BEGINNER WORKSHOP:

INTRODUCTION TO GIT & GITHUB WITH R & RSTUDIO

with Nicole A. Swartwood & Christian Testa

Why version control?

- makes coordinating files across computers easy
- keeps a clean history of your code evolution
  - no need for messy suffixes on scripts (V1, V2, ..., V19380)
- gives you the chance to ask yourself, “do I really want to make these updates?”
- streamlines review of external code changes

Why Git?

- 93.9% of developers use Git according to the StackOverflow developer survey.
  - Large resource base
  - Easy compatibility

Git

- Git is an **open-source version control system**.
- Git stores code and its history in a **repository.**
- Each revision to the code is added to the repository through a **commit** process.
- Git allows you to have **branches** of your code that keeps development **separate** from the main codebase until it is complete.
  - The main version of your code is often on the "**main**" branch (what was previously called the "**master**" branch).
- Git allows you to **push** or **pull** code from **remote** servers.

GitHub

- GitHub is a website and online service with free and paid tiers that allows you to:
    - host Git repositories
    - publicize your profile and repositories
    - track issues
    - document your projects with wikis
    - host static websites
    - coordinate teams of developers
    - do project management
    - automate project workflows

Before getting started

Personal Reflection

- ❑ What features of Git and GitHub are you looking to leverage the most?
- ❑ Look through examples of **successful repositories**.

Logistical

- ❑ If you are working with confidential or code requiring **high security**, talk to your local IT to determine your best path forward.
- ❑ Institutions often have their own **Git server or organization**
  - ❑ Example: Harvard has code.harvard.edu

Getting started – installation

- You'll need to install Git: the installation procedures depend on if you're on Windows, Mac, or Linux.
- Follow the installation instructions here https://happygitwithr.com/install-git.html:

Download git for OSX

Download git for Windows

Download git for Linux

Getting started – local setup

- After installation, you'll need to **configure** Git:

```
git config --global user.name 'Jane Doe'
git config --global user.email 'jane@example.com'
git config --global --list
```

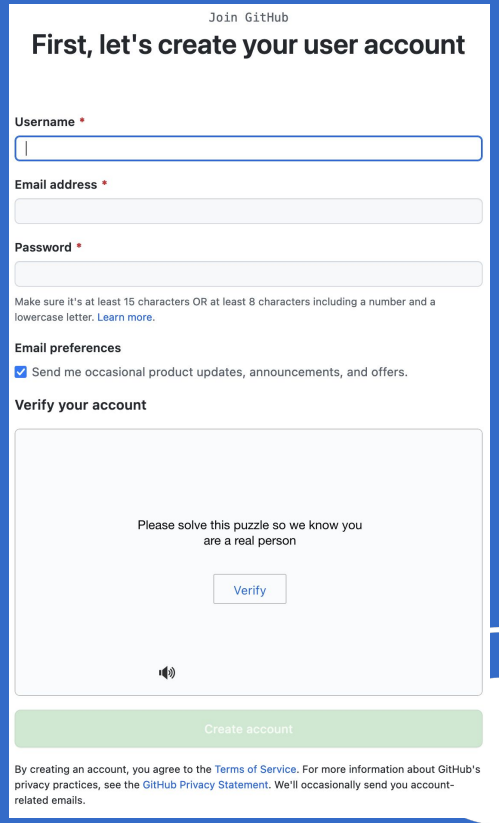
- Make sure to use the **same name** and **email address** you are going to use with **GitHub**.

Getting started – set up a GitHub account

Go to https://github.com/join

Some advice from HappyGitWithR:

- ❑ Incorporate your **actual name**.
- ❑ Reuse your username from other contexts.
- ❑ Pick a username you will be comfortable revealing to your future boss.
- ❑ **Shorter** is better.
- ❑ Be as **unique** as possible in as few characters as possible.
- ❑ Make it **timeless**.
  - ❑ Don't highlight your current university, employer, or place of residence.
- ❑ Recommend all **lowercase**.



Getting started – Set up an SSH key

We recommend setting up **SSH (secure shell) key** based authentication with GitHub.

This allows your computer to be **automatically authenticated** when you communicate with GitHub.

Follow the instructions here:
https://happygitwithr.com/ssh-keys.html

Getting started – set up a local repository – command line

Converting an existing code base

Console | Terminal | Render

Terminal 1 ~/new_directory

```
~ $mkdir new_directory
~ $cd new_directory
new_directory $vim newScript.R
new_directory $git init
Initialized empty Git repository in /Users/nis100/new_directory/.git/
new_directory $ls
newScript.R
new_directory $
new_directory $ls -la
total 8
drwxr-xr-x    4 nis100  staff   128 Aug  9 22:14 .
drwxr-xr-x+ 113 nis100  staff  3616 Aug  9 22:14 ..
drwxr-xr-x    9 nis100  staff   288 Aug  9 22:14 .git
-rw-r--r--    1 nis100  staff    22 Aug  9 22:14 newScript.R
new_directory $
```

Creating a new empty repository

Console | Terminal | Render

Terminal 1 ~/repo-name

```
~ $git init "repo-name"
Initialized empty Git repository in /Users/nis100/repo-name/.git/
~ $cd repo-name/
repo-name $ls
repo-name $ls -la
total 0
drwxr-xr-x    3 nis100  staff    96 Aug  9 22:16 .
drwxr-xr-x+ 114 nis100  staff  3648 Aug  9 22:16 ..
drwxr-xr-x    9 nis100  staff   288 Aug  9 22:16 .git
repo-name $
```

Notice how both of these workflows create a .git file in each of the repositories

Getting started – set up a local repository – RStudio GUI

Creating a new project or package

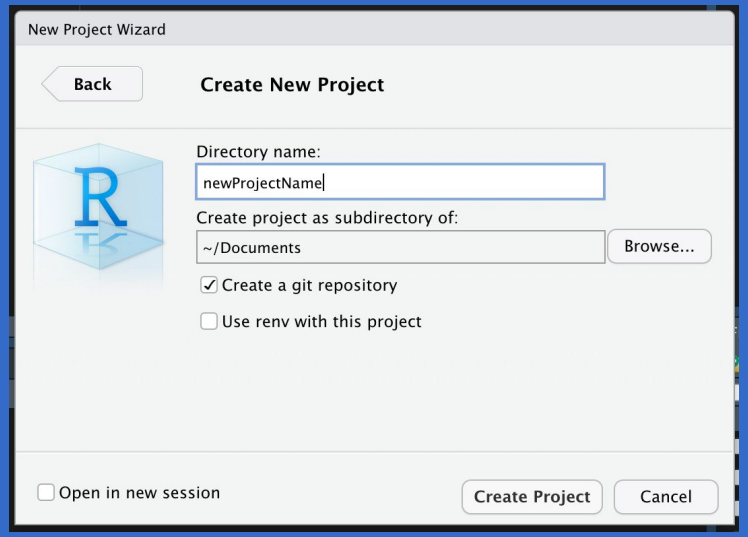
Navigate to

File →

New Project

and select “Create a git repository”

which informs RStudio you want to use Git.



Getting started – set up a local repository – RStudio GUI

Converting an existing project

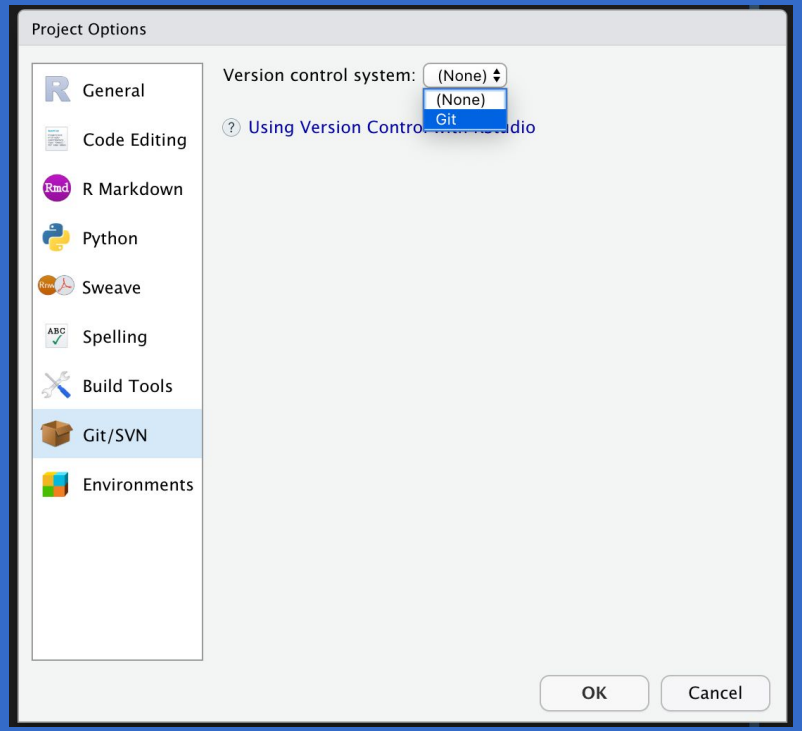
Navigate to

Tools →

Project Options →

Git/SVN

and select “Git” from the “Version control system” dropdown.

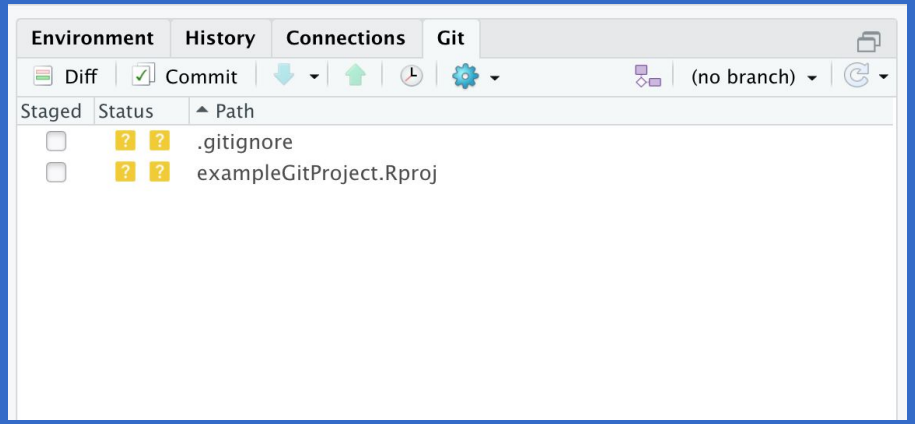


Getting started – git panel in RStudio

RStudio adds a Git tab in your Environment/History panel.

This panel is a point-and-click interface to:

- ❑ review your changes
- ❑ stage changes
- ❑ write commits
- ❑ push and pull commits
- ❑ view the commit history
- ❑ navigate branches



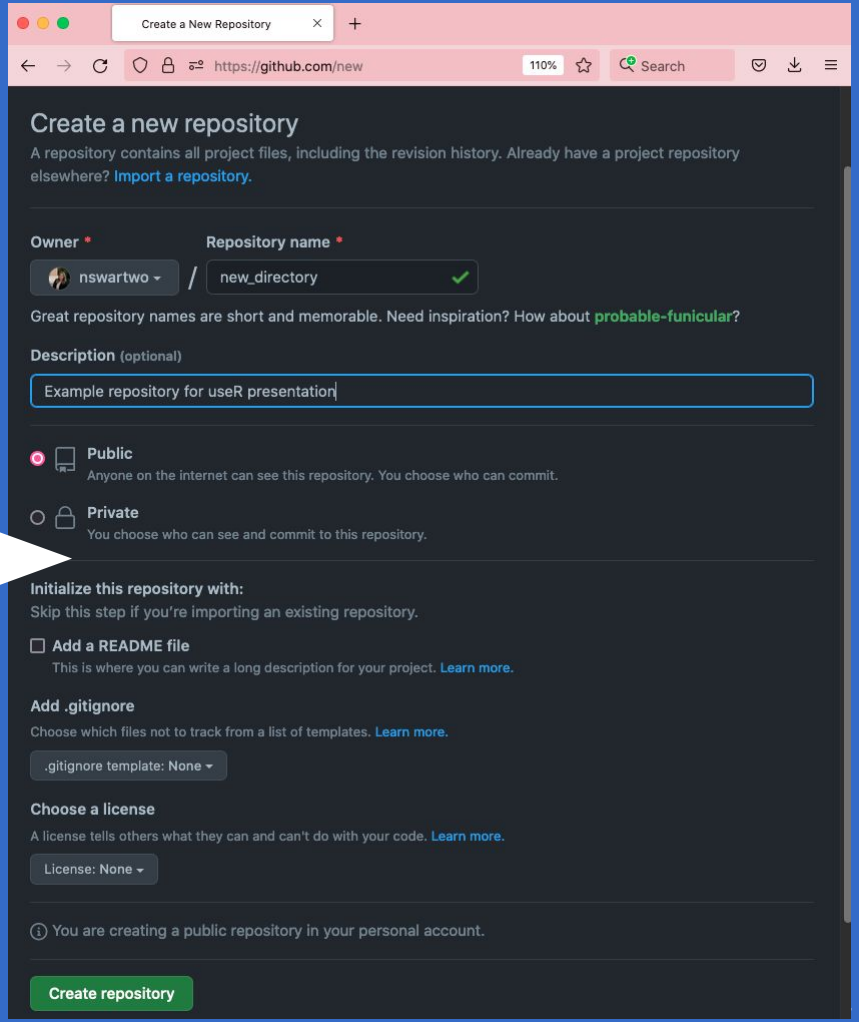
Getting started – setting up a remote repository

Navigate to

github.com →

New →

and fill out the creation form.



Getting started – connecting local and remote

- Need to connect the local and remote repository

```
cd localRepo

git remote add origin repoUrl

git branch -M branchName

git push -u origin branchName
```

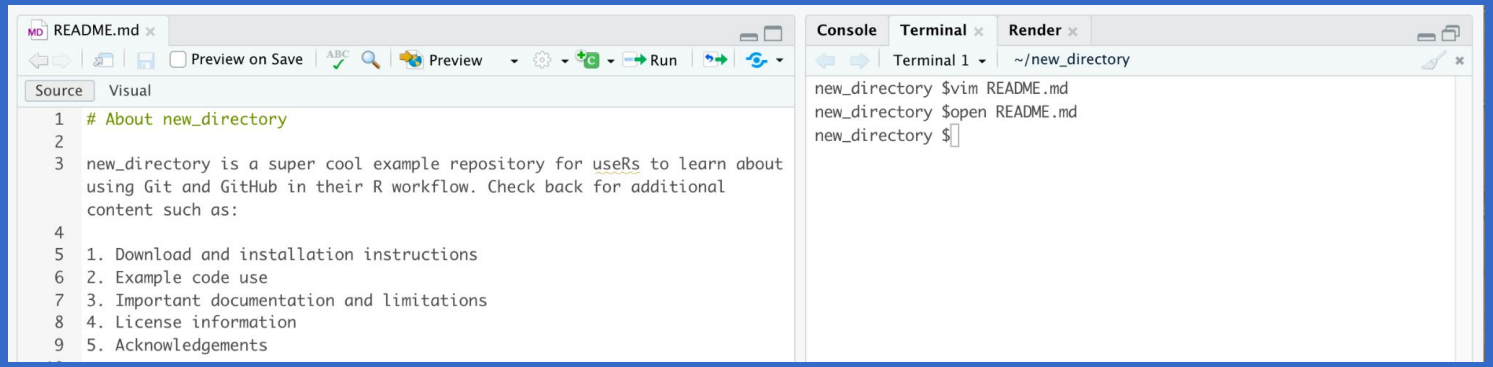
Console | Terminal | Render

Terminal 1 ~/new_directory

```
new_directory $git remote add origin https://github.com/nswartwo/new_directory.git
new_directory $git branch -M main
new_directory $git push -u origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 426 bytes | 426.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/nswartwo/new_directory.git
 * [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
```

README

- A README serves as an introduction to and documentation for your repository.

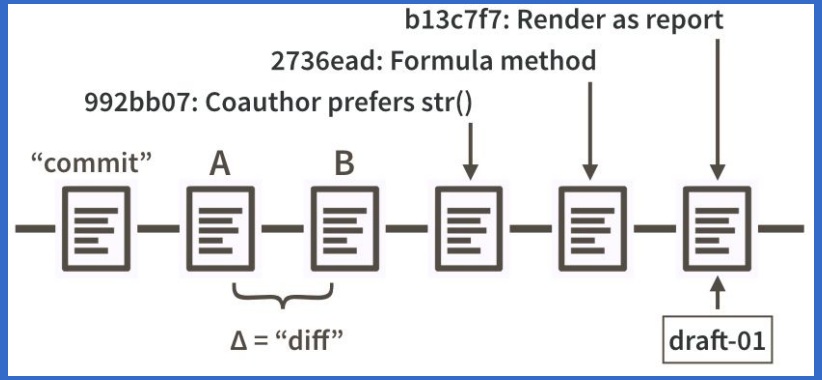


- Like any documentation, feel free to start small and document as you develop!

Basic workflow overview

The basic workflow for making updates to a Git repository is done in three steps:

1. Making changes to your files
2. Adding them to the staging area
3. Commit these changes with a explanatory message



This figure from Happy Git with R shows examples of commits made in a sequence. Each commit is accompanied by an ID, a message, and the differences between two commits are referred to as a "diff".

key commands - git status

- displays the current state of the working directory
- best practice: always run `git status` before code modification
  - will need to run `git fetch` first to see remote changes (slide 25)
- The RStudio Git panel displays most of what we can display with git status.

Console | Terminal | Render

Terminal 1 ~/repo-name

```
repo-name $git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
```

Console | Terminal | Render

Terminal 1 ~/new_directory

```
new_directory $git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        README.md
        newScript.R

nothing added to commit but untracked files present (use "git add" to track)
new_directory $
```

key commands - git add

- adds changes to the staging environment or *index*

command line

```
git add fileName.R
```

Console | Terminal | Render

Terminal 1 ~/new_directory

```
new_directory $git add README.md
new_directory $git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   README.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        newScript.R
```

If you have many changes you can run `git add -A` to stage all of them at once!

RStudio GUI

Environment | History | Connections | Git

Diff | Commit

Staged	Status	Path
☐	? ?	.gitignore
☑	A	README.md
☐	? ?	exampleGitProject.Rproj
☐	? ?	newscript.R

key commands - git commit

- records changes to the repository from the index

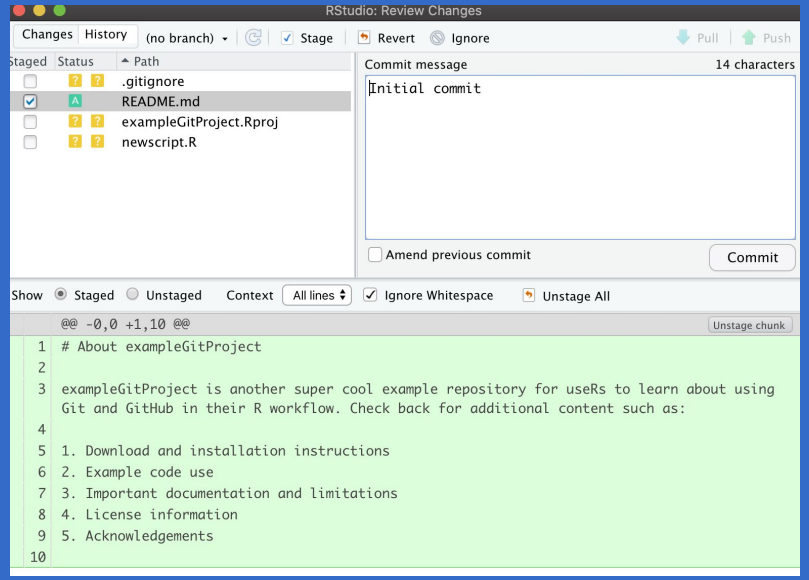
command line

```
git commit -m "commit message"
```

```
Console   Terminal   Render
Terminal 1   ~/new_directory
new_directory $git commit -m "initial commit"
[master (root-commit) 2b3f753] initial commit
 1 file changed, 9 insertions(+)
 create mode 100644 README.md
new_directory $git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        newScript.R

nothing added to commit but untracked files present (use "git add" to track)
new_directory $
```

If you need a longer, multi-level commit message run `git commit` to open a vim editor.

RStudio GUI



Optimizing your commit messages

- **Capitalize the first word** and do not end in punctuation. If using Conventional Commits, remember to use all lowercase.

- Use **imperative mood** in the subject line.

  Example: “Add fix for dark mode toggle state”

- Specify the **type of commit**. It is recommended and beneficial to have a consistent set of words to describe your changes.

  Example: Bugfix, Update, Refactor, Bump, and so on.

- The first line should ideally be **no longer than 50 characters**.

- **Be direct**! Try to eliminate filler words and phrases.

  Examples: though, maybe, I think, kind of.

Optimizing your commit messages

To develop thoughtful commits, consider the following:

- Why have I made these changes?
- What effect have my changes made?
- Why was the change needed?
- What are the changes in reference to?

See more advice here: https://www.freecodecamp.org/news/how-to-write-better-git-commit-messages/

Keep a changelog

- A changelog is a file that contains a **curated**, **informative history** of your project's updates.

- A changelog allows **people** to **easily** see key development and changes in your project.

- Read more about changelogs at keepachangelog.com

key commands - git push

- sends local, committed changes to remote repository

command line

```
git push origin branchName
```

Console | Terminal

Terminal 1 ~/new_directory

```
new_directory $git push origin main
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 312 bytes | 312.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/nswartwo/new_directory.git
   2b3f753..6ce3b07  main -> main
```

RStudio GUI

RStudio

Changes | History | main | Stage | Revert

Your branch is ahead of 'origin/main' by 1 commit.

Staged	Status	Path
☐	? ?	.gitignore
☐	? ?	exampleGitProject.Rproj

Ignore | Pull | Push

Commit message

Git Push | Close

```
>>> /usr/bin/git push origin HEAD:refs/heads/main
To https://github.com/nswartwo/exampleGitProject.git
   5574468..2b1e684  HEAD -> main
```

key commands - git fetch

- downloads objects and history from another repository

command line

```
git fetch branchName
```

Console | Terminal

Terminal 1 ~/new_directory

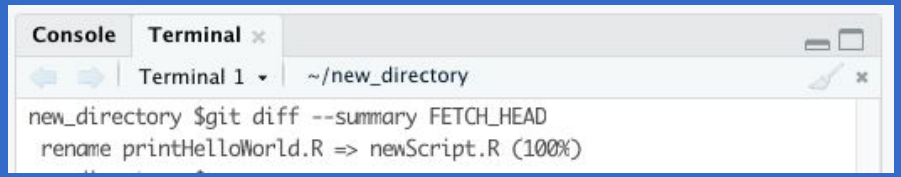
```
new_directory $git fetch
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 2 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (2/2), done.
From https://github.com/nswartwo/new_directory
   6ce3b07..ff0b903  main       -> origin/main
new_directory $git status
On branch main
Your branch is behind 'origin/main' by 1 commit, and can be fast-
forwarded.
  (use "git pull" to update your local branch)

nothing to commit, working tree clean
```

key commands - git diff

- Git diff shows how files differ between their current state and a different version.

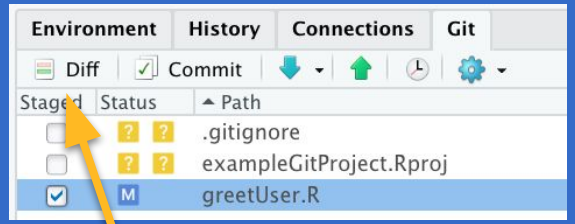
command line

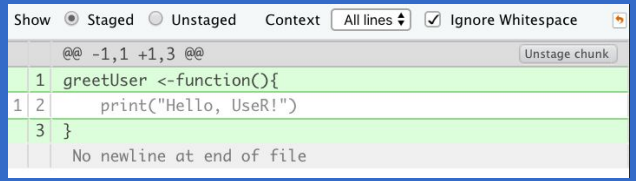
```
git diff --summary FETCH_HEAD
```




```
git diff FETCH_HEAD fileName.R

git diff branch1..branch2

git diff
```

RStudio GUI





key commands - git pull

- Fetch and integrate code from one repository into another

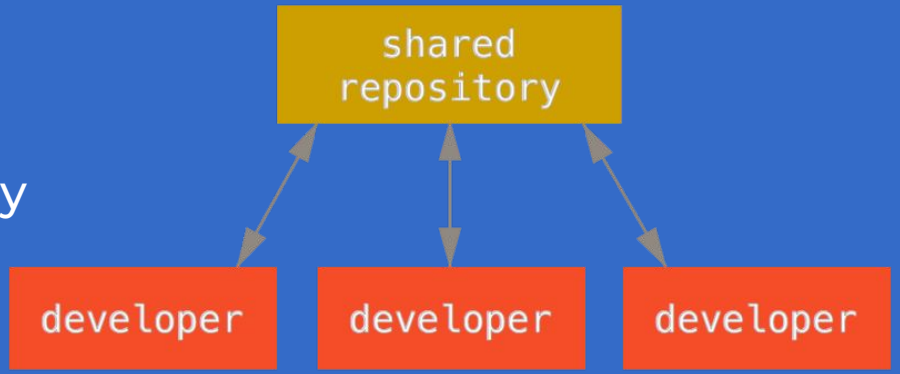
command line

```
git pull
```

RStudio GUI

Workflows for Collaboration

- Small changes – can be reasonable to work on same branch.
- Large changes – create a branch and pull request.
- Outside developer – fork repository and pull request.
- Remember – When working with collaborators, **communication** is key!

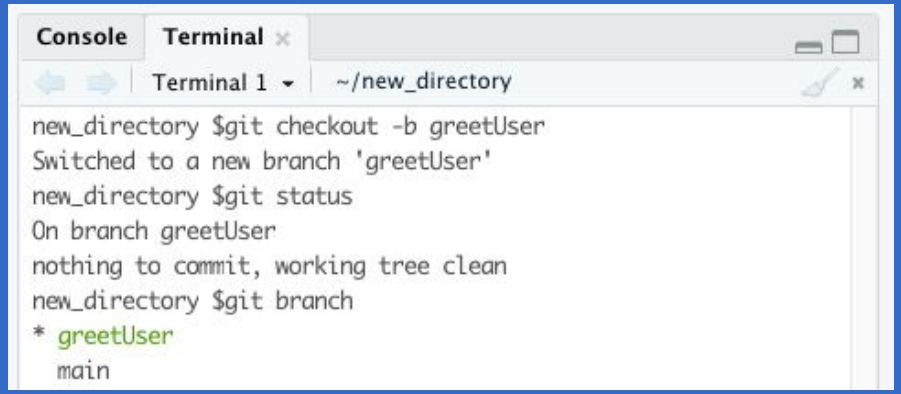


next level commands - git branches

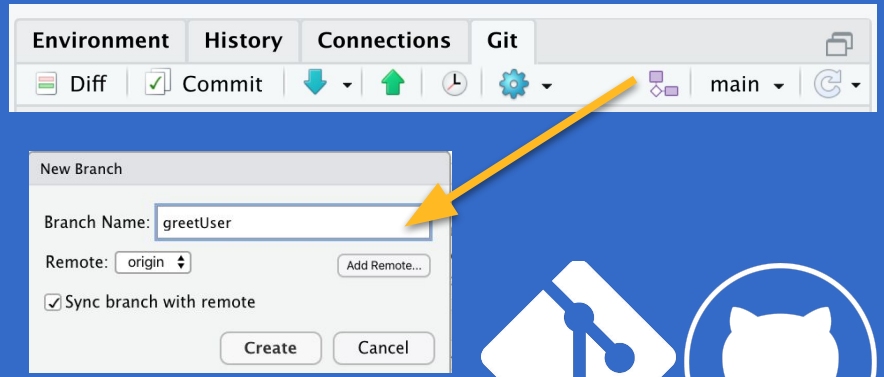
- Branches are essentially a pointer to your changes.
- Creating branches for bug fixes and feature development prevents unstable code from disrupting your project or workflow.

command line

```
git checkout -b branchName
```



RStudio GUI

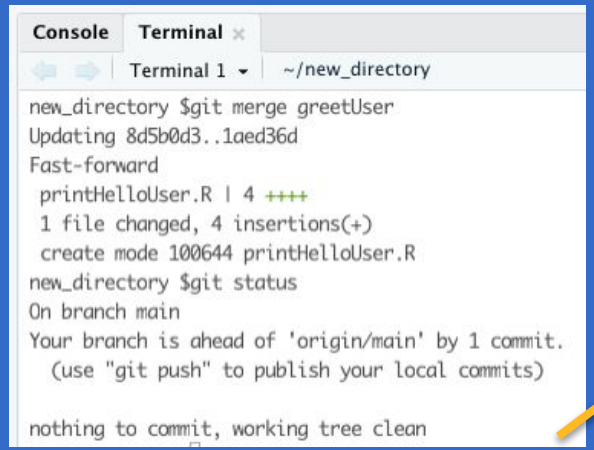


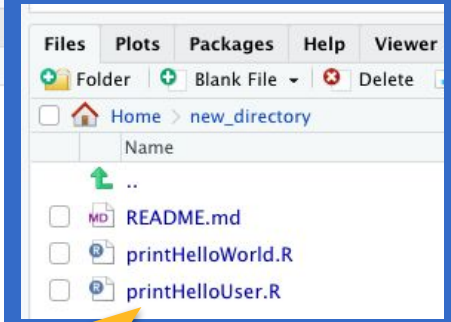
next level commands - git merge

- Merges independent development lines together

command line

```
git checkout branchMergeInto

git merge branchMergeFrom
```

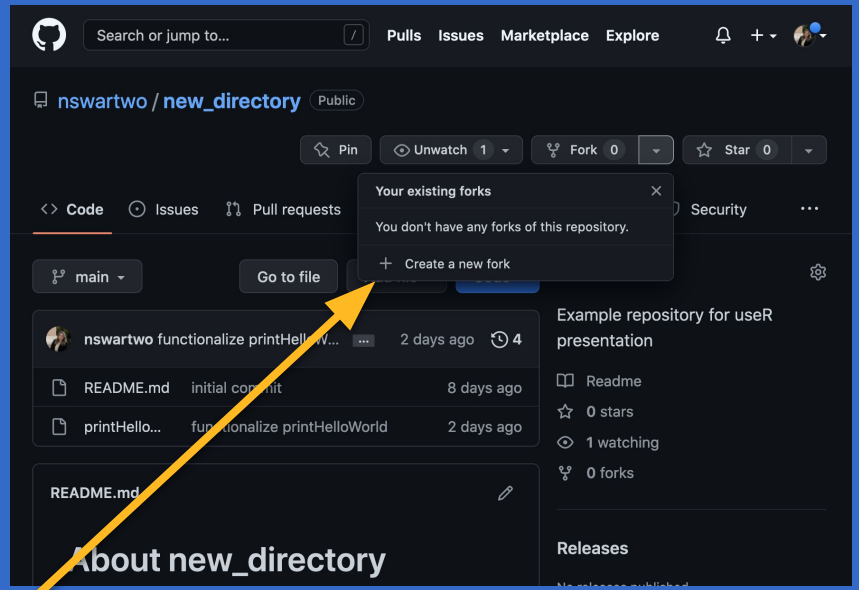




See new file added!

next level commands - git forks

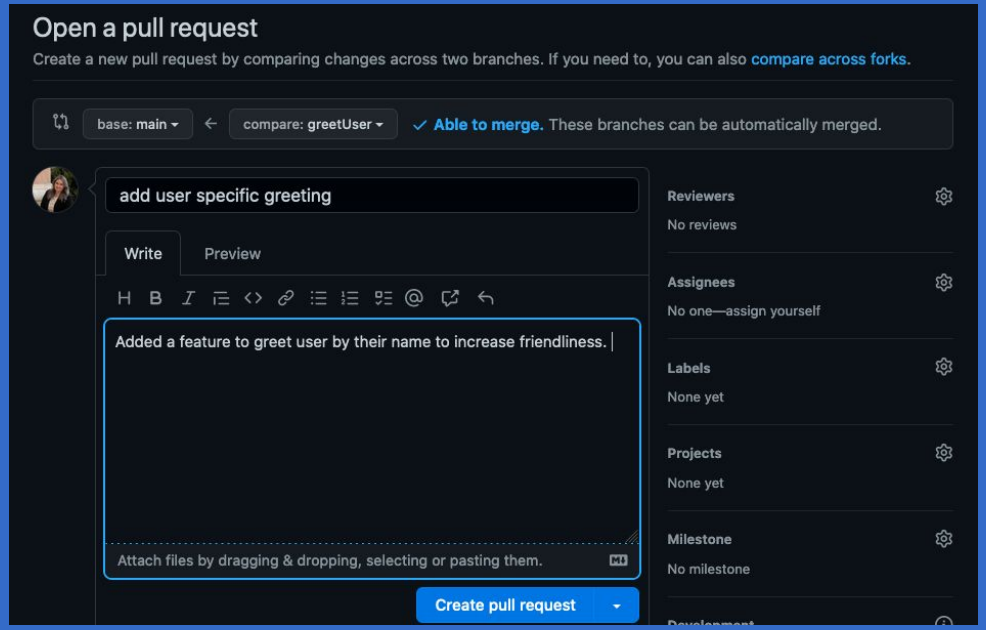
- A copy of a repository which allows development that will not affect the main project.
- Particularly useful when you would like to develop on another person's project.
- Can also be useful for versions of a project that are being developed in parallel without intention of future merge.



creating a pull request

- A pull request alerts the repository owner that changes have been made.
- Gives a chance to review and test the code before adding it into the repository.
- Optimizing language in pull requests is also important!

Further information:
https://github.blog/2015-01-21-how-to-write-the-perfect-pull-request/



next level commands - git stash

- Sometimes when working on one branch, we are not quite ready to commit, but we need to redirect our efforts to another branch.

  Example: you're working on adding a new feature to your project, but then you have to address an immediate customer concern on your production software.

- Instead of needing to commit a half developed feature, we can choose to stash our changes instead.

- Stashing changes allows us to switch branches without losing progress and without compromise to our development history.

next level commands - git stash

command line

```
git stash
git checkout otherBranch
{ do some development; commit}
git checkout branchWithStashes
git stash pop
```

next level commands - git squash

- Squashing commits combines several existing commits into a single commit.
- Can be useful for incorporating development branches into the main branch.
- Use sparingly!

next level commands - options

- Most Git commands have option flags that can be used to further specify the command.

```
--dry_run

--a all

--help
```

- Information on these options can be found on the documentation pages for the Git command.

Some hard-won advice:

- ❑ Try out new git procedures in a dummy repository.

- ❑ If you “break” something (or everything), keep calm, there is almost always a fix.

further resources

- Happy Git and GitHub for the useR – https://happygitwithr.com/
- Git - the simple guide – https://rogerdudler.github.io/git-guide/
- Pro Git – https://git-scm.com/book/en/v2
- Oh Shit, Git!?! – https://ohshitgit.com/

live demonstration