Reproducibility Capstone

June, 2025

Table of Contents

# About this Course

## 0.1 Available course formats

This course is available in multiple formats which allows you to take it in the way that best suites your needs. You can take it for certificate which can be for free or fee.

* The material for this course can be viewed without login requirement on this [Bookdown website](LINK%20HERE). This format might be most appropriate for you if you rely on screen-reader technology.
* This course can be taken for [free certification through Leanpub](LINK%20HERE).
* This course can be taken on [Coursera for certification here](LINK%20HERE) (but it is not available for free on Coursera).
* Our courses are open source, you can find the [source material for this course on GitHub](LINK%20HERE).

# 1 Introduction

## 1.1 Motivation

Cancer research in the 21st century is increasingly data-driven, with researchers generating and analyzing vast amounts of genomic, proteomic, imaging, and clinical data. The ability to effectively manage, analyze, and share these complex datasets has become as crucial as traditional laboratory skills. However, many cancer researchers find themselves working with computational tools and workflows that were developed in isolation, making it difficult to reproduce results, collaborate effectively, or scale their analyses.

The modern cancer informatics landscape demands not just analytical skills, but also proficiency in reproducible computational practices. Research has shown that computational analyses are often difficult or impossible to reproduce without direct contact with the original researchers ([**BeaulieuJones2017?**](#ref-BeaulieuJones2017)). This reproducibility crisis extends beyond academic concerns—it directly impacts the translation of research findings into clinical applications and the overall efficiency of cancer research efforts.

## 1.2 Target Audience

The course is intended for cancer researchers, bioinformaticians, and data scientists who work with computational analyses in cancer research contexts and want to enhance the reproducibility, efficiency, and shareability of their work.

*This capstone course is written for individuals who:*

* This capstone course builds on foundational concepts from the ITCR Training Network’s Reproducibility Series (see list below) Conduct computational analyses for cancer research, including genomics, proteomics, imaging, or clinical data analysis
* Have basic familiarity with command line interfaces and version control (GitHub)
* Write code or scripts for data analysis, whether in R, Python, or other programming languages
* Have experienced challenges with reproducing their own or others’ computational analyses
* Want to learn modern best practices for scientific computing and workflow management
* Need to collaborate with team members or share computational workflows with the scientific community
* Are interested in meeting emerging standards for reproducible research and open science

This capstone course builds on foundational concepts from the ITCR Training Network’s [Reproducibility in Cancer Informatics](https://jhudatascience.org/Reproducibility_in_Cancer_Informatics/introduction.html), [Advanced Reproducibility in Cancer Informatics](https://jhudatascience.org/Adv_Reproducibility_in_Cancer_Informatics/introduction.html), [Containers for Scientists](https://hutchdatascience.org/Containers_for_Scientists/), and [GitHub Automation for Scientists](https://hutchdatascience.org/GitHub_Automation_for_Scientists/) courses. Students should be comfortable with basic GitHub operations, including creating repositories, making commits, and opening pull requests before beginning this capstone.

## 1.3 Curriculum

The course covers practical application of containerization and automation technologies specifically tailored for cancer informatics workflows. Unlike introductory courses that focus on individual concepts, this capstone emphasizes integration and real-world application of multiple technologies working together.

**Assessment Approach:**

The capstone uses a practical, project-based assessment model where students work with a provided computational sandbox environment to complete real-world tasks. Rather than traditional exams, students demonstrate mastery by successfully implementing containerized workflows and automated pipelines that pass automated evaluation criteria.

**Assignment 1: Custom Container Development** - Students modify and extend Docker containers to support specific cancer analysis requirements, demonstrating understanding of containerization principles and bioinformatics software management.

**Assignment 2: Automated Workflow Implementation** - Students create GitHub Actions that automatically execute and validate cancer data analysis pipelines, showing proficiency in continuous integration concepts and workflow automation.

Each assignment utilizes automated evaluation systems that provide immediate feedback and validation codes upon successful completion, simulating real-world development environments where code must pass automated testing before deployment.

**Goal of this course:** Enable cancer researchers to create, maintain, and share reproducible computational workflows that meet the evolving standards of modern cancer informatics while enhancing collaboration and accelerating scientific discovery.

**What is not the goal:** This course does not aim to teach cancer biology, statistical analysis methods, or introductory programming concepts. Students are expected to bring domain expertise and basic computational skills to the course, focusing instead on learning to apply advanced reproducibility technologies to their existing research workflows.

## 1.4 How to use the course

This capstone course is designed as a hands-on, practical experience that builds directly on the foundational knowledge from previous ITCR Training Network courses. Students should approach the course with an active learning mindset, ready to experiment, troubleshoot, and iterate on their solutions.

**Working with the Course Sandbox:** The course utilizes a GitHub-based sandbox environment that simulates real-world research collaboration workflows. Students will work with template repositories, create branches for different assignments, and receive automated feedback through pull request evaluations. This approach mirrors modern software development practices and prepares students for collaborative research environments.

**Iterative Learning Approach:** Unlike traditional courses where concepts are learned in isolation, this capstone emphasizes the iterative refinement of working systems. Students are encouraged to start with simple implementations and gradually add complexity, using automated feedback to guide their development process. This mirrors the reality of computational research, where solutions evolve through experimentation and refinement.

**Integration Focus:** Each assignment builds on previous work and integrates multiple technologies. Students should be prepared to synthesize knowledge from containerization, automation, version control, and domain-specific cancer informatics concepts. The goal is to create holistic solutions rather than demonstrate isolated technical skills.

**Real-world Application:** Throughout the course, students are encouraged to consider how the techniques they’re learning apply to their own research contexts. The assignments use cancer informatics examples, but the principles and technologies transfer to a wide range of computational research domains.

We also recommend that students leverage the broader ITCR Training Network community and resources, including the [GitHub repository discussions](https://www.itcrtraining.org/), course forums, and peer collaboration opportunities to enhance their learning experience and build lasting professional connections in the cancer informatics community.

# 2 Setting up for this course

## 2.1 Phase 1: Creating Your Repository from Template

### 2.1.1 Step 1: Access the Course Template

* Navigate to the [course sandbox template repository](https://github.com/fhdsl/capstone-sandbox)
* Look for the green “Use this template” button at the top of the repository page
* Important: Do NOT fork the repository - you must use the template option

### 2.1.2 Step 2: Configure Your New Repository

* Click “Use this template” → “Create a new repository”
* Fill out the repository creation form:
  + Repository name: Choose a meaningful name (e.g., your-username-capstone-sandbox)
  + Description: Add a brief description of your capstone project
  + Visibility: Set to Public (required for the automated evaluation system to work)
  + Include all branches: Leave unchecked (we only need the main branch)

### 2.1.3 Step 3: Create Your Repository

* Click “Create repository”
* Wait for GitHub to finish creating your repository
* You’ll be redirected to your new repository page
* Verify: The repository should show your username as the owner and contain all the template files

## 2.2 Phase 2: Local Setup and Cloning

### 2.2.1 Step 4: Prepare Your Local Environment

* Ensure you have Git installed on your computer
* Check Git installation: Open terminal/command prompt and run git --version
* If Git isn’t installed, download it from [git-scm.com](https://git-scm.com/)
* Optional but recommended: Install [GitHub Desktop](https://desktop.github.com/) for a visual interface

### 2.2.2 Step 5: Clone Your Repository

#### 2.2.2.1 Option A: Using Command Line

1. Copy the repository URL:
   * On your repository page, click the green “Code” button
   * Copy the HTTPS URL (should look like: https://github.com/yourusername/your-repo-name.git)
2. Clone the repository:

* # Navigate to where you want to store the project  
  cd ~/Desktop # or your preferred location  
    
  # Clone the repository  
  git clone https://github.com/yourusername/your-repo-name.git  
    
  # Navigate into the repository  
  cd your-repo-name

#### 2.2.2.2 Option B: Using GitHub Desktop

1. Open GitHub Desktop
2. Click “Clone a repository from the Internet”
3. Select your repository from the list or enter the URL
4. Choose a local path where you want to store the repository
5. Click “Clone”

### 2.2.3 Step 6: Verify Your Local Setup

* Check repository contents:
* ls -la  
  # You should see the template files and folders
* Verify Git connection:
* git status  
  # Should show "On branch main" and "working tree clean"

## 2.3 Phase 3: Branch Creation and Workflow

### 2.3.1 Step 7: Create Your First Assignment Branch

#### 2.3.1.1 Command Line Method:

# Create and switch to a new branch for Assignment 1  
git checkout -b assignment-1-dockerfile  
  
# Verify you're on the new branch  
git branch  
# The asterisk (\*) should be next to "assignment-1-dockerfile"

#### 2.3.1.2 GitHub Desktop Method:

1. Click on the “Current branch” dropdown (shows “main”)
2. Click “New branch”
3. Name your branch: assignment-1-dockerfile
4. Click “Create branch”

### 2.3.2 Step 8: Understand Branch Naming Strategy

For each assignment, create descriptive branch names. Here are examples: - Assignment 1: assignment-1-dockerfile or dockerfile-build - Assignment 2: assignment-2-github-action or gha-creation - Future work: feature-data-analysis, fix-container-issue, etc.

## 2.4 Phase 4: Making Changes and Committing

### 2.4.1 Step 9: Make Your Assignment Changes

* Work on your assignment files according to the specific assignment instructions
* For Assignment 1: Edit the docker/Dockerfile
* For Assignment 2: Edit and move the ASSIGNMENT\_2.yml file
* Save your changes in your preferred editor

### 2.4.2 Step 10: Stage and Commit Your Changes

#### 2.4.2.1 Command Line Method:

# Check what files have changed  
git status  
  
# Add specific files (recommended)  
git add docker/Dockerfile # for Assignment 1 for example  
   
# OR add all changes (use carefully)  
git add .  
  
# Commit with a descriptive message  
git commit -m "Add package installations to Dockerfile for Assignment 1"  
  
# Alternative: Add and commit in one step  
git commit -am "Add package installations to Dockerfile for Assignment 1"

#### 2.4.2.2 GitHub Desktop Method:

1. Review changed files in the left sidebar
2. Check the boxes next to files you want to include
3. Write a descriptive commit message in the bottom left
4. Click “Commit to assignment-1-dockerfile”

### 2.4.3 Step 11: Push Your Branch to GitHub

#### 2.4.3.1 Command Line Method:

# Push your new branch to GitHub (first time)  
git push --set-upstream origin assignment-1-dockerfile  
  
# For subsequent pushes to the same branch  
git push

#### 2.4.3.2 GitHub Desktop Method:

1. Click “Publish branch” (for first push)
2. For subsequent changes, click “Push origin”

## 2.5 Phase 5: Opening Pull Requests

### 2.5.1 Step 12: Create Your Pull Request

1. Navigate to your repository on GitHub.com
2. You should see a yellow banner saying “assignment-1-dockerfile had recent pushes” with a “Compare & pull request” button
3. Click “Compare & pull request”

#### 2.5.1.1 Alternative method if no banner appears:

1. Click the “Pull requests” tab
2. Click “New pull request”
3. Set the base branch to main and compare branch to your assignment branch
4. Click “Create pull request”

### 2.5.2 Step 13: Configure Your Pull Request

* Title: Use a descriptive title (e.g., “Assignment 1: Docker container with required packages”)
* Description: Add details about what you implemented:
* ## Assignment 1 Submission  
    
  This PR adds the following to the Dockerfile:  
  - [List the packages you added]  
  - [Any other changes you made]  
    
  Ready for automated evaluation.
* Labels: Add any relevant labels if available
* Assignees: Assign yourself

### 2.5.3 Step 14: Submit and Monitor Your Pull Request

1. Click “Create pull request”
2. Wait for automated checks to run - you should see status indicators at the bottom of the PR
3. Monitor for the evaluation comment - the automated system will comment with results
4. Keep the PR open until you receive your validation code

## 2.6 Phase 6: Iteration and Resubmission

### 2.6.1 Step 15: Handle Evaluation Feedback

If your submission needs changes:

1. Stay on your assignment branch:

* git checkout assignment-1-dockerfile # if not already there

1. Make necessary changes based on the automated feedback
2. Commit and push updates:

* git add .  
  git commit -m "Fix Dockerfile syntax based on evaluation feedback"  
  git push

1. The pull request will automatically update and trigger re-evaluation

### 2.6.2 Step 16: Success and Next Steps

When you receive your validation code: 1. Copy the validation code from the automated comment 2. Keep your PR open (don’t merge or close it yet) 3. Submit the code in your Coursera quiz 4. Start the next assignment by creating a new branch from main

## 2.7 Reminders of Best Practices for Success

### 2.7.1 Repository Organization

* One branch per assignment - keeps work organized
* Descriptive commit messages - helps track your progress
* Regular pushes - protects your work and enables evaluation

### 2.7.2 Pull Request Hygiene

* Clear titles and descriptions - will help you and your collaborators know what the work in the branch is.
* Don’t merge your PR before you get the validation code
* Monitor automated feedback - respond to evaluation comments promptly

### 2.7.3 Troubleshooting Common Issues

#### 2.7.3.1 “Repository not found” when cloning:

* Check that the repository is public
* Verify the URL is correct
* Ensure you have access to the repository

#### 2.7.3.2 “Permission denied” when pushing:

* Check that you’re pushing to your own repository (not the template)
* Verify your Git credentials are set up correctly
* Try using a personal access token instead of password

#### 2.7.3.3 Automated evaluation not running:

* Ensure your PR is against the main branch
* Check that required files are in the correct locations
* Verify the repository is public

#### 2.7.3.4 Branch confusion:

# Check current branch  
git branch  
  
# Switch to main branch  
git checkout main  
  
# Create new branch from main  
git checkout -b new-assignment-branch

This setup process creates the foundation for all your capstone assignments. Each assignment will follow a similar pattern: create branch → make changes → commit → push → open PR → receive evaluation → iterate if needed. The automated evaluation system depends on this workflow, so following these steps precisely is crucial for getting your validation codes!

# 3 Assignment 1: Building Your Capstone Dockerfile

## 3.1 Phase 1: Repository Setup and Planning

### 3.1.1 Step 1: Set Up Your Working Environment

* Clone or navigate to your capstone sandbox repository
* Locate the docker/Dockerfile - this is where you’ll be working
* Important: Don’t change the name or location of this file
* Open the existing Dockerfile to see what’s already there

### 3.1.2 Step 2: Create Your Working Branch

* Create a new branch for this assignment (e.g., dockerfile-assignment or docker-build)
* Switch to this branch before making any changes
* This keeps your work organized and allows the automated testing to work properly

### 3.1.3 Step 3: Analyze Your Capstone Project Needs

* Look at your capstone project files in the repository
* Identify what programming languages you’re using (R, Python, etc.)
* Make a list of packages/libraries your analysis scripts require
* Check if there are any special tools or dependencies needed

## 3.2 Phase 2: Dockerfile Development

### 3.2.1 Step 4: Examine the Base Setup

* Open docker/Dockerfile and see what base image is already specified
* Read any existing instructions or comments
* Understand what’s already included before adding new components

### 3.2.2 Step 5: Add Packages Incrementally

* Start by adding just one or two essential packages to your Dockerfile
* Use the templates from the course:
  + For R packages: RUN Rscript -e "install.packages('packagename')"
  + For Python packages: RUN pip3 install packagename
* Add comments explaining what each package does

### 3.2.3 Step 6: Build Locally (If Working on Your Computer)

* If developing locally, test your Dockerfile with:
* cd docker  
  docker build . -t my-capstone-image
* Fix any build errors before pushing to GitHub
* This saves time and GitHub Actions usage

## 3.3 Phase 3: Testing and Iteration

### 3.3.1 Step 7: Open Your Pull Request

* Commit your Dockerfile changes to your branch
* Push the branch to GitHub
* Open a pull request from your branch to the main branch
* Key: This triggers the Docker Assignment Eval test

### 3.3.2 Step 8: Monitor the Automated Testing

* Watch for the Docker Assignment Eval check to start running
* This GitHub Action will attempt to build your Dockerfile
* Wait for it to complete and comment on your PR

### 3.3.3 Step 9: Interpret the Results

* Success: You’ll get a validation code in the comment
* Failure: You’ll get error messages explaining what went wrong
* Read the error messages carefully - they contain clues for fixing issues

## 3.4 Phase 4: Troubleshooting and Refinement

### 3.4.1 Step 10: Debug Build Failures

* Common issues to check:
  + Typos in package names
  + Missing dependencies (install system packages before language packages)
  + Incorrect syntax in RUN commands
  + Base image doesn’t support your installation method

### 3.4.2 Step 11: Apply Troubleshooting Strategies

* Package not found: Check spelling, verify the package exists
* Installation fails: Look for missing system dependencies
* Syntax errors: Review course templates and examples
* Memory issues: Simplify your build, install fewer packages at once

### 3.4.3 Step 12: Iterate Until Success

* Make changes to your Dockerfile on your branch
* Commit and push changes
* Each push will trigger a new test run
* Keep refining until you get a successful build

## 3.5 Phase 5: Completion

### 3.5.1 Step 13: Collect Your Validation Code

* Once your build succeeds, copy the validation code from the PR comment
* Important: Make sure it’s a real validation code, not an error message
* Keep this code safe - you’ll need it for your quiz

### 3.5.2 Step 14: Submit Your Quiz

* Go to your Coursera Capstone Quiz
* Paste the validation code when prompted
* Submit the quiz to get credit for your work

## 3.6 Pro Tips for Success

### 3.6.1 Start Simple

* Begin with just the most essential packages for your analysis
* You can always add more in subsequent iterations
* A working simple image is better than a broken complex one

### 3.6.2 Use the Course Resources

* Reference the package installation templates from the Containers for Scientists course
* Look for examples of other people installing similar packages
* Don’t reinvent the wheel

### 3.6.3 Leverage the Automated Testing

* The GitHub Action is your friend - it provides immediate feedback
* Don’t be afraid to make multiple attempts
* Each failure teaches you something about container building

### 3.6.4 Read Error Messages Carefully

* Error messages often contain the exact solution
* Look for phrases like “package not found” or “missing dependency”
* Google specific error messages if they’re unclear

### 3.6.5 Document Your Process

* Add comments to your Dockerfile explaining your choices
* This helps you remember your reasoning if you need to modify it later
* Good documentation makes debugging easier

Remember: This assignment is designed to give you hands-on experience with the container building process. The automated testing system provides immediate feedback, making it easier to learn through iteration rather than getting everything perfect on the first try.

# 4 Assignment 2: Building Your Capstone GitHub Action

## 4.1 Phase 1: Repository Setup and Planning

### 4.1.1 Step 1: Set Up Your Working Environment

* Navigate to your capstone sandbox repository
* Locate the ASSIGNMENT\_2.yml file - this is your starting template
* Important: Keep the filename exactly as ASSIGNMENT\_2.yml throughout the assignment
* Review the existing template to understand what’s already provided

### 4.1.2 Step 2: Create Your Working Branch

* Create a new branch for this assignment (e.g., github-action-assignment or gha-build)
* Switch to this branch before making any changes
* This isolates your work and enables the automated evaluation system

### 4.1.3 Step 3: Plan Your GitHub Action

* Decide what useful task your GitHub Action will perform
* Ideas for useful actions:
  + Run automated tests on your capstone code
  + Generate reports or documentation
  + Check code quality or formatting
  + Create data visualizations
  + Send notifications when certain conditions are met
  + Validate data files or configurations

## 4.2 Phase 2: GitHub Action Development

### 4.2.1 Step 4: Understand the Template Structure

* Open ASSIGNMENT\_2.yml and examine the existing structure
* Identify the key components:
  + name: - What your action is called
  + on: - When it should trigger
  + jobs: - What it should do
  + runs-on: - What environment to use

### 4.2.2 Step 5: Move File to Correct Location

* GitHub Actions must be in the .github/workflows/ directory
* Move ASSIGNMENT\_2.yml to .github/workflows/ASSIGNMENT\_2.yml
* Critical: The file must be in this exact location for GitHub to recognize it

### 4.2.3 Step 6: Define Your Action’s Purpose

* Choose a meaningful trigger (on:):
  + pull\_request: - Runs when PRs are opened/updated (good for testing)
  + push: - Runs when code is pushed to specific branches
  + workflow\_dispatch: - Allows manual triggering (useful for development)
  + schedule: - Runs on a time schedule
* Design your job steps:
  + Start with actions/checkout@v4 to get your repository files
  + Add steps that accomplish your chosen task
  + Include error handling and status checks

## 4.3 Phase 3: Implementation Strategies

### 4.3.1 Step 7: Start Simple and Build Up

* Begin with a basic action that you know will work
* Test early and often to catch issues quickly
* Add complexity gradually, testing each addition

### 4.3.2 Step 8: Use Course Knowledge

* Apply containers knowledge:
  + Choose appropriate runs-on: environment
  + Consider using Docker containers if you need specific software
  + Reference container images from Docker Hub if needed
* Apply automation principles:
  + Ensure your action fails appropriately when something goes wrong
  + Include meaningful output and logging
  + Use environment variables and secrets when needed

### 4.3.3 Step 9: Common Implementation Patterns

#### 4.3.3.1 For Code Quality/Testing:

name: Code Quality Check  
on: pull\_request  
jobs:  
 quality-check:  
 runs-on: ubuntu-latest  
 steps:  
 - uses: actions/checkout@v4  
 - name: Run quality checks  
 run: |  
 # Your quality check commands here  
 echo "Running code quality checks..."

#### 4.3.3.2 For Data Analysis:

name: Automated Analysis  
on: workflow\_dispatch  
jobs:  
 analyze:  
 runs-on: ubuntu-latest  
 container:  
 image: jhudsl/ottr\_python:main # Example with R and Python  
 steps:  
 - uses: actions/checkout@v4  
 - name: Run analysis  
 run: |  
 # Your analysis commands here

#### 4.3.3.3 For Documentation:

name: Generate Documentation  
on: push  
jobs:  
 docs:  
 runs-on: ubuntu-latest  
 steps:  
 - uses: actions/checkout@v4  
 - name: Generate docs  
 run: |  
 # Documentation generation commands

## 4.4 Phase 4: Testing and Iteration

### 4.4.1 Step 10: Open Your Pull Request

* Commit your changes to your branch
* Push the branch to GitHub
* Open a pull request from your branch to main
* Key: This triggers the GHA Assignment Eval test

### 4.4.2 Step 11: Monitor Automated Evaluation

* Watch for the GHA Assignment Eval check to start running
* This evaluator will test whether your GitHub Action runs successfully
* Wait for it to complete and comment on your PR

### 4.4.3 Step 12: Interpret Evaluation Results

* Success: You’ll receive a validation code in the PR comment
* Failure: You’ll get specific error messages and troubleshooting tips
* The evaluator checks for common issues and provides guidance

## 4.5 Phase 5: Troubleshooting and Refinement

### 4.5.1 Step 13: Debug Common Issues

* YAML syntax errors: Check indentation, colons, and spacing
* File location errors: Ensure file is in .github/workflows/
* Permission errors: May need to add GitHub secrets or tokens
* Missing dependencies: Check if your chosen environment has required software
* Silent failures: Verify your action actually does what you expect

### 4.5.2 Step 14: Apply Troubleshooting Strategies

* Read error messages carefully: They often contain the exact solution
* Check the logs: Go to Actions tab and examine detailed output
* Test incrementally: Make small changes and test each one
* Use marketplace actions: Leverage existing actions when possible
* Print debugging info: Use echo commands to verify assumptions

### 4.5.3 Step 15: Iterate Until Success

* Make changes to your ASSIGNMENT\_2.yml file on your branch
* Commit and push changes to trigger new evaluation runs
* Keep refining based on feedback until you get a successful build

## 4.6 Phase 6: Completion

### 4.6.1 Step 16: Collect Your Validation Code

* Once your GitHub Action runs successfully, copy the validation code from the PR comment
* Verify: Make sure it’s a real validation code, not an error message
* Keep this code safe for your quiz submission

### 4.6.2 Step 17: Submit Your Quiz

* Navigate to your Coursera Capstone Quiz
* Paste the validation code when prompted
* Submit the quiz to receive credit for your work

## 4.7 Pro Tips for Success

### 4.7.1 Choose Appropriate Scope

* Your action should be useful but not overly complex
* Focus on demonstrating GitHub Actions concepts rather than building production software
* A working simple action is better than a broken complex one

### 4.7.2 Leverage Course Materials

* Use troubleshooting strategies from the GitHub Actions course
* Reference YAML examples from course exercises
* Apply container knowledge if your action needs specific software environments

### 4.7.3 Use the Evaluation System Effectively

* The automated evaluator provides immediate feedback
* Don’t be afraid to make multiple attempts - each iteration teaches you something
* Read the evaluator’s comments carefully for specific guidance

### 4.7.4 Think Like a Developer

* Consider when your action should run (triggers)
* Think about error handling and edge cases
* Include meaningful logging and output messages
* Test your assumptions about what the action environment provides

### 4.7.5 Common Useful Actions for Capstone Projects

* Data validation: Check that uploaded data meets expected formats
* Report generation: Automatically create summary reports from data
* Code style checking: Ensure consistent formatting and style
* Dependency checking: Verify all required packages are available
* Backup creation: Automatically backup important files
* Notification system: Send alerts when certain conditions are met

Remember: This assignment builds on both the containers knowledge (for choosing appropriate environments) and the GitHub Actions automation concepts. The goal is to demonstrate your understanding of continuous integration principles by creating something that automatically improves or validates your capstone project.

The automated evaluation system is designed to help you learn through iteration, so embrace the feedback loop and keep refining your action until it works reliably!

# 5 Storing GitHub Secrets: A Step-by-Step Guide for Assignment 3

## 5.1 Phase 1: Understanding GitHub Secrets

### 5.1.1 Step 1: What Are GitHub Secrets?

* GitHub Secrets are encrypted environment variables stored securely in your repository
* They allow you to store sensitive information (like API keys, tokens, passwords) without exposing them in your code
* Secrets are only accessible to GitHub Actions during workflow runs and appear as \* in logs
* This assignment teaches you a fundamental skill for secure automation in research environments

### 5.1.2 Step 2: Why Personal Access Tokens Matter

* Personal Access Tokens (PATs) provide secure authentication for GitHub Actions
* They allow automated workflows to perform actions that require permissions (like creating issues, pushing to repositories, etc.)
* In research contexts, PATs enable automated data sharing, publication workflows, and collaborative tools
* Important: PATs should be treated like passwords - never share them publicly

## 5.2 Phase 2: Creating Your Personal Access Token

### 5.2.1 Step 3: Navigate to GitHub Token Settings

1. Go to your GitHub profile (click your profile picture in the top right)
2. Click “Settings” from the dropdown menu
3. Scroll down to “Developer settings” in the left sidebar (at the bottom)
4. Click “Personal access tokens”
5. Select “Tokens (classic)” from the submenu

Navigation path: Profile → Settings → Developer settings → Personal access tokens → Tokens (classic)

### 5.2.2 Step 4: Generate New Token

1. Click “Generate new token”
2. Select “Generate new token (classic)”
3. GitHub may ask for your password - enter it to continue
4. You’ll be taken to the token creation page

### 5.2.3 Step 5: Configure Your Token

1. Add a descriptive note:
   * Use something like “Capstone Assignment 3 - Secret Storage”
   * This helps you remember what this token is for later
2. Set expiration:
   * For this assignment, you can set it to 30 days or custom
   * In real research, consider the security vs. convenience trade-off
3. Select scopes (permissions):
   * Check “repo” - this gives full repository access
   * For this assignment, “repo” scope is sufficient
   * In production, use the minimum required permissions

Token configuration showing Note field, Expiration dropdown, and Scopes checklist with "repo" highlighted

### 5.2.4 Step 6: Generate and Copy Token

1. Scroll to bottom and click “Generate token”
2. CRITICAL: Copy the token immediately - GitHub will only show it once (perhaps keep the window open as you copy it)
3. The token will look like: ghp\_xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

⚠️ Warning: If you navigate away without copying the token, you’ll need to create a new one!

## 5.3 Phase 3: Storing the Secret in Your Repository

### 5.3.1 Step 7: Navigate to Repository Settings

1. Go to your capstone sandbox repository (not your personal settings)
2. Click the “Settings” tab at the top of the repository page
3. In the left sidebar, find “Secrets and variables”
4. Click “Actions” under “Secrets and variables”

Repository navigation: Repository → Settings → Secrets and variables → Actions

### 5.3.2 Step 8: Create New Repository Secret

1. Click “New repository secret” (green button)
2. Enter the secret details:
   * Name: ASSIGNMENT\_SECRET
   * ⚠️ CRITICAL: Must be exactly this name - case sensitive!
   * Secret: Paste your personal access token from Step 6
3. Click “Add secret”

### 5.3.3 Step 9: Verify Secret Creation

* You should see ASSIGNMENT\_SECRET listed in your repository secrets
* The value will show as “Updated X seconds ago” but won’t display the actual token
* If you need to update it, click the secret name and choose “Update”

## 5.4 Phase 4: Testing Your Secret

### 5.4.1 Step 10: Trigger the Evaluation Action

1. Navigate to the “Actions” tab in your repository
2. Look for “GitHub Secret Evaluation” in the workflows list on the left
3. Click on “GitHub Secret Evaluation”
4. Click the “Run workflow” button (you may need to select branch “main”)
5. Click “Run workflow” again to confirm

Actions tab showing workflow list with "GitHub Secret Evaluation" highlighted and "Run workflow" button

### 5.4.2 Step 11: Monitor the Evaluation

1. Watch for the workflow to start running (you’ll see a yellow circle, then green checkmark or red X)
2. The evaluation will take a few moments to check if your secret exists and is valid
3. Don’t worry if it takes a minute - the system needs to verify everything is configured correctly

### 5.4.3 Step 12: Check the Results

1. Navigate to the “Issues” tab in your repository
2. Look for a new issue created by the evaluation system
3. The issue title will indicate success or failure of your secret storage
4. Click on the issue to see detailed results

## 5.5 Phase 5: Retrieving Your Validation Code

### 5.5.1 Step 13: Success Scenario

If your secret was stored correctly: - The issue will contain a validation code - Copy the entire validation code (it will be clearly marked) - Keep this code safe - you’ll need it for your Coursera quiz

### 5.5.2 Step 14: Failure Scenarios and Troubleshooting

#### 5.5.2.1 Secret Not Found:

* Check the secret name: Must be exactly ASSIGNMENT\_SECRET
* Verify location: Must be in repository secrets, not personal settings
* Re-create if needed: Delete and recreate the secret with correct name

### 5.5.3 Step 15: Re-running Evaluation

If you need to fix issues and re-test: 1. Make necessary corrections (recreate secret, new token, etc.) 2. Return to Actions tab 3. Run the “GitHub Secret Evaluation” workflow again 4. Check for a new issue with updated results

## 5.6 Phase 6: Completing the Assignment

### 5.6.1 Step 16: Submit Your Quiz

1. Navigate to your Coursera Capstone Quiz
2. Find the Assignment 3 question
3. Paste your validation code exactly as provided in the issue
4. Submit the quiz to receive credit

### 5.6.2 Step 17: Clean Up (Optional but Recommended)

After completing the assignment: 1. Consider revoking the PAT if you don’t plan to use it further 2. Delete the secret if it’s no longer needed 3. Document what you learned about secret management

## 5.7 Best Practices and Security Considerations

### 5.7.1 Token Management

* Use descriptive names for tokens to track their purpose
* Set appropriate expiration dates - shorter is more secure
* Regularly audit and rotate tokens in production environments
* Revoke unused tokens to minimize security exposure

### 5.7.2 Secret Storage Best Practices

* Use specific, descriptive names for secrets
* Apply principle of least privilege - only grant necessary permissions
* Never log or print secrets in your workflows
* Regularly review who has access to repository secrets

### 5.7.3 Research Context Applications

* API keys for data sources (e.g., cancer databases, genomic repositories)
* Authentication for computing clusters or cloud services
* Credentials for automated publication or preprint systems
* Tokens for collaborative tools and notification systems

## 5.8 Troubleshooting Common Issues

### 5.8.1 “Secret not found” Error

# 6 Check these items:

* Secret name is exactly: ASSIGNMENT\_SECRET
* Secret is stored in your copy of the capstone-sandbox repository settings
* The repository is the correct sandbox repository (not fhdsl/capstone-sandbox but instead your own copy)

*The following troubleshooting tips do not apply to the assignment itself but may apply in other scenarios where you are using a similar setup*:

### 6.0.1 “Invalid token” Error

Verify: - Token was copied correctly (no extra spaces) - Token hasn’t expired - Token has ‘repo’ scope permissions - Token was generated for the correct GitHub account

What to do to fix: - Generate a new PAT: Your token may have expired or been created incorrectly - Check scopes: Ensure “repo” permission is selected - Update the secret: Replace the old token with the new one

### 6.0.2 Workflow Doesn’t Run

Ensure: - You’re triggering from the correct repository - You’re selecting the main branch - The workflow file exists in .github/workflows/ - You have permissions to run workflows

This assignment teaches a crucial skill for modern computational research: secure credential management. In real research environments, you’ll use these same techniques to store API keys for cancer databases, authentication tokens for computing clusters, and credentials for automated research workflows. The ability to securely manage secrets enables sophisticated automation while maintaining security standards required for sensitive cancer research data.

# About the Authors

These credits are based on our [course contributors table guidelines](https://www.ottrproject.org/more_features.html#giving-credits-to-contributors).

| Credits | Names |
| --- | --- |
| **Pedagogy** |  |
| Lead Content Instructor(s) | [Candace Savonen](https://www.cansavvy.com/) |
| Lecturer(s) | [Candace Savonen](https://www.cansavvy.com/) |
| Content Contributor(s) | [Kate Isaac](https://kweav.github.io/), [Carrie Wright](https://carriewright11.github.io/) |
| Content Editor(s)/Reviewer(s) | [Carrie Wright](https://carriewright11.github.io/) |
| **Production** |  |
| Content Publisher(s) | [Carrie Wright](https://carriewright11.github.io/) |
| Content Publishing Reviewer(s) | [Carrie Wright](https://carriewright11.github.io/) |
| **Technical** |  |
| Course Publishing Engineer(s) | Helped with the code for the technical aspects related to the specific course generation |
| Template Publishing Engineers | [Candace Savonen](https://www.cansavvy.com/), [Carrie Wright](https://carriewright11.github.io/), [Ava Hoffman](https://www.avahoffman.com/) |
| Publishing Maintenance Engineer | [Candace Savonen](https://www.cansavvy.com/) |
| Technical Publishing Stylists | [Carrie Wright](https://carriewright11.github.io/), [Ava Hoffman](https://www.avahoffman.com/), [Candace Savonen](https://www.cansavvy.com/) |
| Package Developers ([ottrpal](https://github.com/jhudsl/ottrpal)) [Candace Savonen](https://www.cansavvy.com/), [Ava Hoffman](https://www.avahoffman.com/), [Howard Baek](https://www.linkedin.com/in/howard-baik/), [Kate Isaac](https://kweav.github.io/), [Carrie Wright](https://carriewright11.github.io/), [John Muschelli](https://johnmuschelli.com/) |  |
| **Art and Design** |  |
| Figure Artist(s) | [Carrie Wright](https://carriewright11.github.io/) [Carrie Wright](https://carriewright11.github.io/) |
| Videographer(s) | [Candace Savonen |
| **Funding** |  |
| Funder(s) | Institution/individual who funded course including grant number |
| Funding Staff | Staff members who help with funding |

## ─ Session info ───────────────────────────────────────────────────────────────  
## setting value  
## version R version 4.3.2 (2023-10-31)  
## os Ubuntu 22.04.4 LTS  
## system x86\_64, linux-gnu  
## ui X11  
## language (EN)  
## collate en\_US.UTF-8  
## ctype en\_US.UTF-8  
## tz Etc/UTC  
## date 2025-06-07  
## pandoc 3.1.1 @ /usr/local/bin/ (via rmarkdown)  
##   
## ─ Packages ───────────────────────────────────────────────────────────────────  
## package \* version date (UTC) lib source  
## bookdown 0.41 2024-10-16 [1] CRAN (R 4.3.2)  
## cachem 1.0.8 2023-05-01 [1] RSPM (R 4.3.0)  
## cli 3.6.2 2023-12-11 [1] RSPM (R 4.3.0)  
## devtools 2.4.5 2022-10-11 [1] RSPM (R 4.3.0)  
## digest 0.6.34 2024-01-11 [1] RSPM (R 4.3.0)  
## ellipsis 0.3.2 2021-04-29 [1] RSPM (R 4.3.0)  
## evaluate 0.23 2023-11-01 [1] RSPM (R 4.3.0)  
## fastmap 1.1.1 2023-02-24 [1] RSPM (R 4.3.0)  
## fs 1.6.3 2023-07-20 [1] RSPM (R 4.3.0)  
## glue 1.7.0 2024-01-09 [1] RSPM (R 4.3.0)  
## htmltools 0.5.7 2023-11-03 [1] RSPM (R 4.3.0)  
## htmlwidgets 1.6.4 2023-12-06 [1] RSPM (R 4.3.0)  
## httpuv 1.6.14 2024-01-26 [1] RSPM (R 4.3.0)  
## knitr 1.48 2024-07-07 [1] CRAN (R 4.3.2)  
## later 1.3.2 2023-12-06 [1] RSPM (R 4.3.0)  
## lifecycle 1.0.4 2023-11-07 [1] RSPM (R 4.3.0)  
## magrittr 2.0.3 2022-03-30 [1] RSPM (R 4.3.0)  
## memoise 2.0.1 2021-11-26 [1] RSPM (R 4.3.0)  
## mime 0.12 2021-09-28 [1] RSPM (R 4.3.0)  
## miniUI 0.1.1.1 2018-05-18 [1] RSPM (R 4.3.0)  
## pkgbuild 1.4.3 2023-12-10 [1] RSPM (R 4.3.0)  
## pkgload 1.3.4 2024-01-16 [1] RSPM (R 4.3.0)  
## profvis 0.3.8 2023-05-02 [1] RSPM (R 4.3.0)  
## promises 1.2.1 2023-08-10 [1] RSPM (R 4.3.0)  
## purrr 1.0.2 2023-08-10 [1] RSPM (R 4.3.0)  
## R6 2.5.1 2021-08-19 [1] RSPM (R 4.3.0)  
## Rcpp 1.0.12 2024-01-09 [1] RSPM (R 4.3.0)  
## remotes 2.4.2.1 2023-07-18 [1] RSPM (R 4.3.0)  
## rlang 1.1.4 2024-06-04 [1] CRAN (R 4.3.2)  
## rmarkdown 2.25 2023-09-18 [1] RSPM (R 4.3.0)  
## sessioninfo 1.2.2 2021-12-06 [1] RSPM (R 4.3.0)  
## shiny 1.8.0 2023-11-17 [1] RSPM (R 4.3.0)  
## stringi 1.8.3 2023-12-11 [1] RSPM (R 4.3.0)  
## stringr 1.5.1 2023-11-14 [1] RSPM (R 4.3.0)  
## urlchecker 1.0.1 2021-11-30 [1] RSPM (R 4.3.0)  
## usethis 2.2.3 2024-02-19 [1] RSPM (R 4.3.0)  
## vctrs 0.6.5 2023-12-01 [1] RSPM (R 4.3.0)  
## xfun 0.48 2024-10-03 [1] CRAN (R 4.3.2)  
## xtable 1.8-4 2019-04-21 [1] RSPM (R 4.3.0)  
## yaml 2.3.8 2023-12-11 [1] RSPM (R 4.3.0)  
##   
## [1] /usr/local/lib/R/site-library  
## [2] /usr/local/lib/R/library  
##   
## ──────────────────────────────────────────────────────────────────────────────

# 7 References