Intro to Git

INFORMS Auburn Student Chapter

Presented March 2, Spring 2022

Dan O’Leary // dan.oleary@auburn.edu

About me…

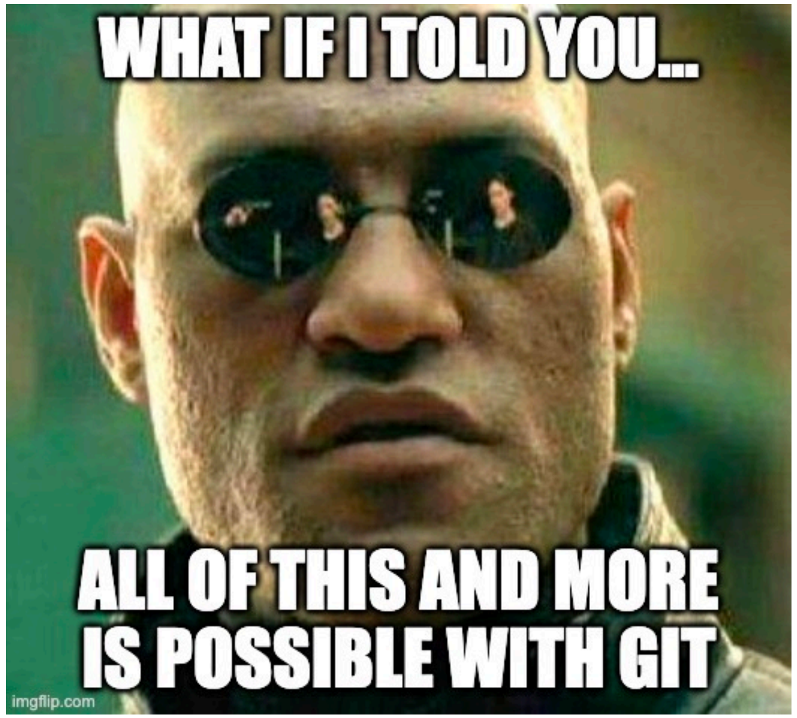
- BS ME AU 1992, Master of Engineering Management AU 2020
- 3rd year PhD student, ISE and full-time instructor in the department:
  - ENGR/BUSI 3520+60+10; INSY 3010 (Python/SQL), 6600, 7730+40
- Dissertation: mixed reality for distance learning, ICAMS
- Plan to defend Spring 2023
- Previously owner-operator of 2 software development companies
  - n-Space: game development studio, 23 years, 44 games
  - GUNSTRUCTION: innovative 3D configurator app
- NOT a software engineer

Have you ever…

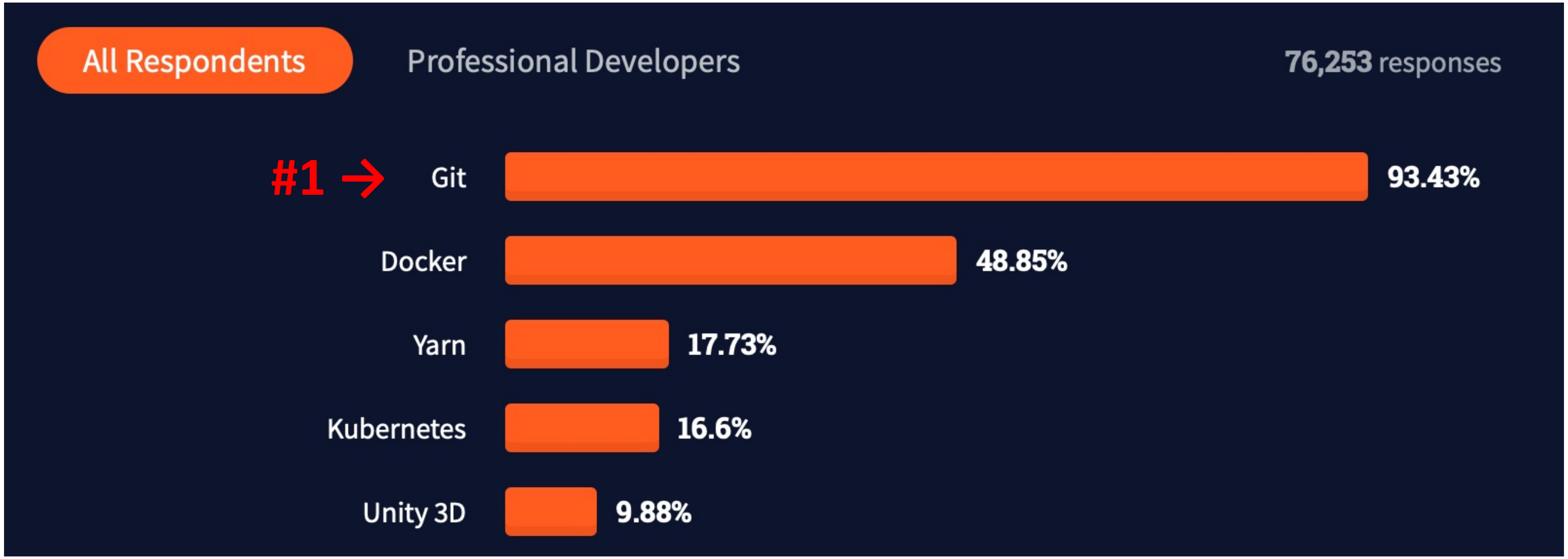
- Named a file `project_final_2a_updated(2).py.txt`?
- Tried to sync changes from a collaborator by hand?
- Copied an entire source tree to try things a different way?
- **NOT tried something because it might break things?**
- Re-downloaded a GitHub project when the author made changes?
- Wanted to compare a file with any past version of itself?
- Wished you could share a fixed set of files with your research paper, enabling reproducibility, without preventing further development?
- Wondered when in the last N revisions something stopped working?

Why Should I Care?

- **Source Control:** versions, diffs, branches – experiment with confidence
- **Backups:** entire history available locally, easy remote sync
- **Sharing:** build a portfolio of work available to the public
- **Collaborating:** work in one shared codebase
- the world of **Open Source** is built on Git
- **Because Professionals Do…**



Over 90% of respondents use Git, suggesting that it is a fundamental tool to being a developer.



Stack Overflow 2021 Developer Survey, Other Tools (N=76,253)
https://insights.stackoverflow.com/survey/2021#experience-years-code-country

Primary References

- MIT - The Missing Semester of Your CS Education, Version Control
  - https://missing.csail.mit.edu/2020/version-control/
- Chacon, S., Straub, B. (2014). Pro Git. United Kingdom: Apress.
  - Free download: https://git-scm.com/book/en/v2

Git `is`

Snapshot-based[2] distributed[3] version control system[1] for source files[4]

1. "system that records changes to a file or set of files over time so that you can recall specific versions later"; stored in a database, *repository*
2. "takes a picture of what all your files look like at that moment and stores a reference to that snapshot" – not *delta-based*
3. every repository is a complete mirror – not client-server model
4. text files used to make things: code, Markdown, LaTeX, etc., not by-products (binaries, PDFs, etc.)

1, 2 quotes: Git Pro, Ch 1

Local, Redundant, Human-Readable → Fast, Secure, Portable, "Future-Proof"

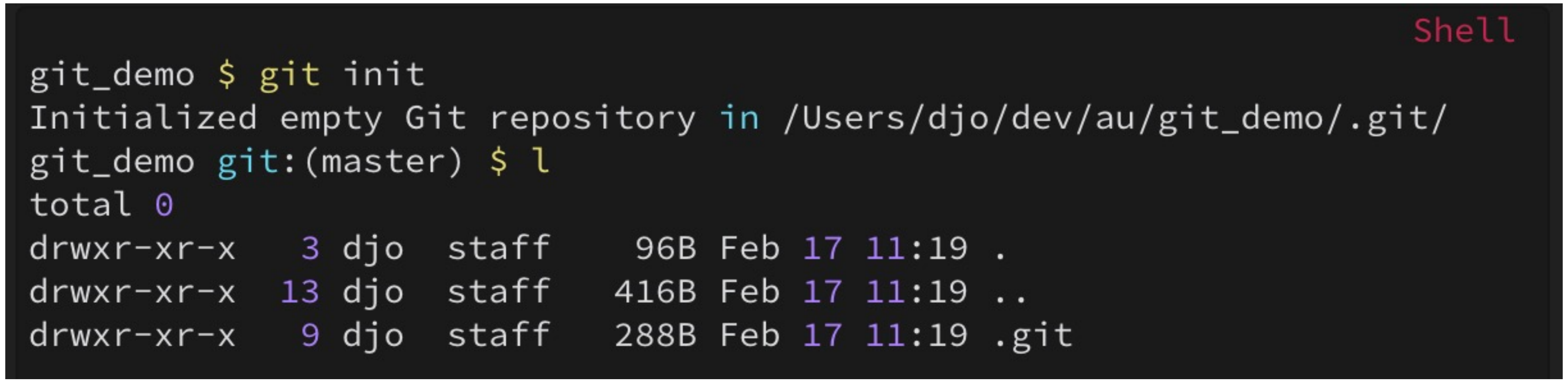
Git `is not`

- Git is not GitHub
- Related, but NOT the same
- Git – local repository
- GitHub – website for remote hosting of GIT repositories
- Git is essential for source control, local "backups"
- GitHub (or others like it) is powerful option for sharing, collaboration
- Git is the original product, others are derivatives

How to Git?

- GUIs – e.g. GitHub Desktop and others (dozens) for all platforms
- Shell Integration – show Git status at command prompt
- Editor Integration – Git in your IDE (VS Code, PyCharm, RStudio, etc.)

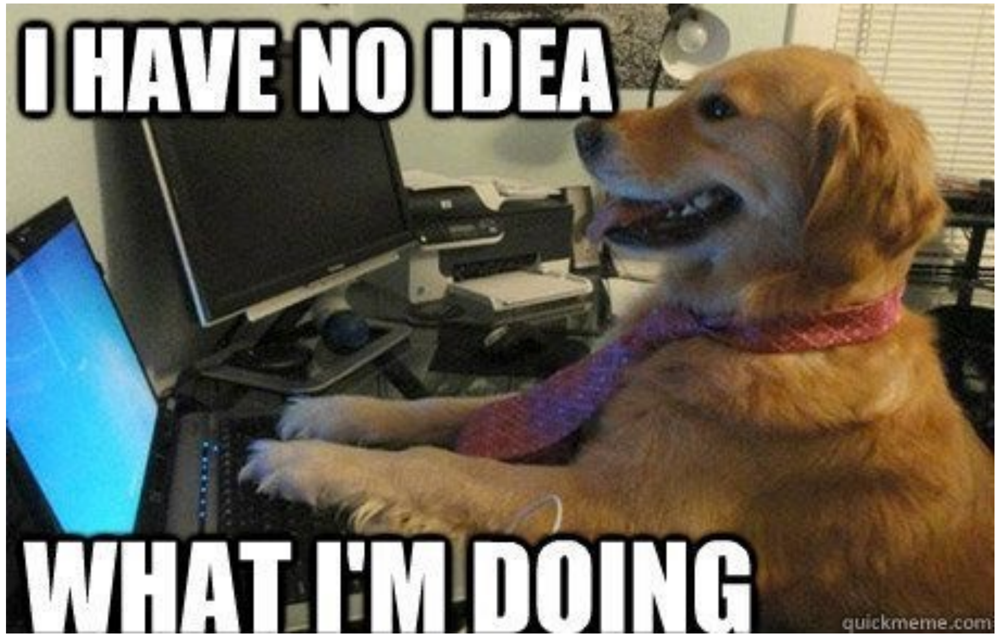
Today I will focus on Git's Command Line Interface



```
                                                                  Shell
git_demo $ git init
Initialized empty Git repository in /Users/djo/dev/au/git_demo/.git/
git_demo git:(master) $ l
total 0
drwxr-xr-x   3 djo  staff   96B Feb 17 11:19 .
drwxr-xr-x  13 djo  staff  416B Feb 17 11:19 ..
drwxr-xr-x   9 djo  staff  288B Feb 17 11:19 .git
```

Command Line
aka “shell”

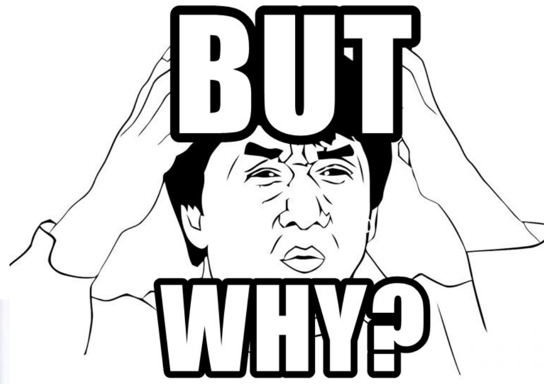
```
C:\>
```



- Terminal or iTerm (macOS), Command Prompt or PowerShell (Win)
- Good introductions:
  - https://missing.csail.mit.edu/2020/course-shell/
  - https://swcarpentry.github.io/shell-novice/

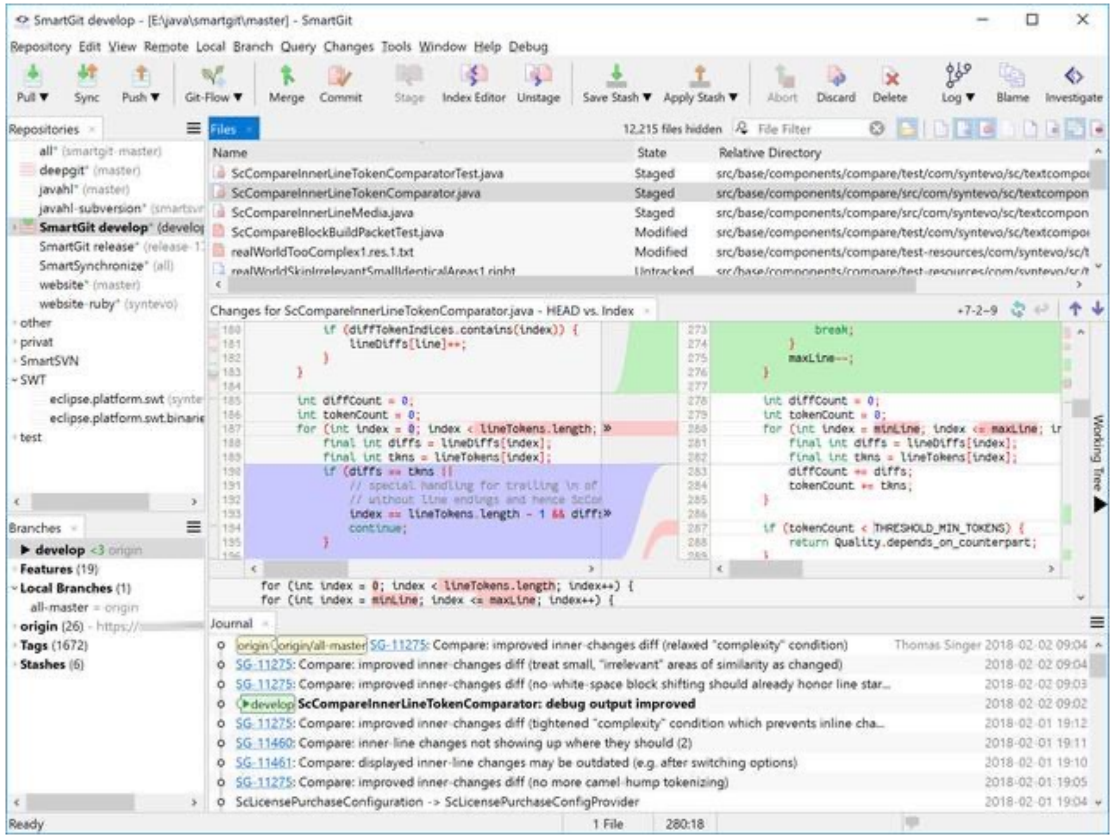
Command Line: Why?!

- Supports all Git commands
- Is explicit – you (can) know exactly what you are doing
- If you can learn the CLI version, you can learn the GUI version; reverse not necessarily true
- GUI choice personal; all users have CLI
- CLI can be automated, etc., etc.
- **Can focus on actions, less (different) interface overhead**



Actual image of Dan teaching grad students to use the command line, because it's the way he learned to use computers in the 80s.

GUIs – Sophisticated or Simplified



Name	State	Relative Directory
ScCompareInnerLineTokenComparatorTest.java	Staged	src/base/components/compare/test/com/syntevo/sc/textcompo
ScCompareInnerLineTokenComparator.java	Staged	src/base/components/compare/src/com/syntevo/sc/textcompon
ScCompareInnerLineMedia.java	Staged	src/base/components/compare/src/com/syntevo/sc/textcompon
ScCompareBlockBuildPacketTest.java	Modified	src/base/components/compare/test/com/syntevo/sc/textcompo
realWorldTooComplex1.res.1.txt	Modified	src/base/components/compare/test-resources/com/syntevo/sc/t
realWorldSkinIrrelevantSmallIdenticalAreas1.right	Untracked	src/base/components/compare/test-resources/com/syntevo/sc/t

Changes for ScCompareInnerLineTokenComparator.java - HEAD vs. Index +7-2-9

```
        if (diffTokenIndices.contains(index)) {
            lineDiffs[line]++;
        }
    }

    int diffCount = 0;
    int tokenCount = 0;
    for (int index = 0; index < lineTokens.length; »
        final int diffs = lineDiffs[index];
        final int tkns = lineTokens[index];
        if (diffs == tkns ||
            // special handling for trailing \n of
            // without line endings and hence ScCo
            index == lineTokens.length - 1 && diff:»
            continue;
        }
```

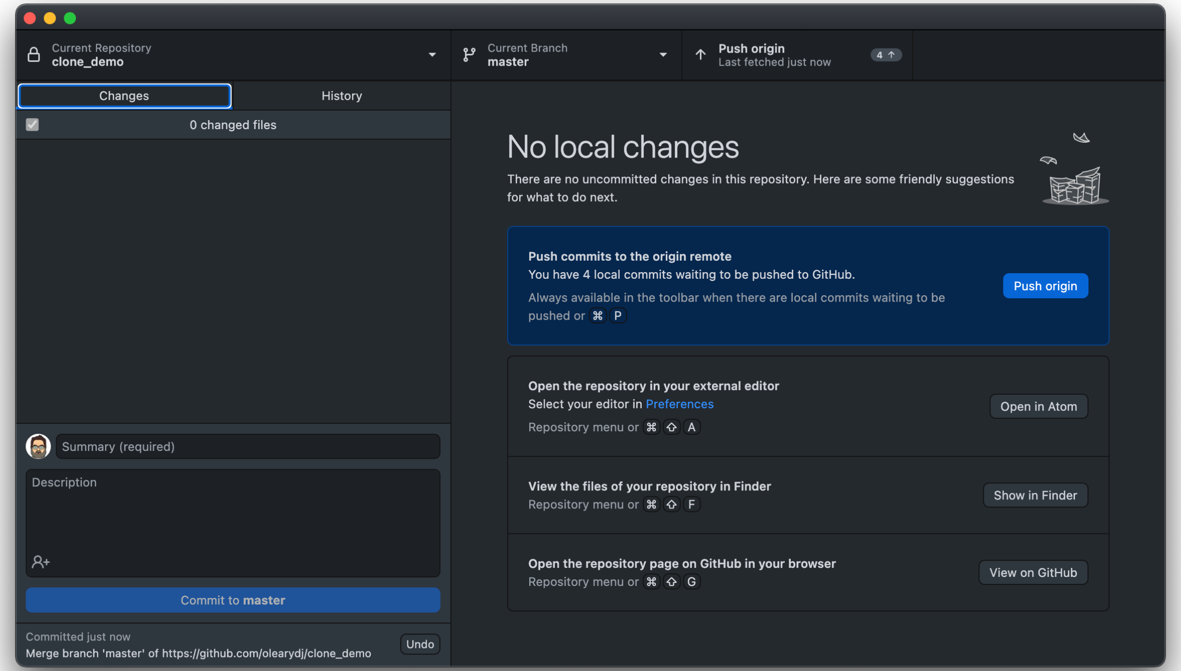
```
            break;
        }
        maxLine--;
    }

    int diffCount = 0;
    int tokenCount = 0;
    for (int index = minLine; index <= maxLine; ir
        final int diffs = lineDiffs[index];
        final int tkns = lineTokens[index];
        diffCount += diffs;
        tokenCount += tkns;
    }

    if (tokenCount < THRESHOLD_MIN_TOKENS) {
        return Quality.depends_on_counterpart;
```

```
for (int index = 0; index < lineTokens.length; index++) {
for (int index = minLine; index <= maxLine; index++) {
```

Journal

```
o origin origin/all-master SG-11275: Compare: improved inner-changes diff (relaxed "complexity" condition)   Thomas Singer 2018-02-02 09:04
o SG-11275: Compare: improved inner-changes diff (treat small, "irrelevant" areas of similarity as changed)   2018-02-02 09:04
o SG-11275: Compare: improved inner-changes diff (no white-space block shifting should already honor line star...   2018-02-02 09:03
o develop ScCompareInnerLineTokenComparator: debug output improved   2018-02-02 09:02
o SG-11275: Compare: improved inner-changes diff (tightened "complexity" condition which prevents inline cha...   2018-02-01 19:12
o SG-11460: Compare: inner-line changes not showing up where they should (2)   2018-02-01 19:11
o SG-11461: Compare: displayed inner-line changes may be outdated (e.g. after switching options)   2018-02-01 19:10
o SG-11275: Compare: improved inner-changes diff (no more camel-hump tokenizing)   2018-02-01 19:05
o ScLicensePurchaseConfiguration -> ScLicensePurchaseConfigProvider   2018-02-01 19:04
```

Full-featured tools like SmartGit (pictured), Fork, Tower, etc.



Committed just now
Merge branch 'master' of https://github.com/olearydj/clone_demo

No local changes

There are no uncommitted changes in this repository. Here are some friendly suggestions for what to do next.

Push commits to the origin remote
You have 4 local commits waiting to be pushed to GitHub.
Always available in the toolbar when there are local commits waiting to be pushed or ⌘ P

Open the repository in your external editor
Select your editor in Preferences
Repository menu or ⌘ ⇧ A

View the files of your repository in Finder
Repository menu or ⌘ ⇧ F

Open the repository page on GitHub in your browser
Repository menu or ⌘ ⇧ G

More limited, workflow / use case oriented tools like GitHub Desktop (pictured)

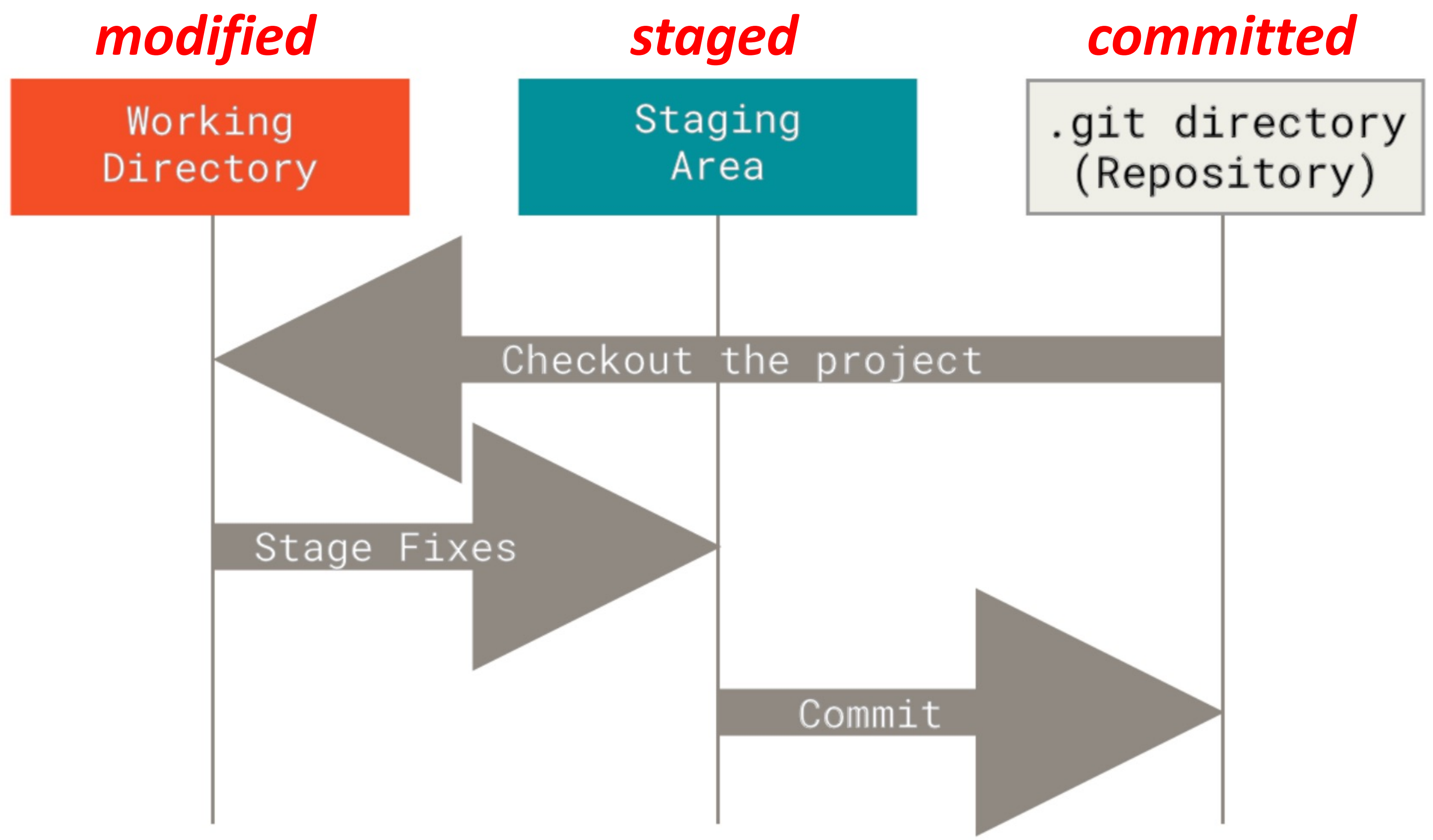
Basic Workflow and Terminology

Workflow:

1. Initialize a Git repository (adds folder to existing project)
2. Add / modify files in the project tree
3. Decide which files to put under version control
4. Selectively stage changes when you want to include them
5. Commit staged changes to the repository

Corresponding file states and project sections:

- File states: *modified, tracked, staged, committed*
- Git project sections: *working directory, staging area, repository*

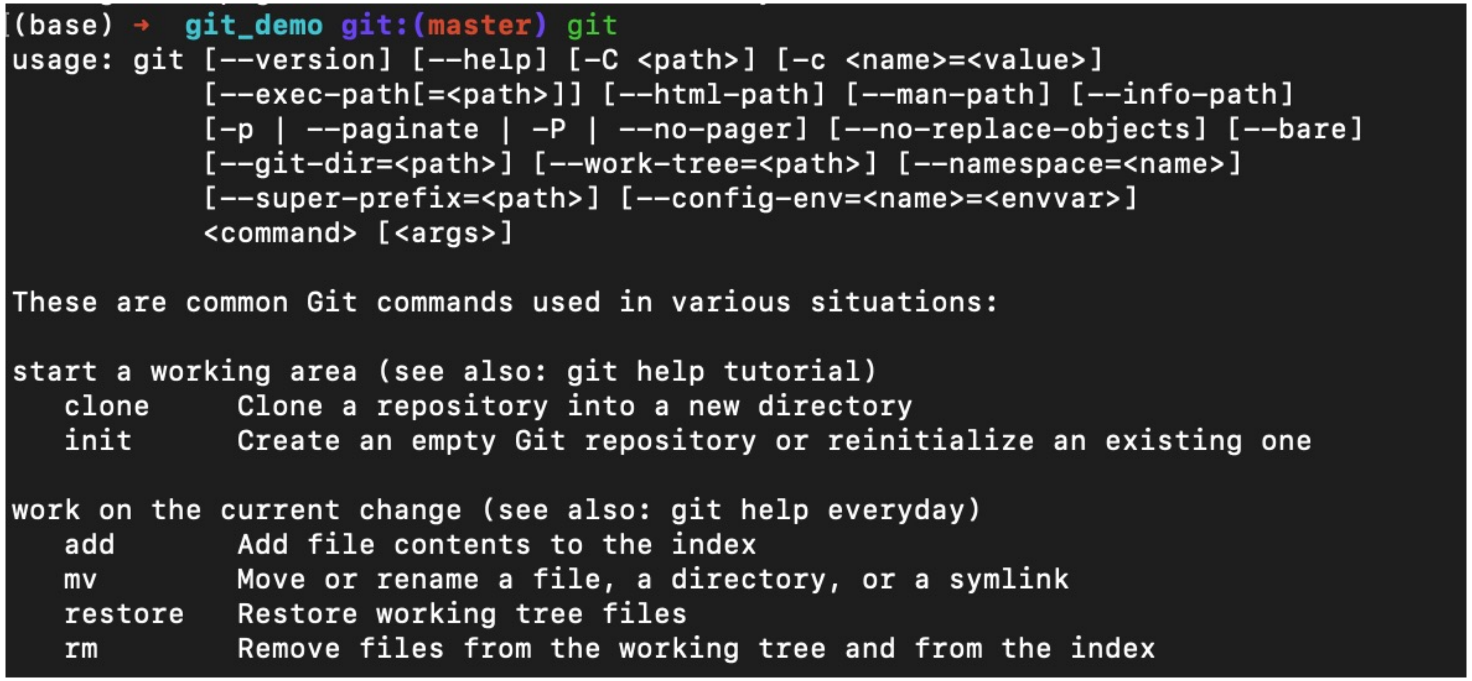


- Staging area is a file: index of tracked files w. info about next commit
- The `.git` directory is a hidden subdirectory of the root project folder, contains the Git database and related metadata

Git things going… Installation

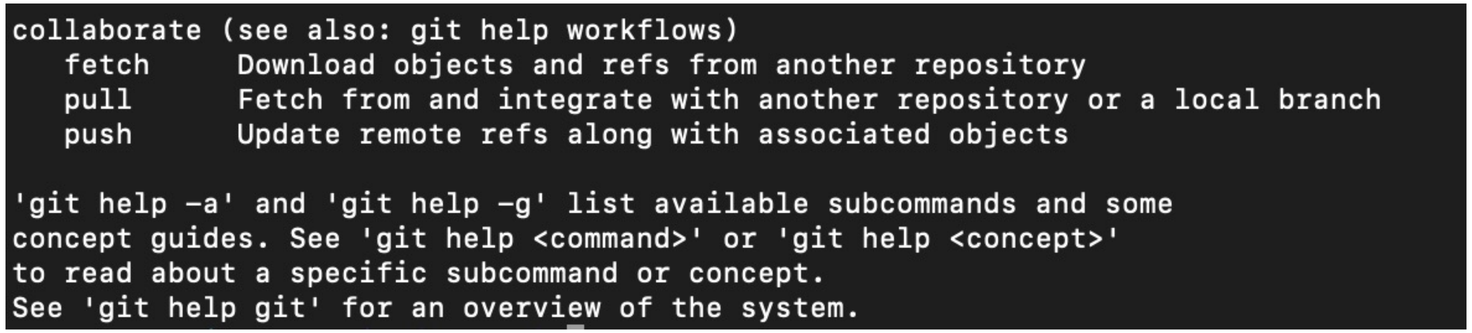
- See Pro Git, p.16 for instructions
- Pro-Tip: learn about package managers like Chocolatey (Win) ❓🍫 or Homebrew (macOS / *nix) ❤️🍺
  - command-line tools for automating the installation / update of other system tools, including git and many, many others
  - for example, to install git using Homebrew: `brew install git`

Command Line Syntax: type 'git'

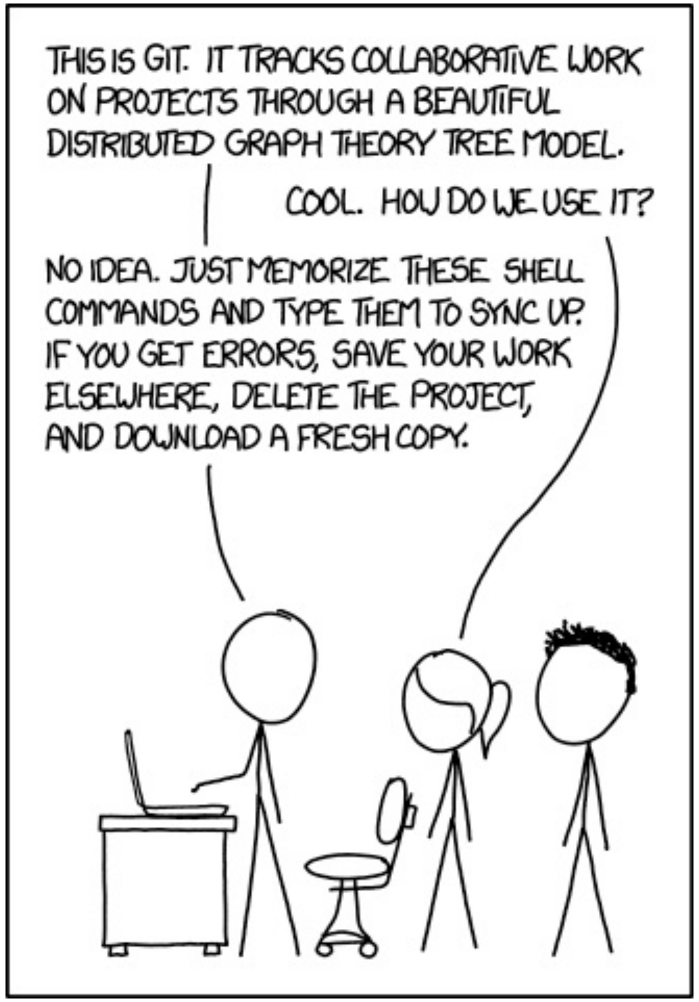


```
(base) ➜  git_demo git:(master) git
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           [--super-prefix=<path>] [--config-env=<name>=<envvar>]
           <command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
   clone     Clone a repository into a new directory
   init      Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
   add       Add file contents to the index
   mv        Move or rename a file, a directory, or a symlink
   restore   Restore working tree files
   rm        Remove files from the working tree and from the index
```

…



```
collaborate (see also: git help workflows)
   fetch     Download objects and refs from another repository
   pull      Fetch from and integrate with another repository or a local branch
   push      Update remote refs along with associated objects

'git help -a' and 'git help -g' list available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
See 'git help git' for an overview of the system.
```


THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

comic: xkcd

hold up…
when **is lunch?!**

Git Command Line Syntax

- Simplified form: `git [<flags>] <command> [<args>]`
- We will focus on a few key commands that give major benefits
  - *A few minutes to learn, a lifetime to master…*

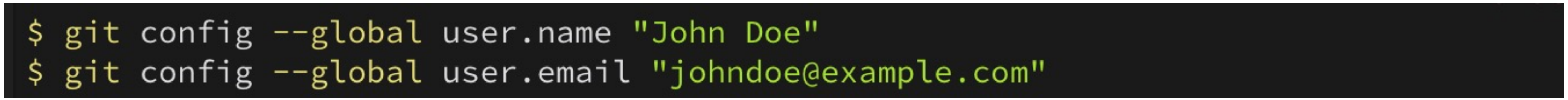
I will demonstrate, using only about 10 commands:

1. Intro to git commands: initial configuration with `git config`
2. Working with a local repo: `init, status, add, commit`
3. Cloning remote repos: `clone, pull / fetch`
4. Working with your own remote repos using GitHub CLI, `push`

1. Intro Git Commands

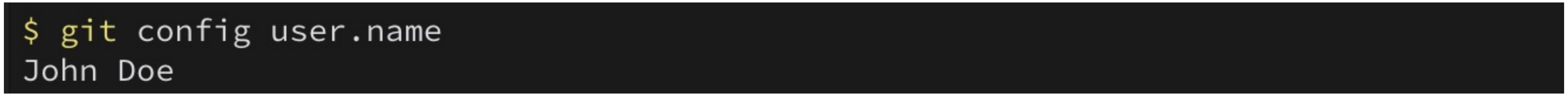
- First thing after installation – configuration
  - Git is absurdly configurable, managed by `config` command
  - Only one essential change: name and email, baked into commits

command option arguments

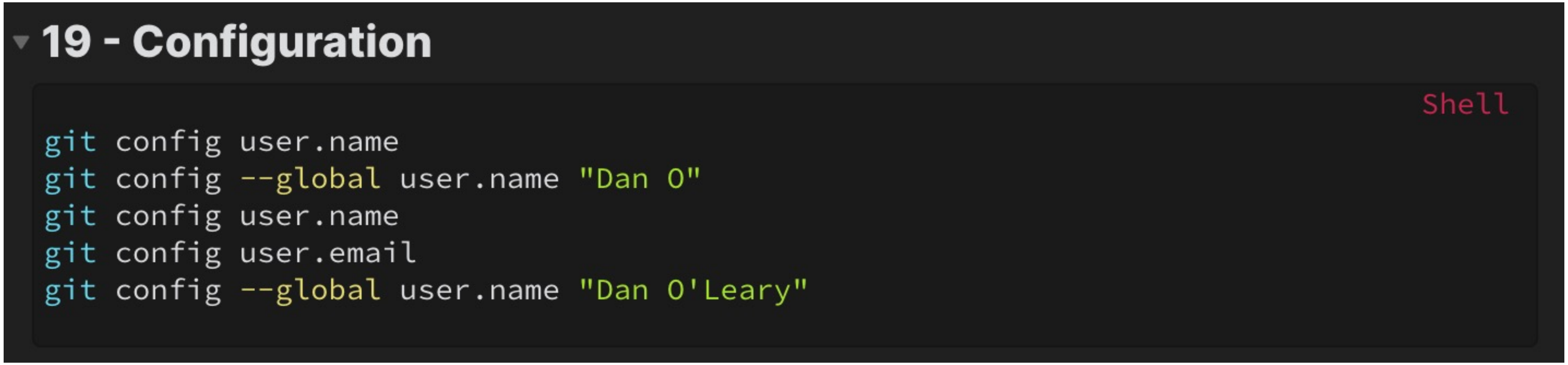


```
$ git config --global user.name "John Doe"
$ git config --global user.email "johndoe@example.com"
```

- Enter the command as shown above
  - Do not include the $ (shell prompt), use your own details
- Check the results:



```
$ git config user.name
John Doe
```



19 - Configuration

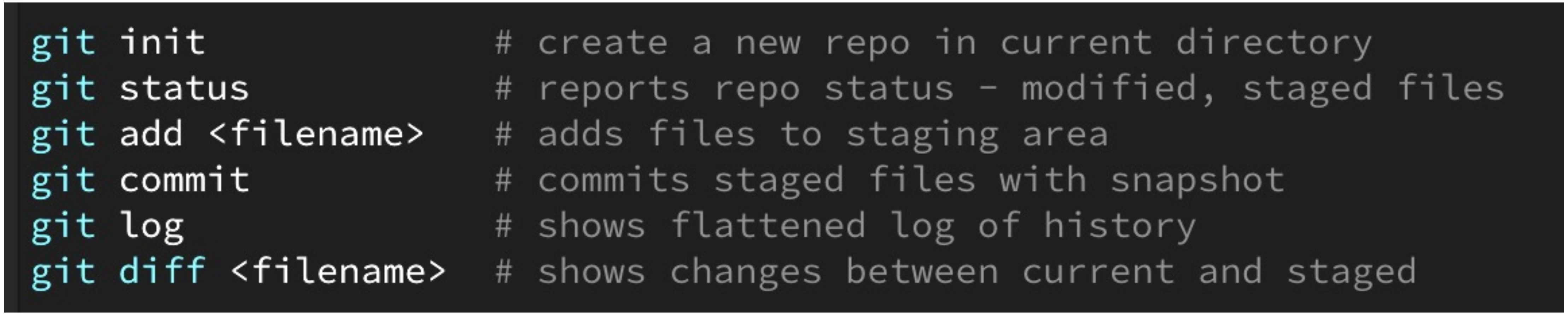
Shell

```
git config user.name
git config --global user.name "Dan O"
git config user.name
git config user.email
git config --global user.name "Dan O'Leary"
```

Gitting Help

- For comprehensive help on a topic: `git help <command>`
  - e.g., `git help config`
  - uses "man page" (manual) interface: scroll, page, / for find, q quit
- For more concise help: `git <command> –h`
  - e.g., `git config –h`
- Learning to read / interpret these is a skill in itself, but the result applies to all command line tools
- Google search results may also be confusing, out of date, etc., especially at first; best as support for primary / secondary references

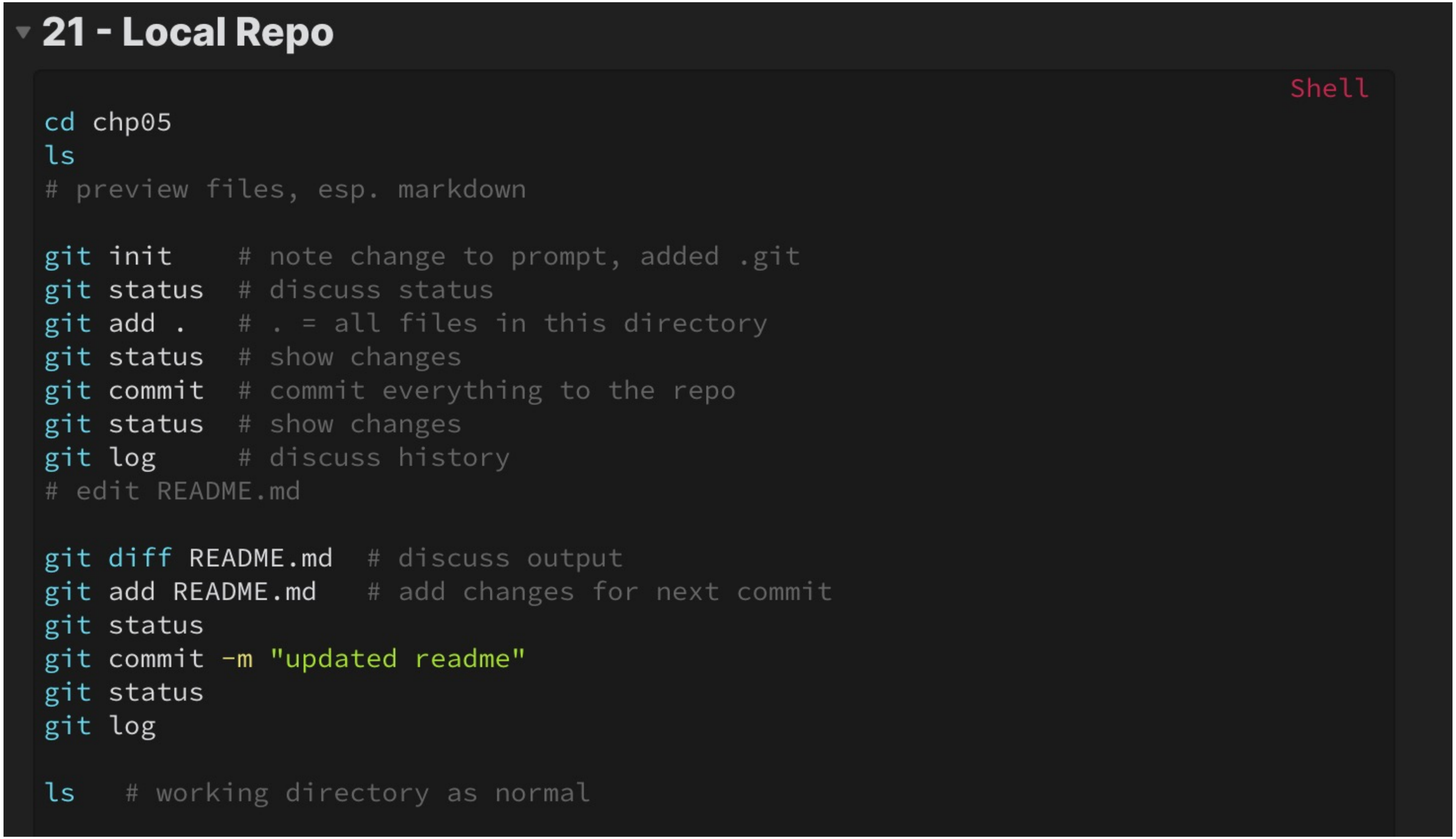
2. Work with a Local Repo



```
git init                # create a new repo in current directory
git status              # reports repo status - modified, staged files
git add <filename>      # adds files to staging area
git commit              # commits staged files with snapshot
git log                 # shows flattened log of history
git diff <filename>     # shows changes between current and staged
```

Goal: Use git for local version control

- Initialize a project
- Check the status
- Add files to staging area
- Commit changes
- Look at log and diffs



21 - Local Repo

```
cd chp05
ls
# preview files, esp. markdown

git init    # note change to prompt, added .git
git status  # discuss status
git add .   # . = all files in this directory
git status  # show changes
git commit  # commit everything to the repo
git status  # show changes
git log     # discuss history
# edit README.md

git diff README.md  # discuss output
git add README.md   # add changes for next commit
git status
git commit -m "updated readme"
git status
git log

ls   # working directory as normal
```

What’s the worst that could happen?

- With these commands – confusion
- Experiment with small test projects first
- Need a clean start? – delete the .git folder and re-initialize repo
  - lose history, working copy unchanged
- Git is almost always additive
  - Hard to lose data, by design (meant to be a record of changes)
  - More “dangerous” commands are highlighted in Git Pro
- Can do a lot with just what we’ve covered so far, learn more as you go

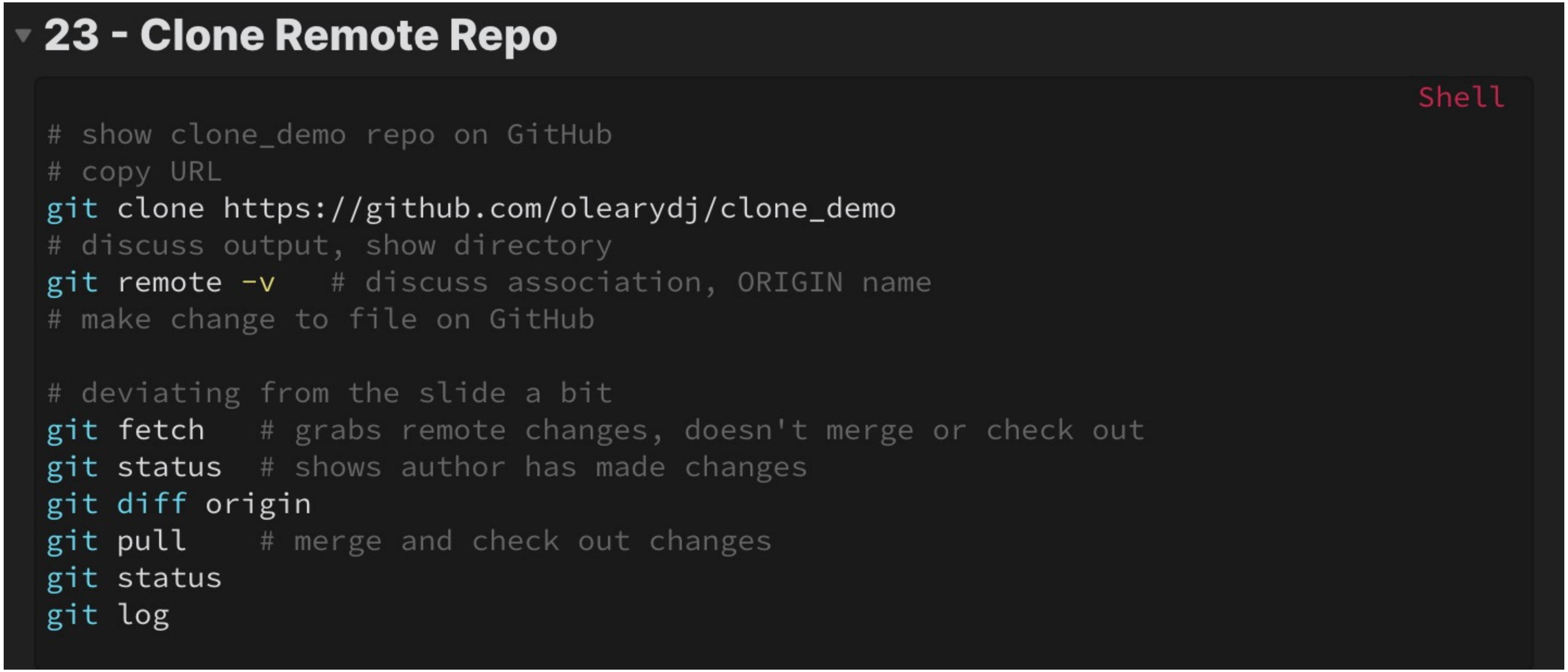
3. Clone a Remote Repo

- Typical Use Case: found a book, course, or other repo you want to work from, e.g.
  - HOML 2: https://github.com/ageron/handson-ml2
  - INSY6500: https://github.com/ausim/InfoTechForOps
  - Any repo on GitHub or other git “remotes” (web, server, local)
- Goal: Create a local copy of those files, able to easily update
  - Want to keep up with author’s changes, will accept them “as is”
  - Limited local changes, mostly in new files
  - No plan to share changes with the author

Git Clone + Pull

```
git clone <url>         # creates local copy of repo
git remote -v           # show linked remote repos
git pull                # get changes and merge
```

- e.g.: `git clone https://github.com/olearydj/clone_demo`
    - Creates directory `clone_demo` and initializes `.git` directory inside
    - Pulls down all data for that repository
    - Checks out a working copy of the latest version
- Time to update? `git pull`
    - gets updates on the remote repo
    - tries to merge them with the current branch
- Demo deviates / expands on this with `fetch` and `diff`



23 - Clone Remote Repo

```
# show clone_demo repo on GitHub
# copy URL
git clone https://github.com/olearydj/clone_demo
# discuss output, show directory
git remote -v   # discuss association, ORIGIN name
# make change to file on GitHub

# deviating from the slide a bit
git fetch   # grabs remote changes, doesn't merge or check out
git status  # shows author has made changes
git diff origin
git pull    # merge and check out changes
git status
git log
```

What’s the worst that could happen?

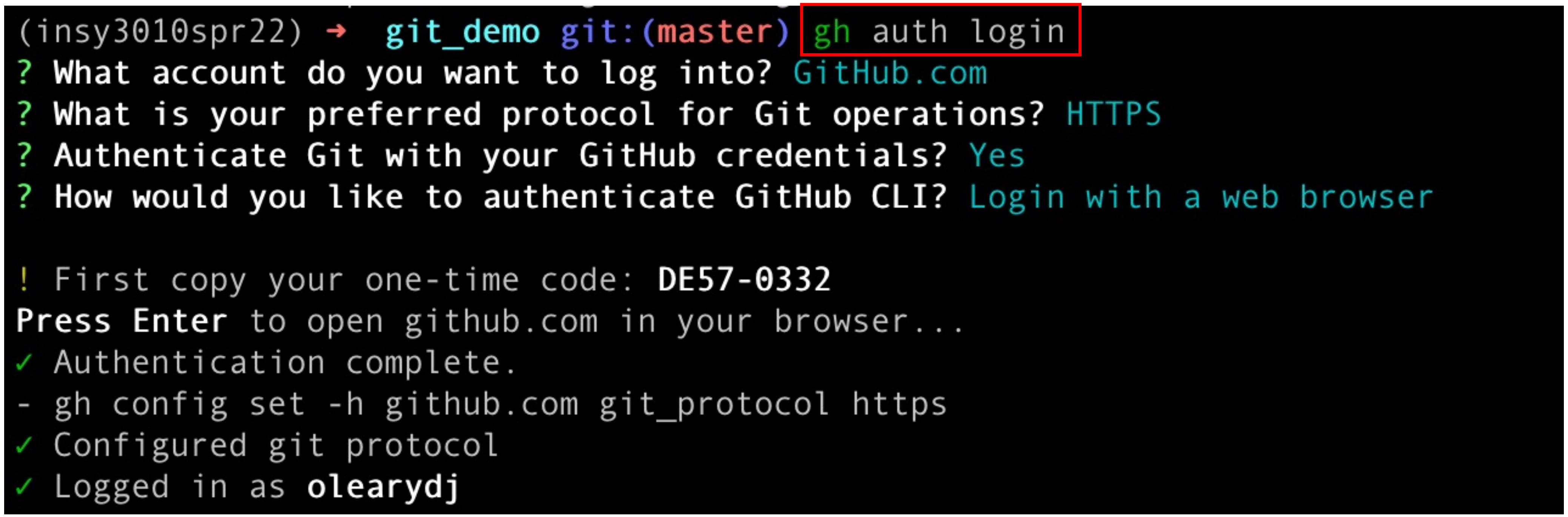
- Adding your own files – `add`, `commit` without worries (optional)
- Editing author’s files can create conflicts, gets sticky!
  - you change a file, then the author changes it
  - pull → merge changes, which to keep: yours, authors, both, how?
- Either don’t modify cloned files or
  - Learn to merge and
  - Use `git fetch` and merge selectively
- Simple, zero risk option – keep your notes in a separate folder (with its own repo); treat clone as a reference only
- PS: clone → all history, can be large: use `--depth=1` for shallow clone

4. Work with Your Own Remote Repo

- Share publicly or as a private copy for backup and/or synchronization
- Goal: Create a remote copy of a local repo, sync changes
  - Primarily interested in pushing local changes to the remote
  - May also want to pull down changes from remote
- For this example we'll use GitHub
  - Other hosts exist (e.g GitLab, BitBucket)
  - Remotes can also be on local server or even another local copy
- To manipulate GitHub from the command line you need another tool

GitHub Setup Process

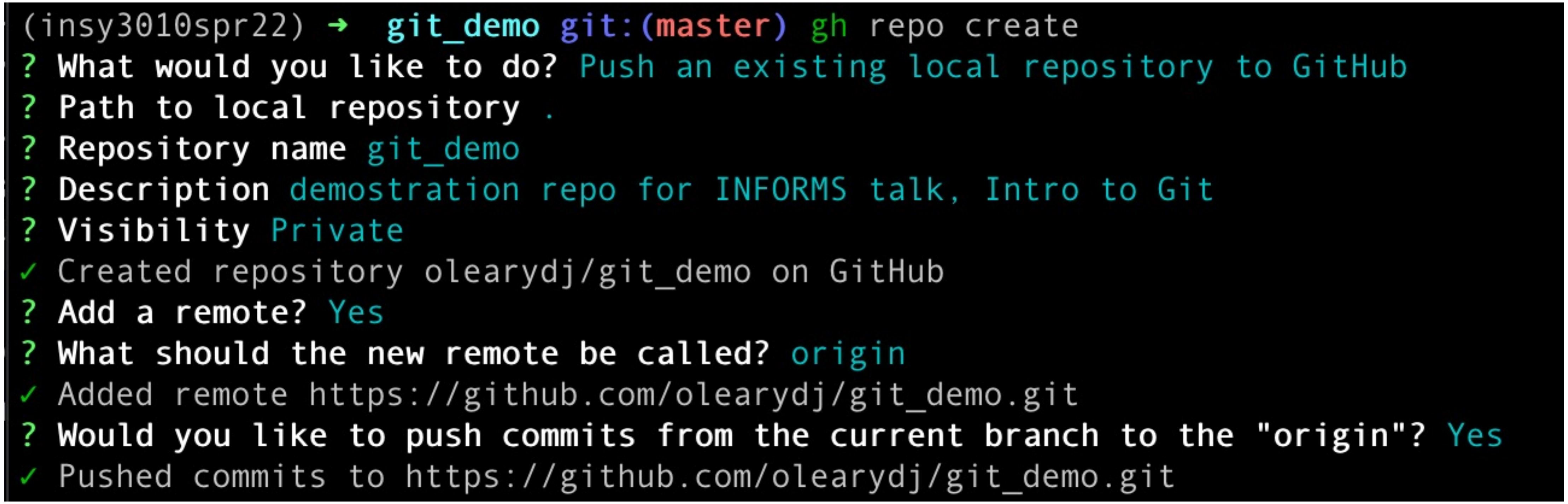
1. Create GitHub account and log in
2. Get the GitHub CLI tool
3. Authenticate with a GitHub host:
   `gh auth login` and follow prompts as below (defaults)



```
(insy3010spr22) ➜  git_demo git:(master) gh auth login
? What account do you want to log into? GitHub.com
? What is your preferred protocol for Git operations? HTTPS
? Authenticate Git with your GitHub credentials? Yes
? How would you like to authenticate GitHub CLI? Login with a web browser

! First copy your one-time code: DE57-0332
Press Enter to open github.com in your browser...
✓ Authentication complete.
- gh config set -h github.com git_protocol https
✓ Configured git protocol
✓ Logged in as olearydj
```

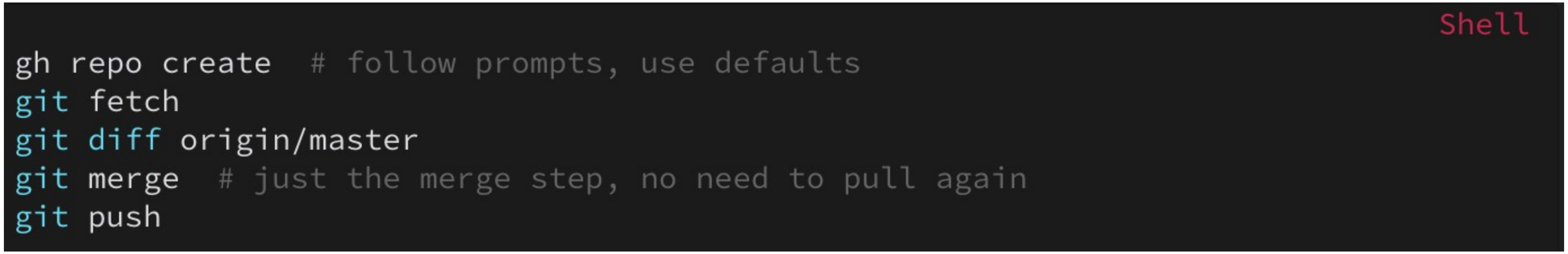
Clone Local Repo to GitHub

- In the local repo: `gh repo create`
  - choose ‘Push existing local repo…’ and follow prompts
  - Details: Add existing project to GitHub using the command line



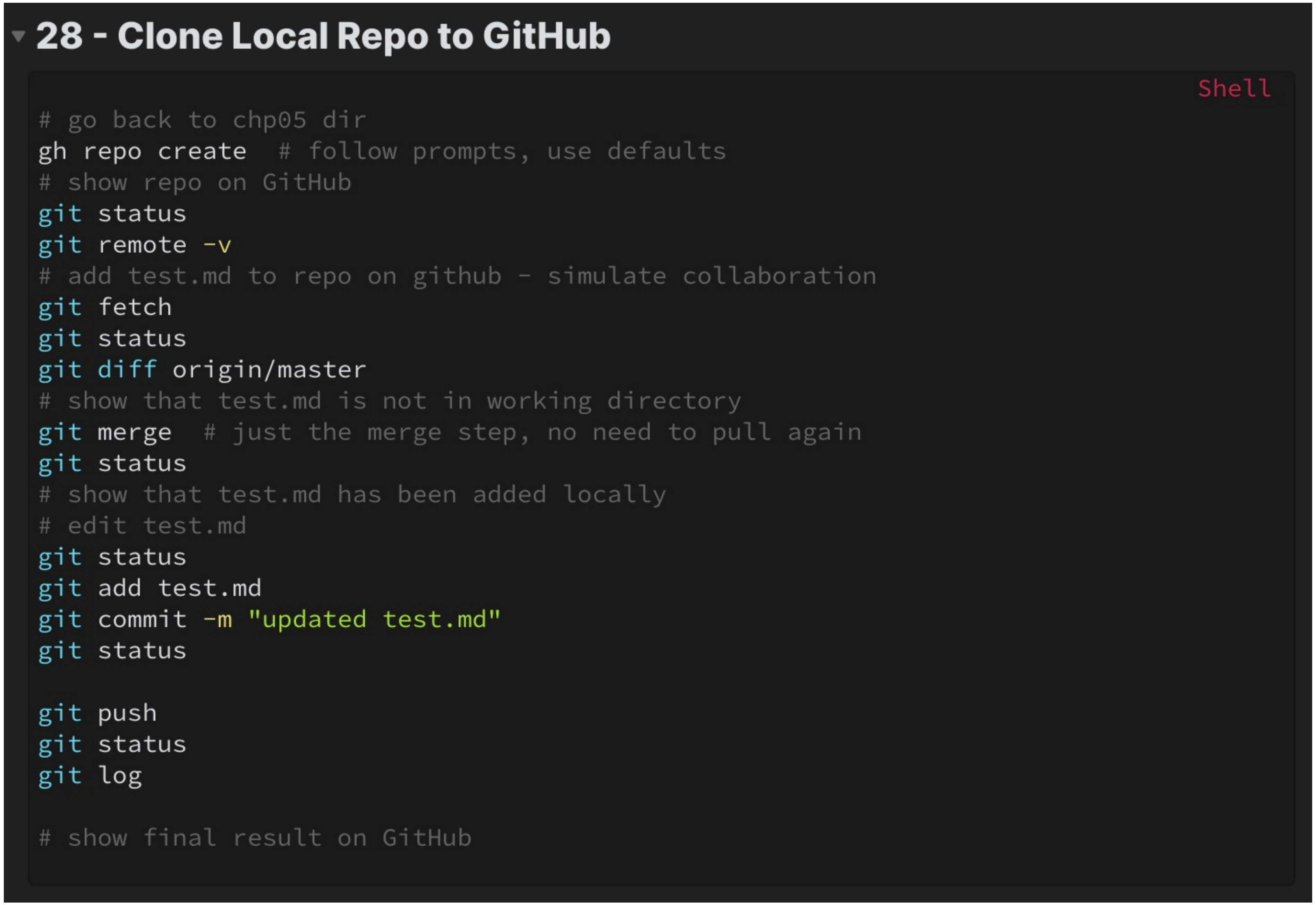
```
(insy3010spr22) ➜  git_demo git:(master) gh repo create
? What would you like to do? Push an existing local repository to GitHub
? Path to local repository .
? Repository name git_demo
? Description demostration repo for INFORMS talk, Intro to Git
? Visibility Private
✓ Created repository olearydj/git_demo on GitHub
? Add a remote? Yes
? What should the new remote be called? origin
✓ Added remote https://github.com/olearydj/git_demo.git
? Would you like to push commits from the current branch to the "origin"? Yes
✓ Pushed commits to https://github.com/olearydj/git_demo.git
```

Work with Your Own Remote Repo



```
                                                                            Shell
gh repo create  # follow prompts, use defaults
git fetch
git diff origin/master
git merge  # just the merge step, no need to pull again
git push
```

- Same methods + `push`
- Create or clone a remote repo
- Commit local changes and push them to the shared remote
- Work from multiple computers, keep them synchronized
- Fetch / pull updates from the remote to local
- Opens the door to collaboration…



28 - Clone Local Repo to GitHub

Shell

```
# go back to chp05 dir
gh repo create  # follow prompts, use defaults
# show repo on GitHub
git status
git remote -v
# add test.md to repo on github - simulate collaboration
git fetch
git status
git diff origin/master
# show that test.md is not in working directory
git merge  # just the merge step, no need to pull again
git status
# show that test.md has been added locally
# edit test.md
git status
git add test.md
git commit -m "updated test.md"
git status

git push
git status
git log

# show final result on GitHub
```

Collaboration: Small to Medium Teams

Devs clone the repo,
make changes,
push them,
pull changes from others,
merge as necessary

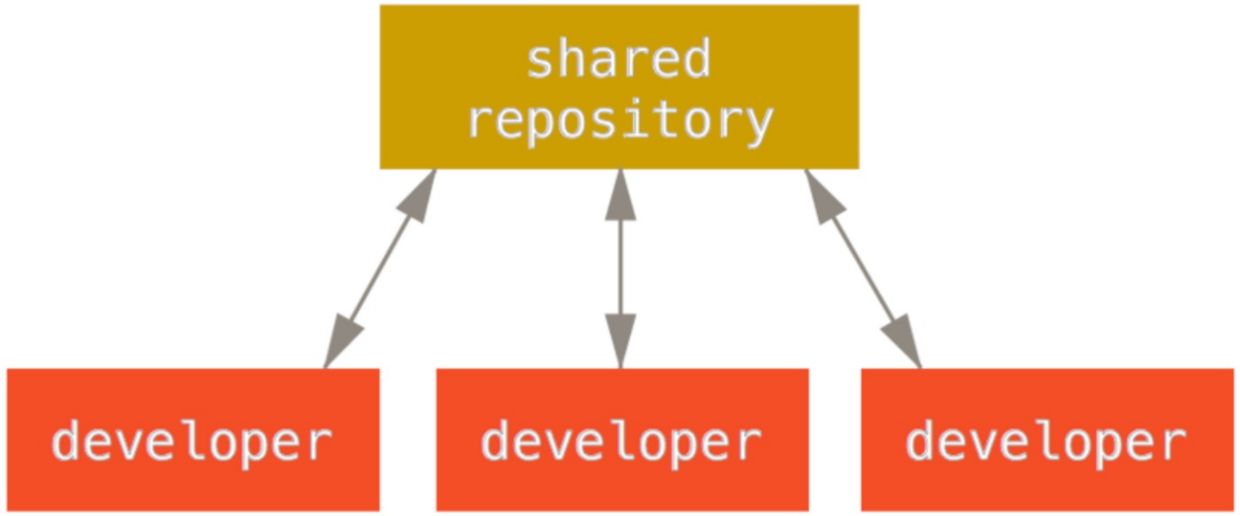


Figure 53. Centralized workflow

Merges can be tricky – managed by tools, mitigated by practices

Collaboration Practices

- Strategies to minimize overlap – don’t put everything in one file and all work on it at the same time!
- Break work into many files, by functional groups (UI, save, load, etc.)
- Have team members specialize in certain functional areas
- Use branches, and have a release strategy:
    - https://www.endoflineblog.com/oneflow-a-git-branching-model-and-workflow

Other Considerations

- Git is designed for text-based file formats (code, markup languages, csv, etc.) – refrain from using it with others, esp. large or binary files
  - Use `.gitignore`, put those files in cloud, etc. (secure, backups)
  - Data version control tools exist, critical complement for ML, etc.
- NEVER include sensitive information (e.g. passwords, API/SSH/AWS keys, CC numbers, PIN numbers, etc.) in files you push to a public repository
  - Use configuration files, environment variables, or more sophisticated methods to secure this data outside the repo
- Write good commit messages

Did Not Cover…

- Merging, branching, tagging, rebasing, etc.
- Tip of the iceburg, but…
  - A little can go a long way
  - Hopefully enough for you to justify the investment in learning more

Next Steps: Git Some!

- Use it regularly – "perishable skill"
- Continue with the CLI, set up Git integration in your IDE, and/or supplement with simple GUI (GitHub Desktop) – see how CLI commands translate
- Migrate to more text-based workflows, e.g. using Markdown, Notebooks, LaTeX, etc. for notes (Obsidian), publishing (RStudio), etc.
- Learn more software carpentry skills and techniques!
  - command line tools, virtual environments, package managers, unit testing, regex, debugging, make…
- BUILD REPRODUCABILITY INTO YOUR METHODS!

with virtual environments and package managers and automated unit tests and continuous integration and…

Additional Resources

Broader Topic: Software Carpentry, “Lab skills for research computing”

- MIT - The Missing Semester of Your CS Education, Version Control
  - https://missing.csail.mit.edu/2020/version-control/
- The Carpentries
  - https://software-carpentry.org/
  - https://carpentries.org/
- Also: Huff, K. D., Scopatz, A. (2015). Effective Computation in Physics: Field Guide to Research with Python. United States: O'Reilly Media.