**Week 1 Teacher's Guide: "Setting Up Your Gear"**

**Professional Development Environment & Mindset**

**Week:** February 17-21, 2025  
**Total Hours:** 30 hours (Mon-Thu: 6.5h/day, Fri: 4h)  
**Theme:** Establishing professional foundation before writing code

**Week Overview & Teaching Philosophy**

**The "Gig Ready" Foundation**

This week is **critical** - students must understand that being a professional developer is about more than writing code. Use the musician analogy repeatedly:

* A musician doesn't just learn chords; they learn to set up equipment, work with a band, read the room, handle technical difficulties
* Similarly, developers need professional tools, workflows, and communication skills

**Key Teaching Principles for Week 1:**

1. **Don't rush into coding** - resist student pressure to "start programming now"
2. **Model professional behavior** - how you communicate, handle mistakes, document work
3. **Create psychological safety** - normalize not knowing, asking questions, making mistakes
4. **Establish professional habits** - these habits will carry through the entire program

**Week 1 Success Metrics:**

* Every student has working development environment
* Every student has made commits and pull requests
* Students use professional terminology naturally
* Students participate in daily standups comfortably
* Class culture of code review and feedback is established

**Daily Breakdown**

**DAY 1: Monday, February 17, 2025**

**"Welcome to Professional Development" (6.5 hours)**

**Schedule:**

* 08:30-09:00: Registration, Setup, Welcome Coffee
* 09:00-10:30: "Gig Ready" Introduction & Course Overview
* 10:30-10:45: Break
* 10:45-12:00: Command Line Fundamentals
* 12:00-12:30: Lunch Break
* 12:30-14:30: Command Line Practice & Professional Identity Setup
* 14:30-15:00: Daily Standup Practice & Day 1 Reflection
* 15:00-15:30: Break & Networking

**Session 1: "Gig Ready" Introduction (1.5 hours)**

**Learning Objectives:**

* Understand what "gig ready" means
* See the complete picture of professional development
* Build excitement and appropriate expectations
* Understand why we're starting with tools, not code

**Materials Needed:**

* Projector for "Day in the Life of Sarah" speech
* Printed course syllabus for each student
* Name tags
* Whiteboard for capturing questions

**Session Structure:**

**Opening (15 min):**

* Instructor introduction using Frank's background
* "I've worked with hundreds of developers over 25 years. The ones who succeed have something in common - and it's not that they're the best coders..."
* Set up the musician analogy

**"Day in the Life of Sarah" Speech (30 min):** Use the speech we created earlier, enhanced with visuals:

"Sarah arrives at work and opens her terminal to pull the latest changes

from the main branch using git. She checks Slack for any blockers from

yesterday's standup, then reviews two pull requests from her teammates -

one has a tricky JavaScript async/await function that she suggests could

be refactored, and another has a SQL query that's missing an index.

She opens Jira to grab her next user story from the current sprint backlog -

it's about adding a new API endpoint that needs to fetch user data with a

complex JOIN query and return it as JSON for the React frontend. She spins

up her local Docker containers, runs the test suite to make sure everything's

green, then starts coding.

Halfway through implementing the Express route handler, she realizes the

frontend team expects the data in a different format. She checks the staging

environment API documentation in Swagger, then modifies her JavaScript

response mapping. She SSH's into the server to check the application logs

for any similar formatting issues.

After lunch, she adds error handling for database connection failures, writes

unit tests for her new JavaScript functions, commits her changes with a clear

commit message, pushes to her feature branch, and opens a pull request. The

automated tests run her JavaScript test suite and check for SQL injection

vulnerabilities.

During the afternoon retrospective, the team discusses why the last deployment

to production was slow - turns out one of the SQL queries was doing a full

table scan. Sarah volunteers to research database optimization and

containerization best practices for their next sprint planning session.

Before heading home, she updates her story's status in Jira and adds notes

for tomorrow's standup about the database migration script she's writing

that will need review."

**After the speech (15 min):**

* "Right now, probably 80% of what Sarah did sounds like gibberish"
* Write all the unfamiliar terms on whiteboard
* "By the end of 3 months, you'll understand every single thing she did"
* "More importantly - you'll BE Sarah"

**Course Overview (30 min):**

* Walk through syllabus highlighting the professional skills
* Show the progression: Month 1 (foundations), Month 2 (building), Month 3 (professional delivery)
* Explain Agile structure: sprints, standups, retrospectives
* Set expectations: this is intensive, full-time, professional training
* Answer questions

**Break (15 min)**

**Session 2: Command Line Fundamentals (1.25 hours)**

**Learning Objectives:**

* Navigate file system using terminal
* Create, move, copy, delete files and directories
* Understand paths (absolute vs relative)
* Feel comfortable with command line interface

**Teaching Approach:**

* **Model first:** Instructor demonstrates on projected screen
* **Explain thinking:** "I'm going to check where I am first using pwd..."
* **Students follow:** Students replicate on their machines
* **Add complexity gradually:** Start simple, build up

**Core Commands to Teach:**

**Navigation & Information:**

bash

pwd *# Where am I?*

ls *# What's here?*

ls -la *# Show me everything, including hidden files*

cd Documents *# Go to Documents*

cd .. *# Go up one level*

cd ~ *# Go home*

cd - *# Go to previous directory*

**File Operations:**

bash

mkdir projects *# Create directory*

mkdir -p projects/week1 *# Create nested directories*

touch README.md *# Create empty file*

cp file.txt backup.txt *# Copy file*

mv oldname.txt newname.txt *# Rename/move file*

rm file.txt *# Delete file (careful!)*

rm -r folder *# Delete directory (very careful!)*

**Viewing Files:**

bash

cat file.txt *# Display file contents*

less file.txt *# View file (scrollable)*

head file.txt *# First 10 lines*

tail file.txt *# Last 10 lines*

**Professional Practices to Emphasize:**

* Always use pwd before running destructive commands
* Use ls frequently to verify where you are
* Tab completion is your friend
* Use clear to clean up terminal when it gets messy
* Command history (up arrow) saves time

**Hands-On Exercise (30 min):** Create a project structure from command line:

bash

*# Students create this structure*

bootcamp/

├── week1/

│ ├── notes.md

│ └── exercises/

├── week2/

└── projects/

└── portfolio/

├── index.html

├── style.css

└── README.md

**Commands they'll use:**

bash

mkdir bootcamp

cd bootcamp

mkdir week1 week2 projects

cd week1

touch notes.md

mkdir exercises

cd ../projects

mkdir portfolio

cd portfolio

touch index.html style.css README.md

ls -la

**Common Issues & Solutions:**

* **Spaces in names:** Teach cd "My Folder" or cd My\ Folder
* **Wrong directory:** Regular use of pwd and ls
* **Can't find command:** PATH explanation (keep simple)
* **Deleted wrong file:** Use this as teaching moment about being careful

**Resources:**

* Command Line Crash Course: <https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Understanding_client-side_tools/Command_line>
* Bash Cheat Sheet: <https://github.com/RehanSaeed/Bash-Cheat-Sheet>
* Practice: <https://cmdchallenge.com/>

**Lunch Break (30 min)**

**Session 3: Command Line Practice & Professional Identity (2 hours)**

**Part A: Advanced Command Line (45 min)**

**Environment Variables:**

bash

echo $HOME *# See HOME environment variable*

echo $PATH *# See PATH*

export NAME="Frank" *# Set variable (temporary)*

echo $NAME

**Useful Professional Commands:**

bash

history *# See command history*

!number *# Run command from history*

grep "text" file *# Search in files*

find . -name "\*.js" *# Find files*

which node *# Find where command is installed*

alias ll='ls -la' *# Create shortcut*

**Piping & Redirection (introduce, don't deep dive):**

bash

ls -la > files.txt *# Write output to file*

cat file.txt | grep "search" *# Pipe output to another command*

ls | wc -l *# Count files in directory*

**Exercise: Professional File Organization** Students create their bootcamp workspace:

bash

mkdir -p ~/bootcamp/{projects,resources,notes}

cd ~/bootcamp

touch notes/daily-journal.md

echo "# My Bootcamp Journey" > notes/daily-journal.md

cat notes/daily-journal.md

**Part B: Professional Identity Setup (1 hour)**

**Why This Matters:** "Before you write code, you need professional presence. When employers look you up, what will they find?"

**GitHub Account Setup (30 min):**

1. **Create Account** (if needed): <https://github.com/>
   * Professional username (avoid "coolcoder69")
   * Professional email
   * Professional profile photo
2. **Profile Setup:**

markdown

Example good profile:

Name: [First Last]

Bio: Full-stack developer | JavaScript, React, Node.js | Digital Campus Vorarlberg Bootcamp

Location: Feldkirch, Austria

Website: [coming soon]

Pinned repositories will show your best work

1. **Profile README (Optional but impressive):**
   * Show example: <https://github.com/abhisheknaiidu/awesome-github-profile-readme>
   * "We'll create yours as the course progresses"

**LinkedIn Profile Setup (30 min):**

1. **Create/Update Profile:**
   * Professional photo (same as GitHub)
   * Professional headline: "Full-Stack Developer Student at Digital Campus Vorarlberg"
   * About section example:

Currently completing an intensive 300-hour Software Development Bootcamp

at Digital Campus Vorarlberg, focusing on professional full-stack

development with JavaScript, React, Node.js, and modern development

practices including Git, Agile methodology, and test-driven development.

Learning to code is one thing - learning to work as a professional

developer is another. This program emphasizes the professional practices

that make developers valuable to teams from day one.

Connect with me as I document my journey from beginner to gig-ready

professional developer.

1. **Add Current Education:**
   * School: Digital Campus Vorarlberg
   * Degree: Software Development Bootcamp
   * Start: February 2025
   * Description: "Intensive full-stack development program covering JavaScript, React, Node.js, databases, and professional development practices"
2. **Connect with Classmates:**
   * Have students find and connect with each other
   * Connect with instructor
   * "This is your professional network - it starts now"

**Professional Email Setup (if needed):**

* Use professional email for all course communication
* Not "[partykid@email.com](mailto:partykid@email.com)" - use "[firstname.lastname@email.com](mailto:firstname.lastname@email.com)" or similar

**Resources:**

* GitHub Profile Guide: <https://docs.github.com/en/account-and-profile/setting-up-and-managing-your-github-profile>
* LinkedIn Profile Tips: <https://www.linkedin.com/business/talent/blog/product-tips/linkedin-profile-summaries-that-we-love-and-how-to-boost-your-own>

**Session 4: Daily Standup Practice & Reflection (45 min)**

**Introducing Agile Standup (15 min):**

"Every professional development team has daily standups. Let's practice now."

**Standup Format:** Each person answers three questions (60-90 seconds):

1. What did I do yesterday?
2. What will I do today?
3. Any blockers?

**Today's Modified Format (since it's Day 1):**

1. Who am I and why am I here?
2. What's one thing that excited me today?
3. What's one thing that confused or worried me?

**Practice (20 min):**

* Stand in circle (or semi-circle if remote)
* Instructor models first
* Go around circle
* Keep time - 90 seconds max per person
* No discussion - just updates
* Note blockers to address later

**Reflection & Journaling (10 min):**

Students open their daily-journal.md:

bash

cd ~/bootcamp/notes

code daily-journal.md *# or use preferred editor*

Add entry:

markdown

# Daily Journal

## Day 1 - Monday, February 17, 2025

### What I Learned Today

-

-

-

### Commands I Used

-

-

-

### Questions I Still Have

-

-

### Tomorrow's Goals

-

-

**Homework Assignment Review:** Walk through the assignment expectations (see assignment section below)

**Day 1 Homework Assignment**

**Assignment: "Professional Foundation Setup"**

**Due:** Before class tomorrow (students should aim to complete tonight)

**Requirements:**

1. **Complete Environment Setup:**
   * Verify command line access
   * Create bootcamp directory structure
   * GitHub account configured with professional profile
   * LinkedIn profile created/updated
2. **Practice Command Line (30 min):**
   * Complete exercises at: <https://cmdchallenge.com/> (first 10 challenges)
   * Create a notes file documenting 10 commands you learned today
   * Use command line to create tomorrow's workspace
3. **Professional Identity:**
   * Take/upload professional profile photo to GitHub and LinkedIn
   * Write LinkedIn "About" section (use template provided)
   * Connect with at least 5 classmates on LinkedIn
4. **Reflection:**
   * Complete daily journal entry
   * Write one paragraph about what "gig ready" means to you
5. **Reading (optional but recommended):**
   * The Professional Programmer's Mindset: <https://www.freecodecamp.org/news/professional-programmer-mindset/>
   * Why Git: <https://www.atlassian.com/git/tutorials/what-is-git>

**Submission:**

* No formal submission
* Instructor will verify in tomorrow's standup
* Be ready to share one command line thing you learned

**Day 1 Teaching Notes & Tips**

**Time Management:**

* First day often runs long - students need setup help
* Build in buffer time
* Better to end early than rush

**Common Day 1 Issues:**

1. **Windows vs Mac vs Linux differences:**
   * Have cheat sheet ready for Windows (PowerShell/GitBash)
   * Pair Mac users with Mac users for troubleshooting
2. **Students want to code now:**
   * Validate their excitement
   * Reinforce: "Professional tools first, then we code"
   * "Week 4 you'll be coding daily - this foundation matters"
3. **Some students feel behind already:**
   * Normalize: "Everyone feels overwhelmed Day 1"
   * Pair stronger students with those struggling
   * Office hours availability
4. **Technical issues:**
   * Have backup computers available
   * Pre-install software on school machines
   * Technical support contact info ready

**Creating Class Culture:**

* Model asking questions: "I don't know, let's find out together"
* Celebrate mistakes: "Great! Now we all learned what NOT to do"
* Encourage peer help: "Who figured this out? Can you help [name]?"
* Use names frequently
* Set tone: professional but friendly

**Assessment:**

* Observe during hands-on: who's struggling?
* Note who helps others (leadership potential)
* Check understanding through questions, not tests
* Mental note of who might need extra support

**DAY 2: Tuesday, February 18, 2025**

**"Git Version Control - Your Professional Safety Net" (6.5 hours)**

**Schedule:**

* 08:30-09:00: Daily Standup
* 09:00-10:30: Git Fundamentals Lecture & Demo
* 10:30-10:45: Break
* 10:45-12:00: Git Hands-On Practice
* 12:00-12:30: Lunch Break
* 12:30-14:00: Branching & Merging
* 14:00-14:15: Break
* 14:15-15:15: Collaboration with GitHub
* 15:15-15:30: Day 2 Standup & Reflection

**Session 1: Morning Standup (30 min)**

**Structure:**

* 15 min: Standup (each student ~60 seconds)
* 15 min: Address blockers from yesterday, answer questions

**Standup Questions Today:**

1. What did you work on last night (homework)?
2. What command line command is your new favorite?
3. Any blockers?

**Instructor Notes:**

* Model good standup: concise, factual, no problem-solving during standup
* If discussions start, say "let's take that offline"
* Write blockers on board to address after

**Session 2: Git Fundamentals (1.5 hours)**

**Learning Objectives:**

* Understand why version control exists
* Learn Git basics: init, add, commit, status, log
* Understand the Git workflow
* Write professional commit messages

**The "Why Git?" Story (15 min):**

Use a relatable analogy:

"Imagine you're writing a paper. You make Version 1, then Version 2, then

Version 2 final, Version 2 final FINAL, Version 2 final FINAL for real this

time...

Now imagine you're working on code with 5 other people. Everyone is editing

files. How do you:

- Keep track of who changed what?

- Go back if someone breaks something?

- Work on new features without breaking the main code?

- Merge everyone's work together?

That's why Git exists. It's your time machine and your collaboration tool."

**Show the Problem Without Git (10 min):**

Create a simple file, make changes, demonstrate the chaos:

bash

*# Create file*

echo "Hello World" > app.js

*# Make changes*

echo "Hello World v2" > app.js

*# Oh no! Where's my original?*

*# Can't see history*

*# Can't collaborate*

*# Can't experiment safely*

**Introduce Git (15 min):**

"Git tracks every change, keeps history, lets you experiment safely"

**Key Concepts:**

* **Repository (repo):** Project folder that Git tracks
* **Commit:** Snapshot of your project at a point in time
* **Working Directory:** Files you're currently editing
* **Staging Area:** Changes you're preparing to commit
* **History:** All your past commits

**Visual on Whiteboard:**

Working Directory → Staging Area → Repository

(edit) → (git add) → (git commit)

Your files → Changes ready → Permanent snapshot

to commit in history

**Demo Time (30 min):**

**Setup:**

bash

*# Check Git is installed*

git --version

*# Configure Git (first time only)*

git config --global user.name "Frank Blau"

git config --global user.email "frank@example.com"

*# Check configuration*

git config --list

**First Repository:**

bash

*# Create project*

mkdir my-first-repo

cd my-first-repo

*# Initialize Git*

git init

*# Explain: This creates hidden .git folder that tracks everything*

*# Check status*

git status

*# Explain: "On branch main, No commits yet, nothing to commit"*

**First Commit:**

bash

*# Create a file*

echo "# My First Project" > README.md

*# Check status*

git status

*# Point out: "Untracked files" - Git sees it but isn't tracking it*

*# Add to staging*

git add README.md

*# Check status again*

git status

*# Point out: "Changes to be committed" - now Git is tracking it*

*# Commit*

git commit -m "Add README file"

*# Explain the message: clear, present tense, describes what changed*

*# Check status*

git status

*# Point out: "working tree clean" - everything is committed*

*# See history*

git log

*# Point out: commit hash, author, date, message*

**More Changes:**

bash

*# Edit file*

echo "This is my learning journal" >> README.md

*# Check what changed*

git status

*# Point out: "modified" not "untracked"*

git diff

*# Explain: Shows what changed (+ for additions, - for deletions)*

*# Commit the change*

git add README.md

git commit -m "Add description to README"

*# See history*

git log

git log --oneline *# Shorter format*

**Professional Commit Messages (10 min):**

**Bad Commit Messages:**

bash

git commit -m "stuff"

git commit -m "fixed it"

git commit -m "asdfasdf"

git commit -m "updates"

**Good Commit Messages:**

bash

git commit -m "Add user authentication feature"

git commit -m "Fix bug in login validation"

git commit -m "Update README with installation instructions"

git commit -m "Remove deprecated API endpoints"

**Professional Format:**

* Present tense ("Add" not "Added")
* Imperative mood ("Fix" not "Fixes")
* Clear and specific
* Under 50 characters for first line
* If need more detail, add blank line then paragraph

**Resources:**

* Git Documentation: <https://git-scm.com/doc>
* Pro Git Book (free): <https://git-scm.com/book/en/v2>
* Git Cheat Sheet: <https://education.github.com/git-cheat-sheet-education.pdf>
* Commit Message Guide: <https://www.conventionalcommits.org/>

**Break (15 min)**

**Session 3: Git Hands-On Practice (1.25 hours)**

**Exercise 1: Personal Journal Repository (30 min)**

Students create their own repository:

bash

cd ~/bootcamp

mkdir journal

cd journal

git init

*# Create first entry*

echo "# Development Journal" > README.md

echo "" >> README.md

echo "## Day 2 - Learning Git" >> README.md

echo "Today I learned about version control..." >> README.md

git add README.md

git commit -m "Initialize journal with Day 2 entry"

*# Add more entries*

echo "" >> README.md

echo "### Key Concepts" >> README.md

echo "- Repository: tracked project folder" >> README.md

echo "- Commit: snapshot of changes" >> README.md

git add README.md

git commit -m "Add key concepts section"

*# Check history*

git log --oneline

**Exercise 2: Making Mistakes & Fixing Them (20 min)**

"Let's learn to undo things - because you WILL make mistakes"

bash

*# Make a change*

echo "This is a mistake" >> README.md

*# Oh no! I didn't want that*

git status

*# Show: modified file*

*# Undo changes (before adding)*

git restore README.md

*# OR older syntax: git checkout -- README.md*

*# Check - changes gone*

cat README.md

*# Now let's say you staged something by mistake*

echo "Another mistake" >> README.md

git add README.md

git status

*# Unstage it*

git restore --staged README.md

*# OR older syntax: git reset HEAD README.md*

*# Then restore the file*

git restore README.md

**Common Git Operations:**

bash

*# See what you changed*

git diff

*# See staged changes*

git diff --staged

*# Remove file from Git tracking*

git rm filename

*# Rename file*

git mv oldname newname

*# See commit history*

git log

git log --oneline

git log --graph

git log -p *# shows diffs*

*# Ignore files*

echo "node\_modules/" > .gitignore

echo ".DS\_Store" >> .gitignore

git add .gitignore

git commit -m "Add gitignore file"

**Instructor Circulates:**

* Help students having issues
* Check commit messages
* Answer questions
* Identify who needs extra help

**Exercise 3: Project Time Capsule (30 min)**

"Create a project and commit its evolution"

bash

cd ~/bootcamp/projects

mkdir time-capsule

cd time-capsule

git init

*# Version 1*

echo "<h1>Hello World</h1>" > index.html

git add index.html

git commit -m "Create initial HTML file"

*# Version 2*

echo "<p>This is my first web page</p>" >> index.html

git add index.html

git commit -m "Add paragraph to page"

*# Version 3*

cat > styles.css << EOF

h1 {

color: blue;

}

EOF

git add styles.css

git commit -m "Add CSS stylesheet"

*# Version 4*

*# Edit HTML to link CSS*

echo '<link rel="stylesheet" href="styles.css">' | cat - index.html > temp && mv temp index.html

git add index.html

git commit -m "Link CSS to HTML"

*# See the evolution*

git log --oneline

**Lunch Break (30 min)**

**Session 4: Branching & Merging (1.5 hours)**

**Learning Objectives:**

* Understand why branches exist
* Create and switch between branches
* Merge branches
* Resolve basic merge conflicts

**The Branch Concept (20 min):**

"Imagine you're working on an app. It's live and working. Now you want to add a new feature. But what if you break something? Branches let you experiment safely."

**Draw on Whiteboard:**

main branch: o---o---o---o

\

feature branch: o---o---o

"The main branch is your stable code. Feature branches are experiments. When the experiment works, you merge it back."

**Demo: Basic Branching (30 min):**

bash

cd ~/bootcamp/journal

git status *# Make sure we're on main*

*# See branches*

git branch

*# Shows: \* main (asterisk shows current branch)*

*# Create new branch*

git branch feature-add-skills

git branch *# See both branches*

*# Switch to new branch*

git checkout feature-add-skills

*# Or newer syntax: git switch feature-add-skills*

*# Make changes on this branch*

echo "" >> README.md

echo "## Skills I'm Learning" >> README.md

echo "- Command Line" >> README.md

echo "- Git" >> README.md

git add README.md

git commit -m "Add skills section"

*# Switch back to main*

git checkout main

*# Notice: skills section is gone!*

cat README.md

*# Now merge the feature*

git merge feature-add-skills

*# Skills section is back!*

cat README.md

*# See history*

git log --oneline --graph

**Professional Branch Naming:**

bash

*# Good branch names:*

feature/user-authentication

fix/login-bug

refactor/database-connection

hotfix/security-patch

*# Bad branch names:*

my-changes

test

branch1

asdf

**Merge Conflicts (30 min):**

"Sometimes Git can't automatically merge. Let's create that situation."

bash

*# On main branch*

git checkout main

echo "Line added on main" >> README.md

git add README.md

git commit -m "Add line on main"

*# Create and switch to new branch*

git checkout -b feature-conflict

*# Edit the SAME file in a different way*

echo "Line added on feature" >> README.md

git add README.md

git commit -m "Add line on feature"

*# Try to merge*

git checkout main

git merge feature-conflict

*# CONFLICT!*

*# Git shows: "CONFLICT (content): Merge conflict in README.md"*

**Resolving Conflicts:**

bash

*# Open the file - see conflict markers*

cat README.md

*# Conflict looks like:*

<<<<<<< HEAD

Line added on main

=======

Line added on feature

>>>>>>> feature-conflict

*# Explain markers:*

*# <<<<<<< HEAD - this is main branch version*

*# ======= - separator*

*# >>>>>>> feature-conflict - this is feature branch version*

**Resolution Process:**

1. Open file in editor
2. Decide what to keep (or combine both)
3. Remove conflict markers
4. Save file
5. Add and commit

bash

*# After editing*

git add README.md

git commit -m "Resolve merge conflict"

*# Conflict resolved!*

**Practice Exercise (20 min):**

Students practice branching:

bash

*# Create a branch for each day*

git checkout -b day2-learning

*# Add what they learned today*

git add .

git commit -m "Add Day 2 learnings"

git checkout main

git merge day2-learning

**Resources:**

* Git Branching: <https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell>
* Interactive Git Tutorial: <https://learngitbranching.js.org/>
* Resolving Conflicts: <https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/addressing-merge-conflicts>

**Break (15 min)**

**Session 5: GitHub & Collaboration (1 hour)**

**Learning Objectives:**

* Connect local Git to GitHub
* Push and pull code
* Understand remote repositories
* Create first pull request

**GitHub vs Git (10 min):**

"Git is the tool on your computer. GitHub is the website where you store and share your Git repositories."

**Analogy:**

* Git = Microsoft Word (the software)
* GitHub = Google Drive (the cloud storage)

**Demo: First GitHub Repository (40 min):**

**Step 1: Create GitHub Repo**

* Go to github.com
* Click "New Repository"
* Name: first-repo
* Description: My first GitHub repository
* Public
* DO NOT initialize with README (we already have one)
* Create repository

**Step 2: Connect Local Repo to GitHub**

bash

cd ~/bootcamp/journal

*# Add remote*

git remote add origin https://github.com/yourusername/first-repo.git

*# Verify remote*

git remote -v

*# Push code*

git push -u origin main

*# Explain: -u sets upstream, origin is the remote name, main is the branch*

**Step 3: View on GitHub**

* Refresh GitHub page
* See your code!
* Point out: commit history, files, README renders

**Step 4: Make Changes on GitHub**

* Edit README directly on GitHub
* Commit changes (GitHub commits for you)

**Step 5: Pull Changes Locally**

bash

*# Fetch and merge changes*

git pull origin main

*# See the changes*

cat README.md

**The Professional Workflow:**

1. Pull latest changes: git pull

2. Create branch: git checkout -b feature-name

3. Make changes locally: (edit files)

4. Commit changes: git add . && git commit -m "message"

5. Push branch: git push origin feature-name

6. Create Pull Request on GitHub

7. Review and merge

8. Pull merged changes: git checkout main && git pull

**Class Exercise: Collaborative Repository (10 min setup, homework to complete):**

Instructor creates class repository:

1. Create repo: "bootcamp-2025-cohort"
2. Add all students as collaborators
3. Create folders: "day1", "day2", etc.
4. Each student will add their journal entry

bash

*# Students clone the repo*

git clone https://github.com/instructor/bootcamp-2025-cohort.git

cd bootcamp-2025-cohort

*# Create their own file*

mkdir day2

cd day2

echo "# [Your Name]'s Day 2 Journal" > yourname-day2.md

echo "Today I learned..." >> yourname-day2.md

*# Commit*

git add yourname-day2.md

git commit -m "Add [Your Name]'s Day 2 journal"

*# Push*

git push origin main

**Resources:**

* GitHub Docs: <https://docs.github.com/en>
* GitHub Hello World: <https://docs.github.com/en/get-started/quickstart/hello-world>
* GitHub Flow: <https://docs.github.com/en/get-started/quickstart/github-flow>

**Session 6: Day 2 Standup & Reflection (15 min)**

**Standup:**

1. What I learned about Git today
2. One Git command I feel confident about
3. One thing I'm still confused about

**Reflection:** Update journal with today's learning

**Day 2 Homework**

**Assignment: "Git Workflow Mastery"**

1. **Create Personal Portfolio Repo (1 hour):**

bash

*# Create local repo*

mkdir portfolio

cd portfolio

git init

*# Create files*

touch index.html style.css README.md

*# Add content to README*

echo "# My Portfolio" > README.md

*# Make commits*

git add .

git commit -m "Initial commit"

*# Create on GitHub and push*

1. **Practice Branching (30 min):**
   * Create branch "feature-about-page"
   * Add about.html
   * Commit changes
   * Merge to main
   * Push to GitHub
2. **Contribute to Class Repo:**
   * Clone class repository
   * Add your Day 2 journal entry
   * Push changes
   * View everyone's entries on GitHub
3. **Reading:**
   * Git Basics: <https://git-scm.com/book/en/v2/Getting-Started-Git-Basics>
   * GitHub Flow: <https://guides.github.com/introduction/flow/>
4. **Reflection:**
   * Update daily journal
   * Write: "How will Git help me as a professional developer?"

**Teaching Notes for Days 3-5**

**Day 3: IDE Setup & Code Quality**

* VS Code installation and configuration
* Extensions for professional development
* Code formatting and linting
* Professional code organization

**Day 4: Agile Deep Dive & Project Management**

* Agile ceremonies in detail
* Creating user stories
* Estimation and planning poker
* Using Jira/GitHub Projects

**Day 5: Week 1 Integration & Review**

* Complete end-to-end workflow
* Set up first real project
* Code review session
* Sprint retrospective
* Week 1 assignment finalization

**Week 1 Resources Library**

**Command Line**

* MDN Command Line Tutorial: <https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Understanding_client-side_tools/Command_line>
* Command Challenge: <https://cmdchallenge.com/>
* Bash Guide for Beginners: <https://tldp.org/LDP/Bash-Beginners-Guide/html/>

**Git**

* Pro Git Book: <https://git-scm.com/book/en/v2>
* Git Cheat Sheet: <https://education.github.com/git-cheat-sheet-education.pdf>
* Learn Git Branching: <https://learngitbranching.js.org/>
* Oh Shit, Git!: <https://ohshitgit.com/>
* Git Flight Rules: <https://github.com/k88hudson/git-flight-rules>

**GitHub**

* GitHub Docs: <https://docs.github.com/en>
* GitHub Learning Lab: <https://lab.github.com/>
* GitHub Flow Guide: <https://guides.github.com/introduction/flow/>

**Professional Development**

* The Professional Programmer: <https://www.freecodecamp.org/news/how-to-think-like-a-programmer-lessons-in-problem-solving-d1d8bf1de7d2/>
* Clean Code Summary: <https://gist.github.com/wojteklu/73c6914cc446146b8b533c0988cf8d29>
* Conventional Commits: <https://www.conventionalcommits.org/>

**Assessment Rubric for Week 1**

**Professional Environment Setup (25%)**

* Functional command line proficiency
* Git installed and configured
* GitHub account professional
* LinkedIn profile professional
* Project directory structure organized

**Git Skills (40%)**

* Can init, add, commit
* Writes professional commit messages
* Can create and merge branches
* Can resolve basic merge conflicts
* Can push/pull from GitHub
* Understands Git workflow

**Professional Practices (25%)**

* Participates in standups effectively
* Helps classmates
* Asks good questions
* Documents learning
* Meets deadlines
* Professional communication

**Homework Completion (10%)**

* All assignments submitted
* Quality meets expectations
* Evidence of practice and learning

**Troubleshooting Guide**

**Common Week 1 Issues**

**Git Installation Issues:**

* **Mac:** Install Xcode command line tools: xcode-select --install
* **Windows:** Download Git for Windows: <https://git-scm.com/download/win>
* **Linux:** sudo apt-get install git or sudo yum install git

**SSH vs HTTPS:**

* Week 1: Use HTTPS (simpler)
* Week 2+: Introduce SSH keys
* HTTPS means password every push (can cache credentials)

**Merge Conflicts Paralysis:**

* Students often freeze when seeing conflict markers
* Practice with small, obvious conflicts first
* "Conflict markers are just Git saying 'You decide!'"

**Push Rejected:**

* Usually means remote has changes
* Solution: git pull then git push
* Explain: Always pull before push

**Lost Changes:**

* git reflog is your friend
* Demo this early: "Git never loses anything"

**Branch Confusion:**

* Students forget which branch they're on
* Make them use git status constantly
* Consider adding branch to terminal prompt

**Week 1 Success Indicators**

By end of Week 1, students should:

* Be comfortable in terminal
* Make Git commits without thinking
* Understand professional workflows
* Participate confidently in standups
* Help classmates
* Be excited about learning
* Feel part of a cohort
* Understand "gig ready" mindset
* Have professional online presence
* Be ready to start coding in Week 2

**Remember:** Week 1 sets the foundation for everything else. Take the time to get it right. Students who master these professional practices early will excel throughout the program.