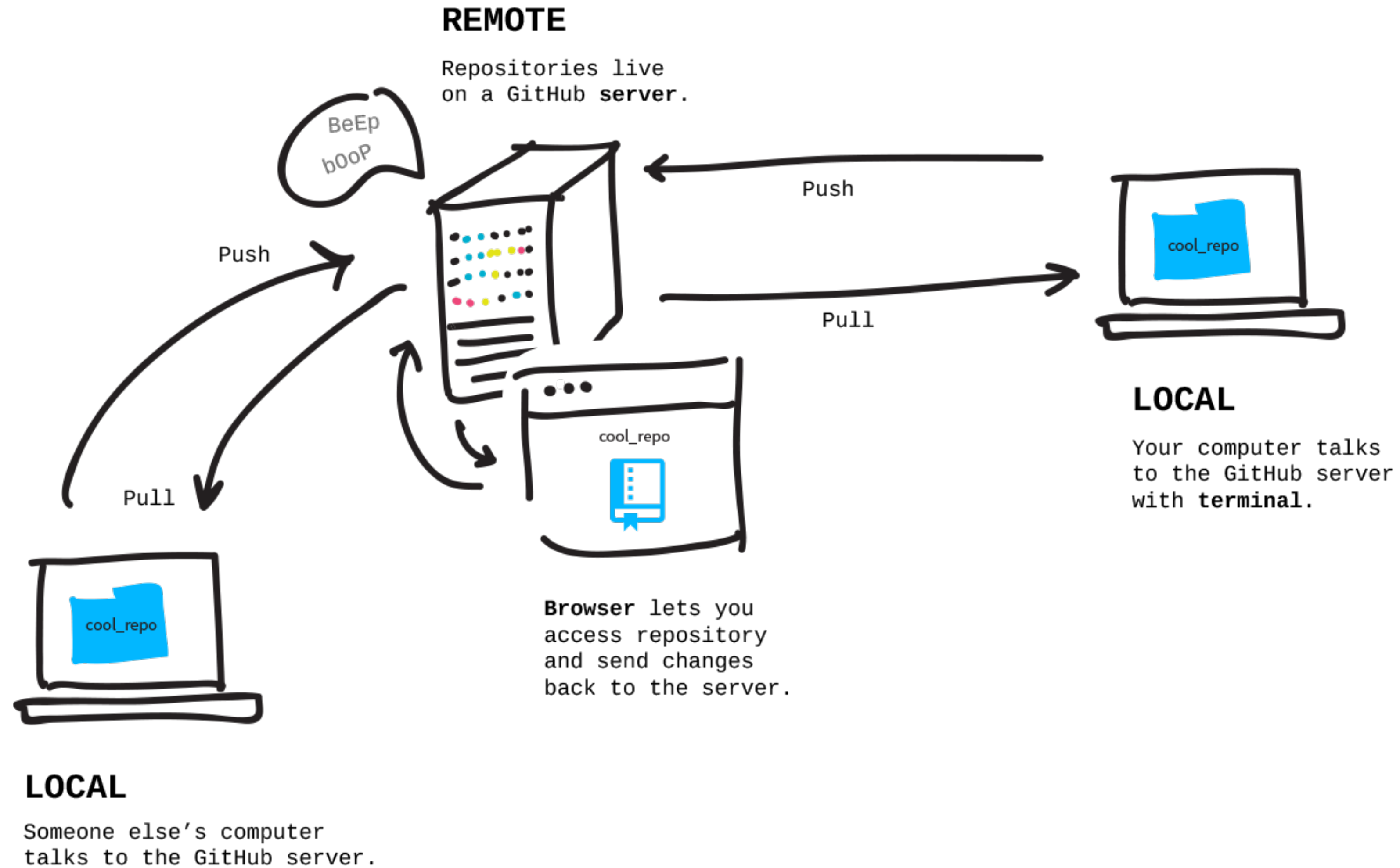
- **Working directory** is the directory on your machine where the repo lives
- **Staging area** is the set of files that has changed since your last *commit*
- The **local repository** is the repo on your machine (including its complete history)
- A **remote repository** is a version of the repo on a remote site such as Github



# Git workflow



# Visualizing local and remote repos



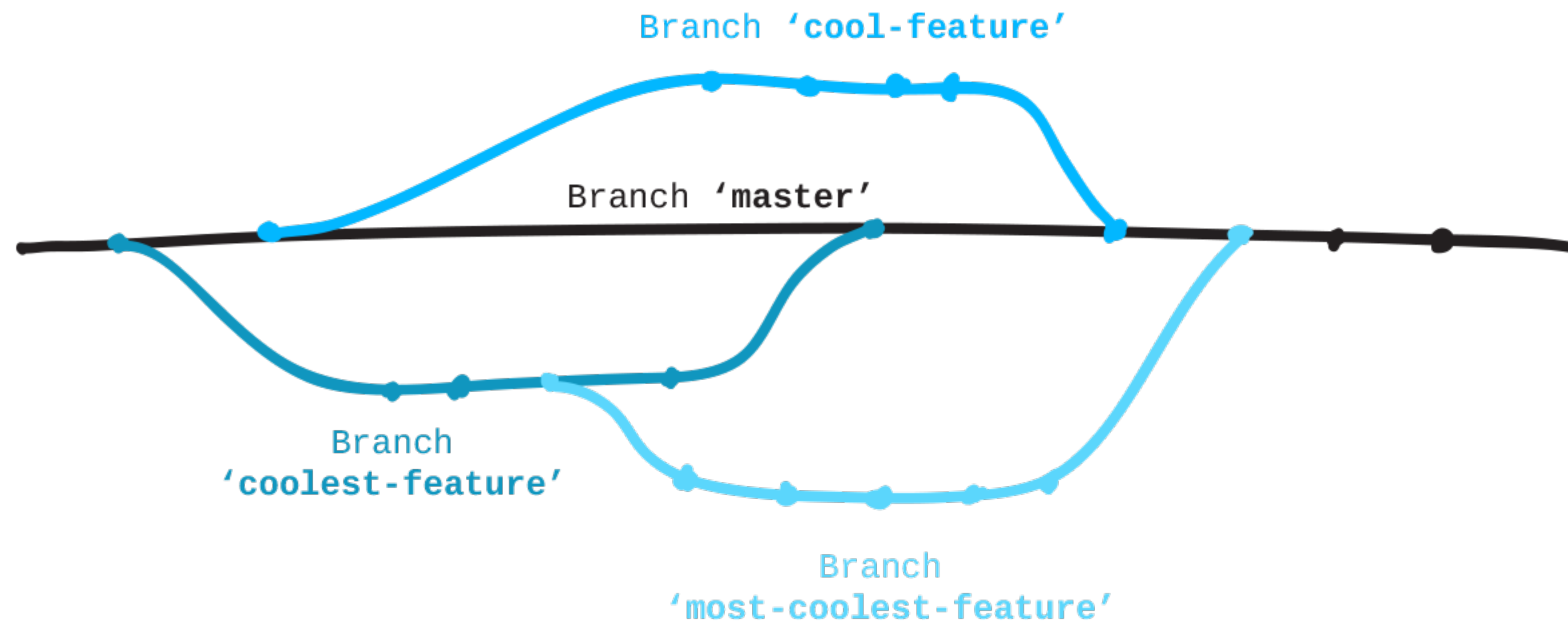
# Basic Git commands

- `git add` will add new or changed files to the staging area
  - ◆ `git add --all` to add all new or changed files
- `git commit` creates a commit out of the staged changes
  - ◆ `git commit -m "notes here"` to commit with a short message
- `git push/pull <remote> <branch>` pushes or pulls commits from your local repo to a remote repo
  - ◆ E.g., `git push origin main`

# A typical workflow

1. Fetch your teammates' changes from Github with `git pull`
2. Make changes to your local repo
3. Stage changes for commit with `git add`
4. Commit staged changes to your local repo with `git commit`
5. Push your changes to Github with `git push`
6. Repeat steps 1-5 and always `pull` before `pushing`!

# Use branches to organize development



# Basic branch commands

- `git branch` lists available branches
- `git checkout -b <name>` will create a new branch
- `git checkout <name>` switches to a different branch
- `git merge <name>` merges a branch into the current one
  - ◆ Commits from the other branch are copied into the current one
  - ◆ You may need to manually fix merge conflicts

BREAK