Creating repos

INTRODUCTION TO GIT

George Boorman
Curriculum Manager, DataCamp

Why make a repo?

Benefits of repos

- Systematically track versions
- Collaborate with colleagues
- Git stores everything!

[1] Image credit: https://unsplash.com/@jasongoodman_youxventures

Creating a new repo

```
git init mental-health-workspace
```

```
cd mental-health-workspace
```

```
git status
```

```
On branch main

No commits yet

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

Converting a project

```
git init
```

```
Initialized empty Git repository in /home/repl/mental-health-workspace/.git/
```

What is being tracked?

```
git status
```

```
On branch main

No commits yet

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

        data/
        report.md

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

Nested repositories

- Don't create a Git repo inside another Git repo
  - Known as nested repos

- There will be two `.git` directories
- Which `.git` directory should be updated?
- Not necessary in most circumstances

Let's practice!

INTRODUCTION TO GIT

Working with remotes

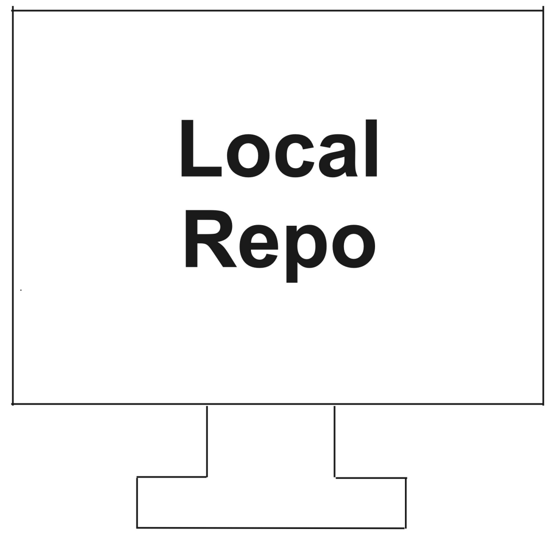
INTRODUCTION TO GIT

George Boorman
Curriculum Manager, DataCamp

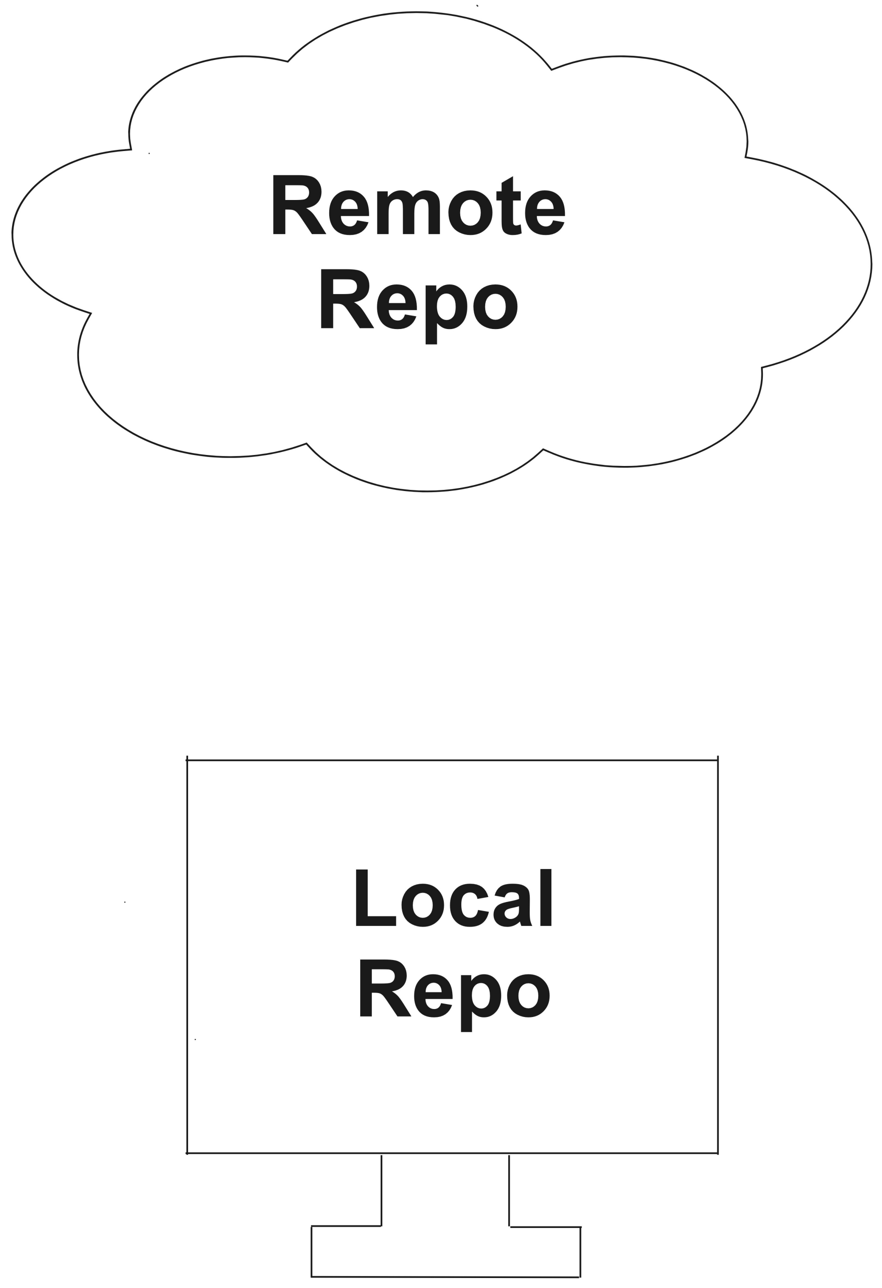
What is a remote?

[1] Image credit: https://unsplash.com/@glenncarstenspeters

Local repo



Remote repo



Why use remote repos?

Benefits of remote repos

- Everything is backed up
- Collaboration, regardless of location

[1] Image credit: https://unsplash.com/@mahlkornel

Cloning locally

```
git clone path-to-project-directory
```

```
git clone /home/john/repo
```

```
git clone /home/john/repo new_repo
```

Cloning a remote

- Remote repos are stored in an online hosting service e.g., GitHub, Bitbucket, or Gitlab
- If we have an account:
  - we can clone a remote repo on to our local computer

```
git clone [URL]
```

```
git clone https://github.com/datacamp/project
```

Identifying a remote

- When cloning a repo
  - Git remembers where the original was
- Git stores a remote **tag** in the new repo's configuration

```
git remote
```

```
datacamp
```

Getting more information

```
git remote -v
```

```
datacamp     https://github.com/datacamp/project (fetch)
datacamp     https://github.com/datacamp/project (pull)
```

Creating a remote

- When cloning, Git will automatically name the remote `origin`

```
git remote add name URL
```

```
git remote add george https://github.com/george_datacamp/repo
```

- Defining remote names is useful for merging branches

Let's practice!

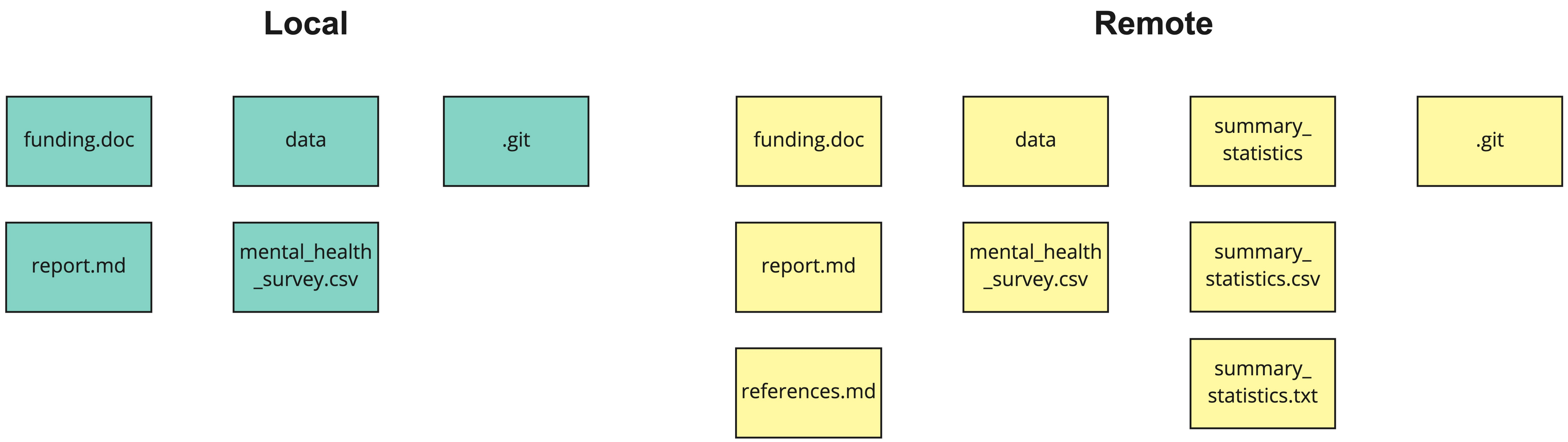
INTRODUCTION TO GIT

Gathering from a remote

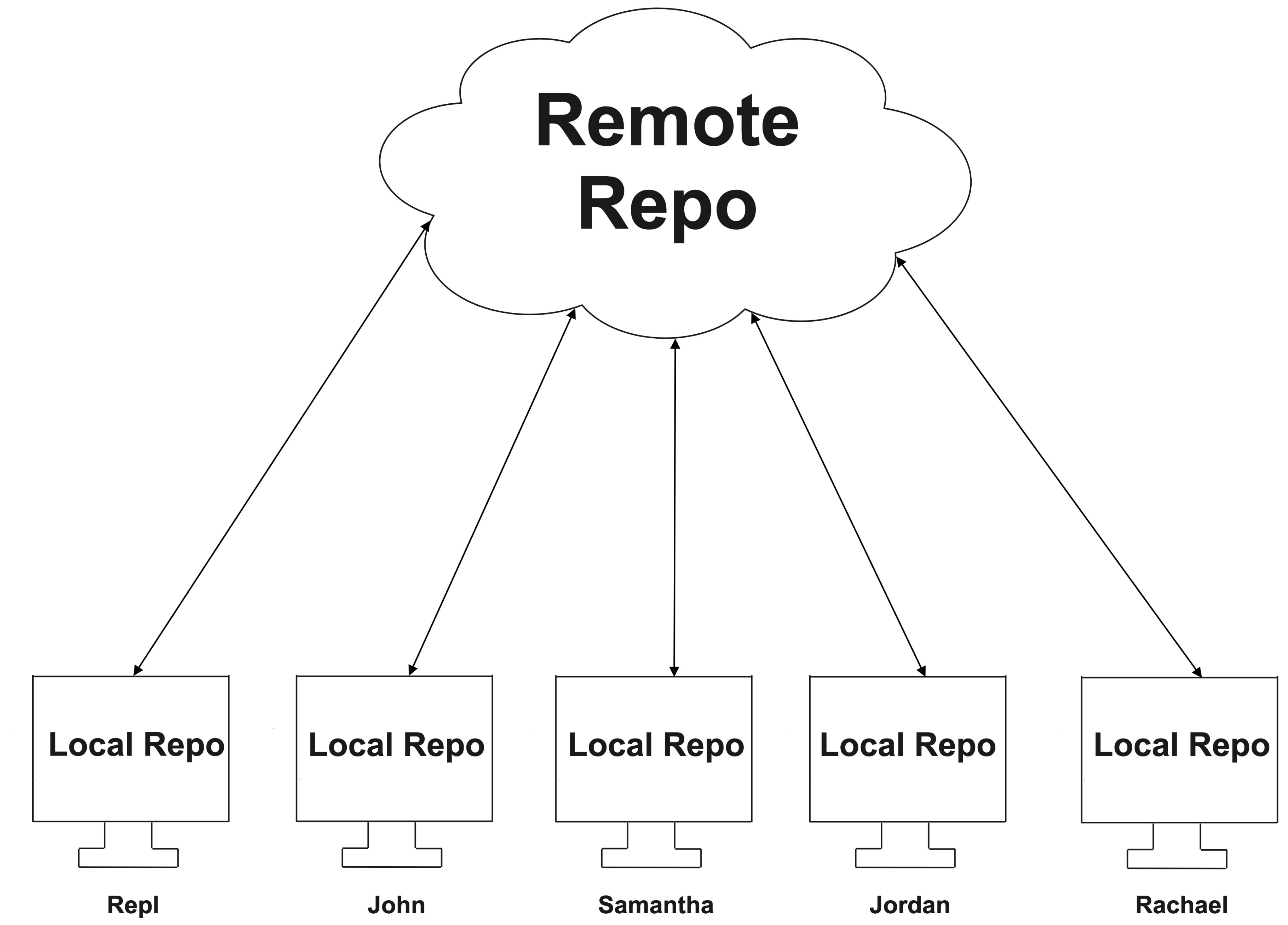
INTRODUCTION TO GIT

George Boorman
Curriculum Manager, DataCamp

Remote vs. local



Collaborating on Git projects



Fetching from a remote

```
git fetch origin main
```

```
From https://github.com/datacamp/project
 * branch                main     -> FETCH_HEAD
```

Fetching from a remote

```
git fetch origin report
```

Synchronizing content

```
git merge origin main
```

```
Updating 9dcf4e5..887da2d
Fast-forward
 data/mental_health_survey.csv | 3 +++
 report.md                     | 1 +
 2 files changed, 4 insertions (+)
```

Pulling from a remote

- `remote` is often ahead of `local` repos
- `fetch` and `merge` is a common workflow
- Git simplifies this process for us!

```
git pull origin main
```

Pulling from a remote

```
From https://github.com/datacamp/project
 * branch                main      -> FETCH_HEAD
Updating 9dcf4e5..887da2d
Fast-forward
 data/mental_health_survey.csv | 3 +++
 report.md                     | 1 +
 2 files changed, 4 insertions (+)
```

Pulling with unsaved local changes

```
git pull origin
```

```
Updating 9dcf4e5..887da2d
error: Your local changes to the following files would be overwritten by merge:
        report.md
Please commit your changes or stash them before you merge.
Aborting
```

- Important to save locally before pulling from a `remote`

Let's practice!

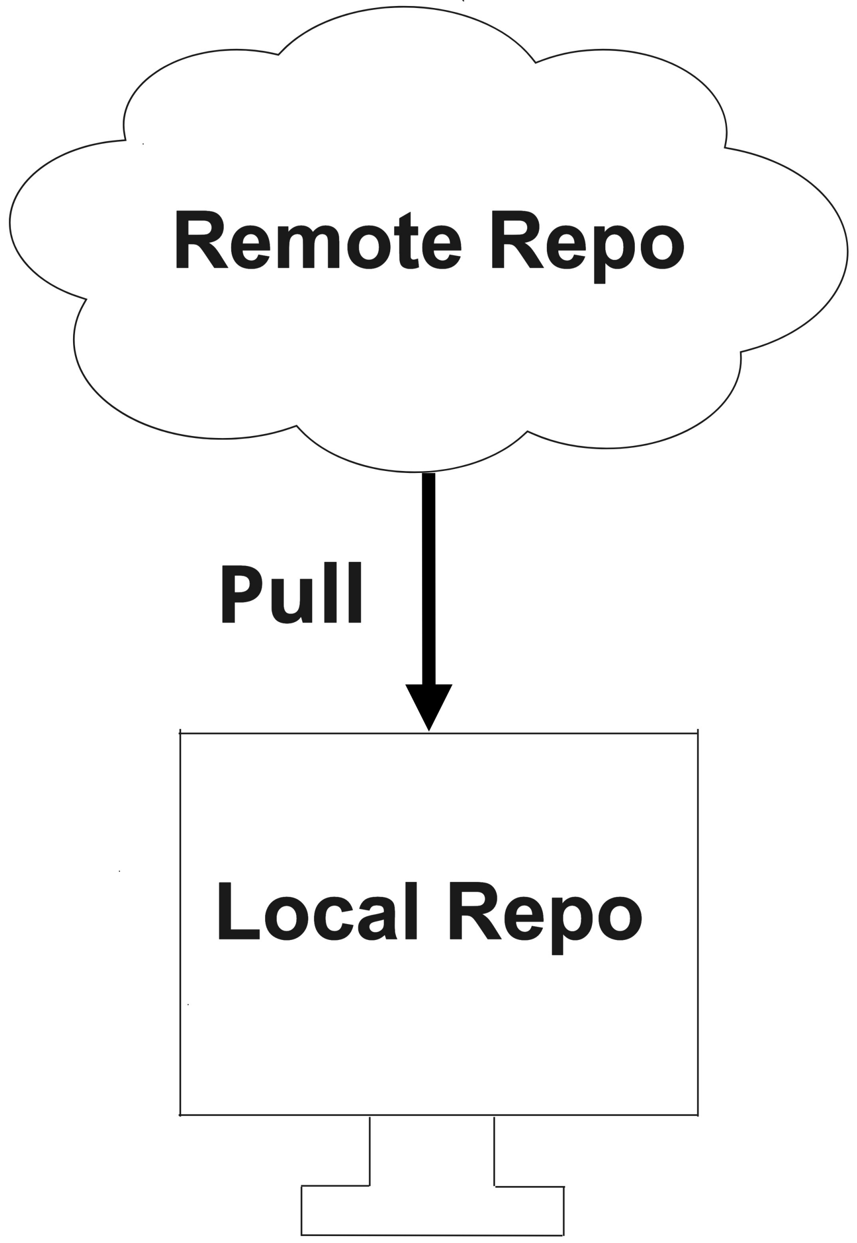
INTRODUCTION TO GIT

Pushing to a remote

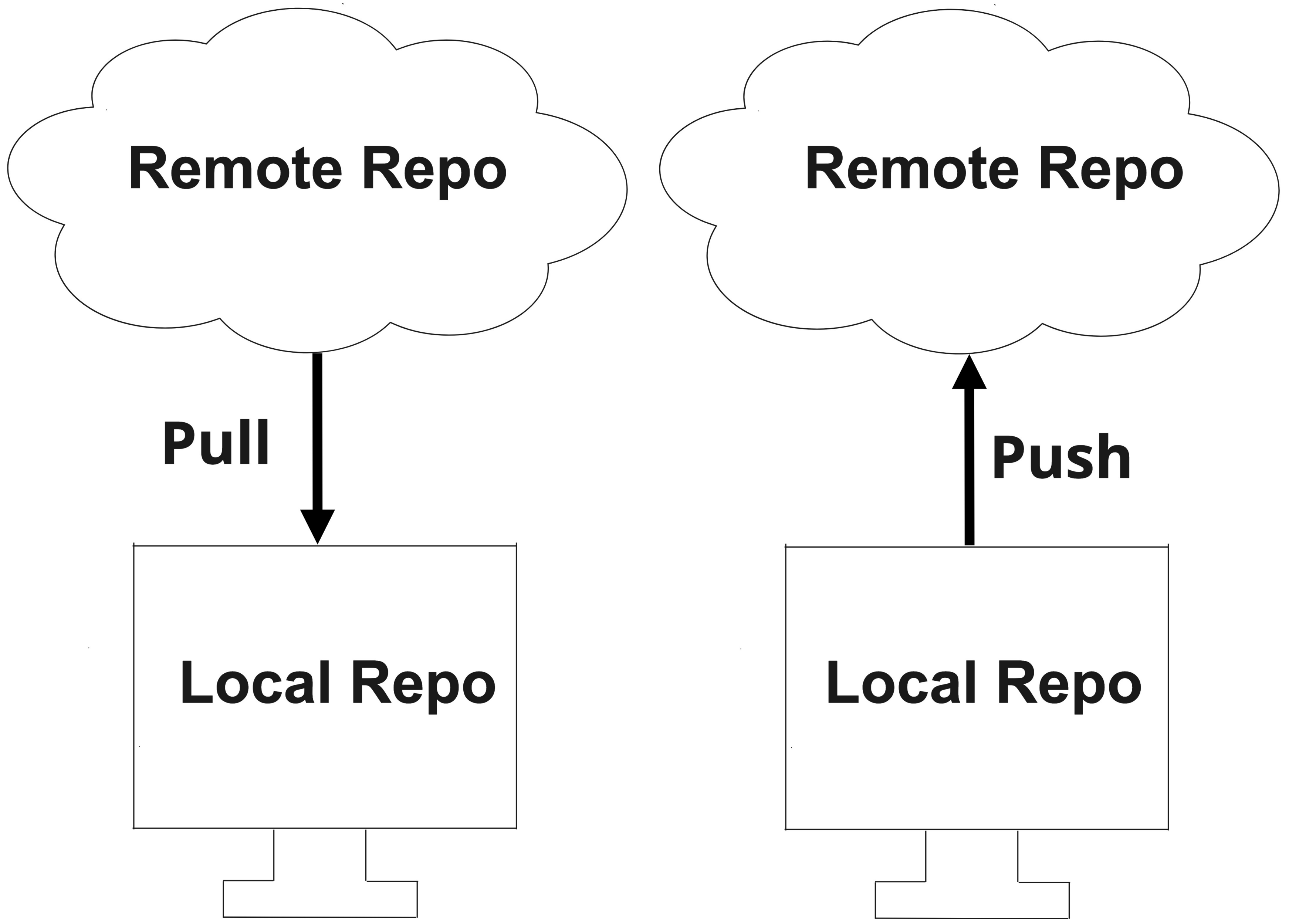
INTRODUCTION TO GIT

George Boorman
Curriculum Manager, DataCamp

Pulling from a remote



Pushing to a remote



git push

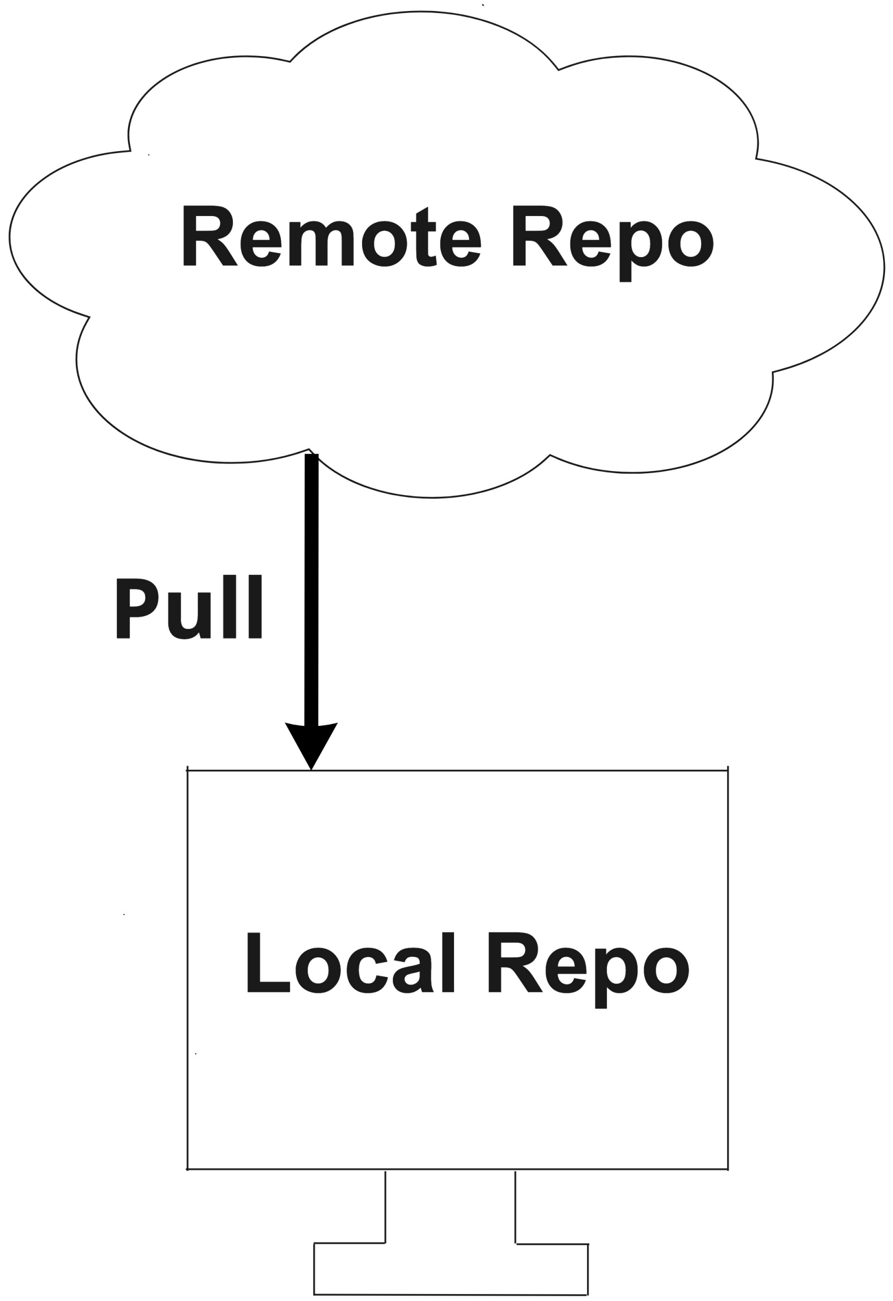
- Save changes locally first!

```
git push remote local_branch
```

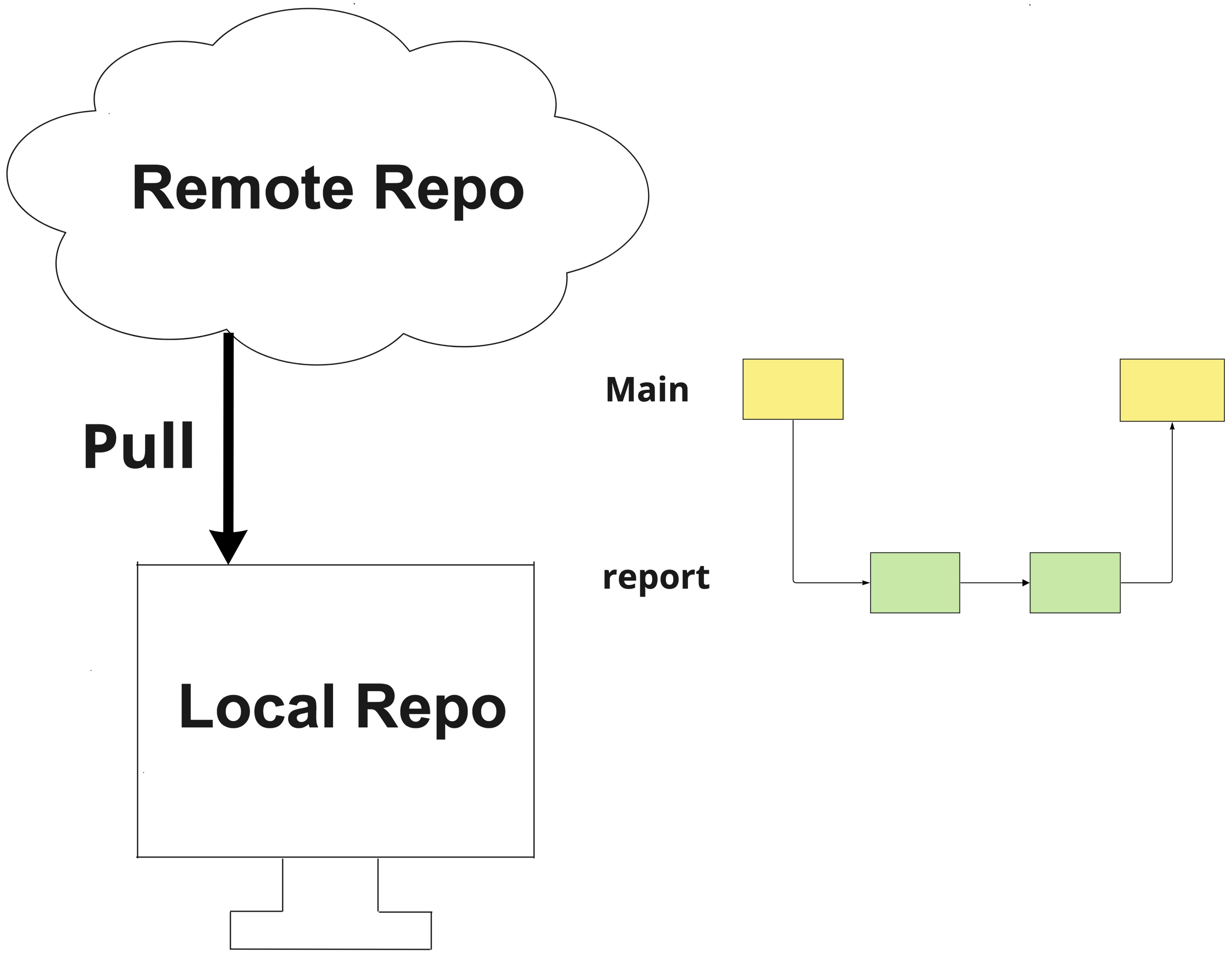
- Push *into* `remote` **from** `local_branch`

```
git push origin main
```

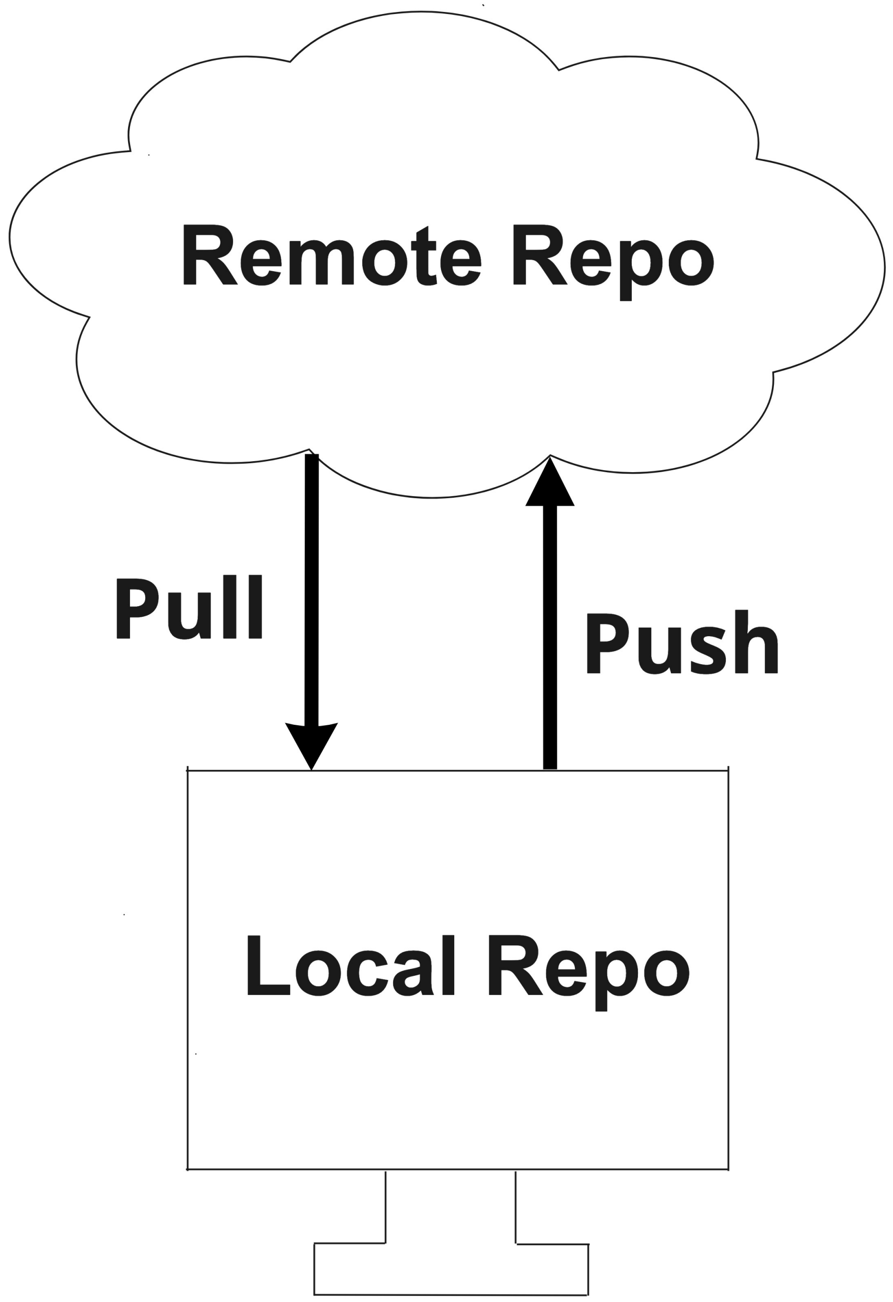
Push/pull workflow

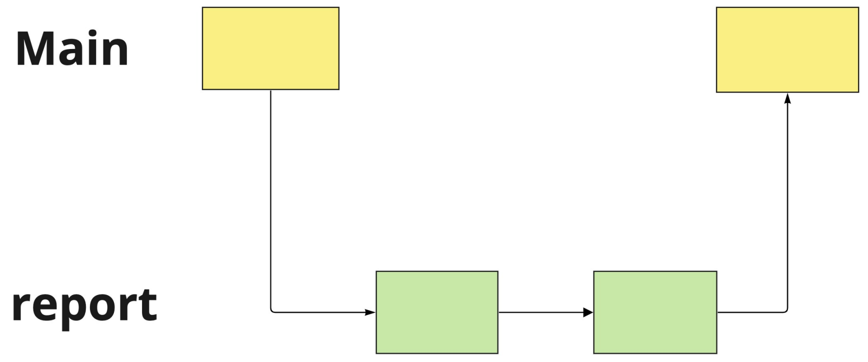


Push/pull workflow



Push/pull workflow

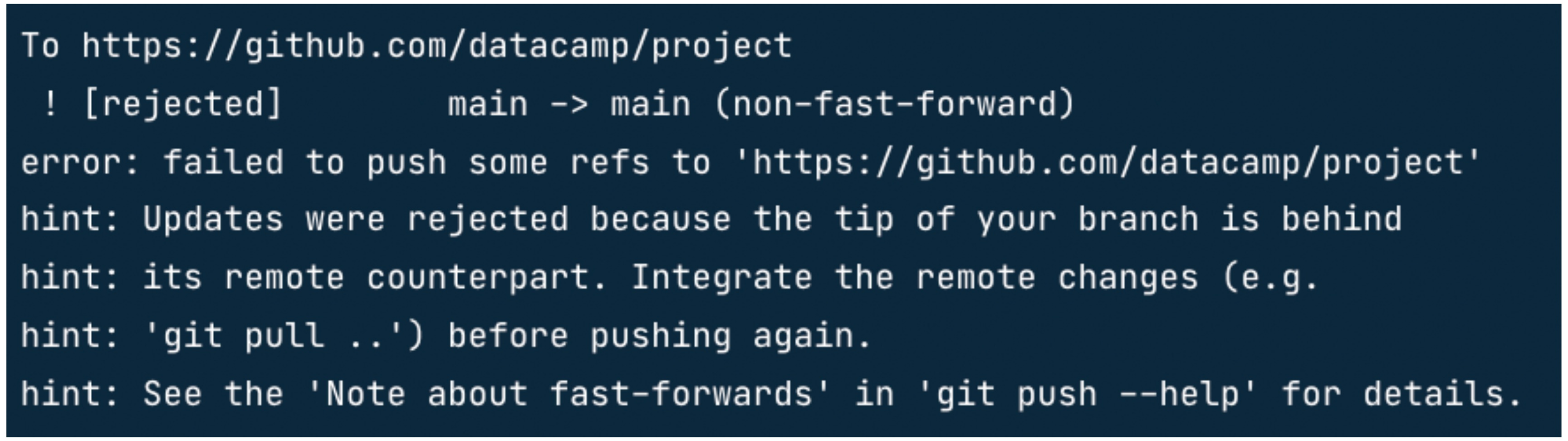




Pushing first

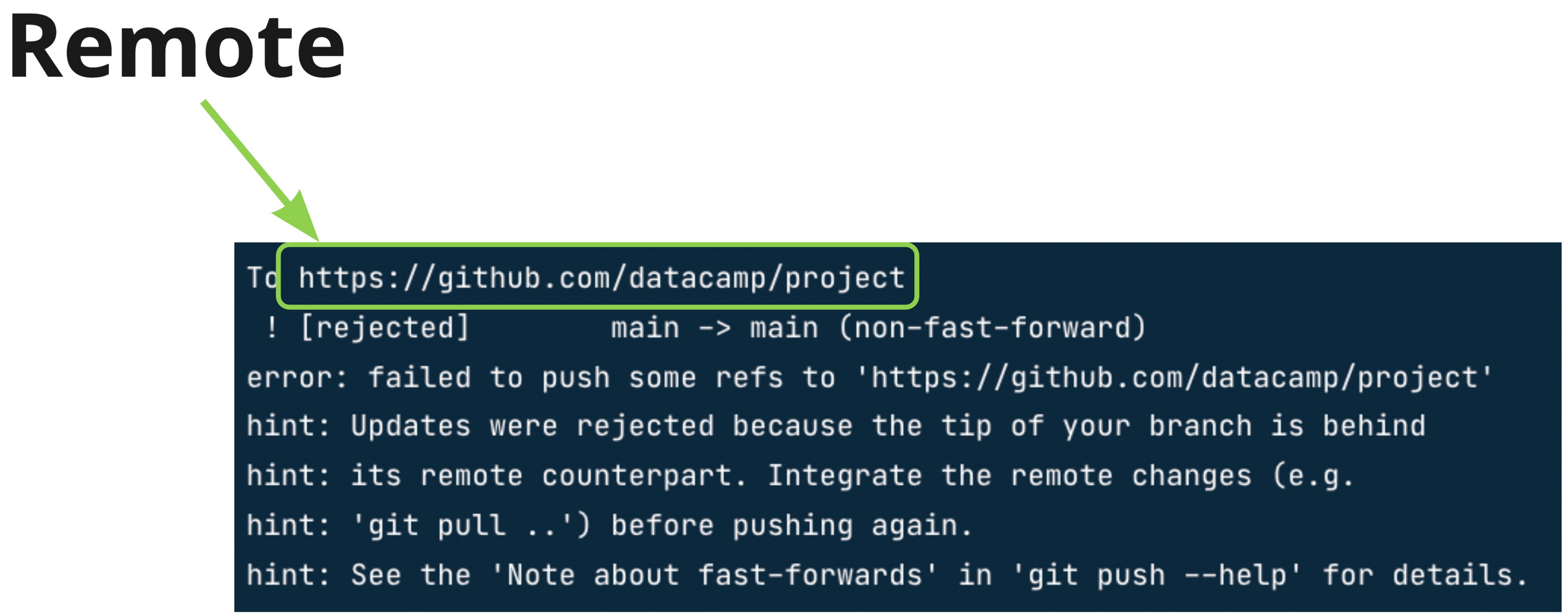
```
git push origin main
```

Remote/local conflicts



```
To https://github.com/datacamp/project
 ! [rejected]        main -> main (non-fast-forward)
error: failed to push some refs to 'https://github.com/datacamp/project'
hint: Updates were rejected because the tip of your branch is behind
hint: its remote counterpart. Integrate the remote changes (e.g.
hint: 'git pull ..') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

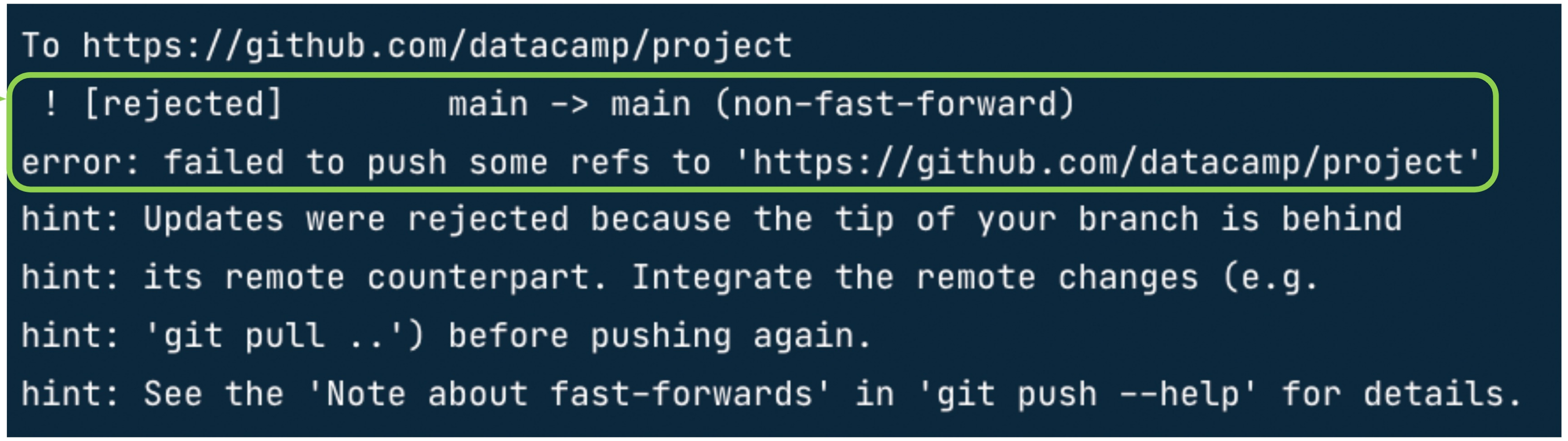
Remote/local conflicts



```
To https://github.com/datacamp/project
 ! [rejected]        main -> main (non-fast-forward)
error: failed to push some refs to 'https://github.com/datacamp/project'
hint: Updates were rejected because the tip of your branch is behind
hint: its remote counterpart. Integrate the remote changes (e.g.
hint: 'git pull ..') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

Remote/local conflicts

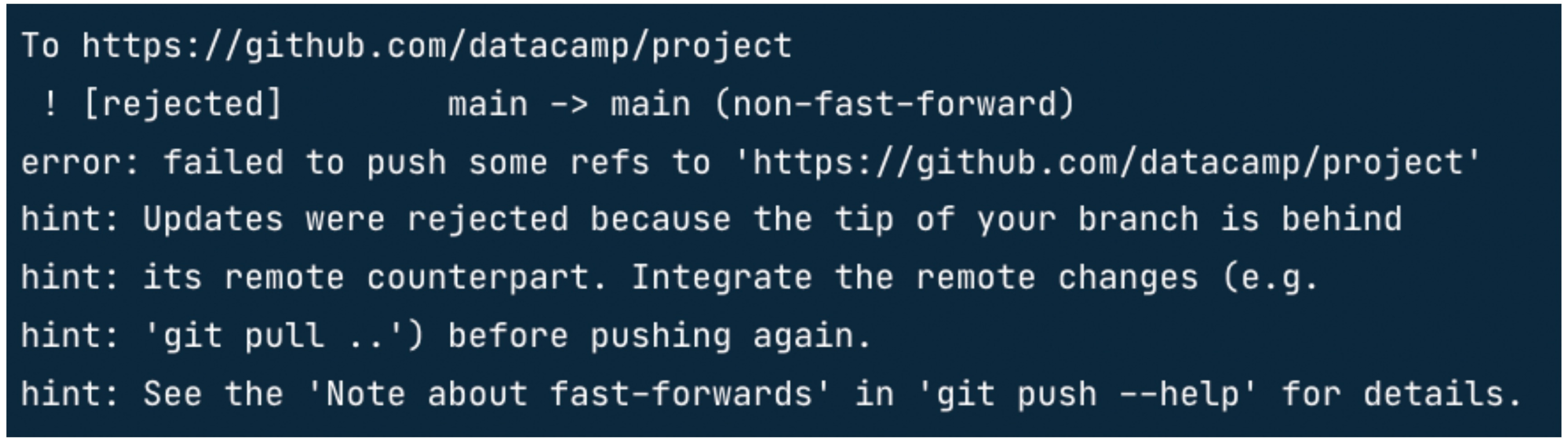
Outcome



```
To https://github.com/datacamp/project
 ! [rejected]        main -> main (non-fast-forward)
error: failed to push some refs to 'https://github.com/datacamp/project'
hint: Updates were rejected because the tip of your branch is behind
hint: its remote counterpart. Integrate the remote changes (e.g.
hint: 'git pull ..') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

Remote/local conflicts

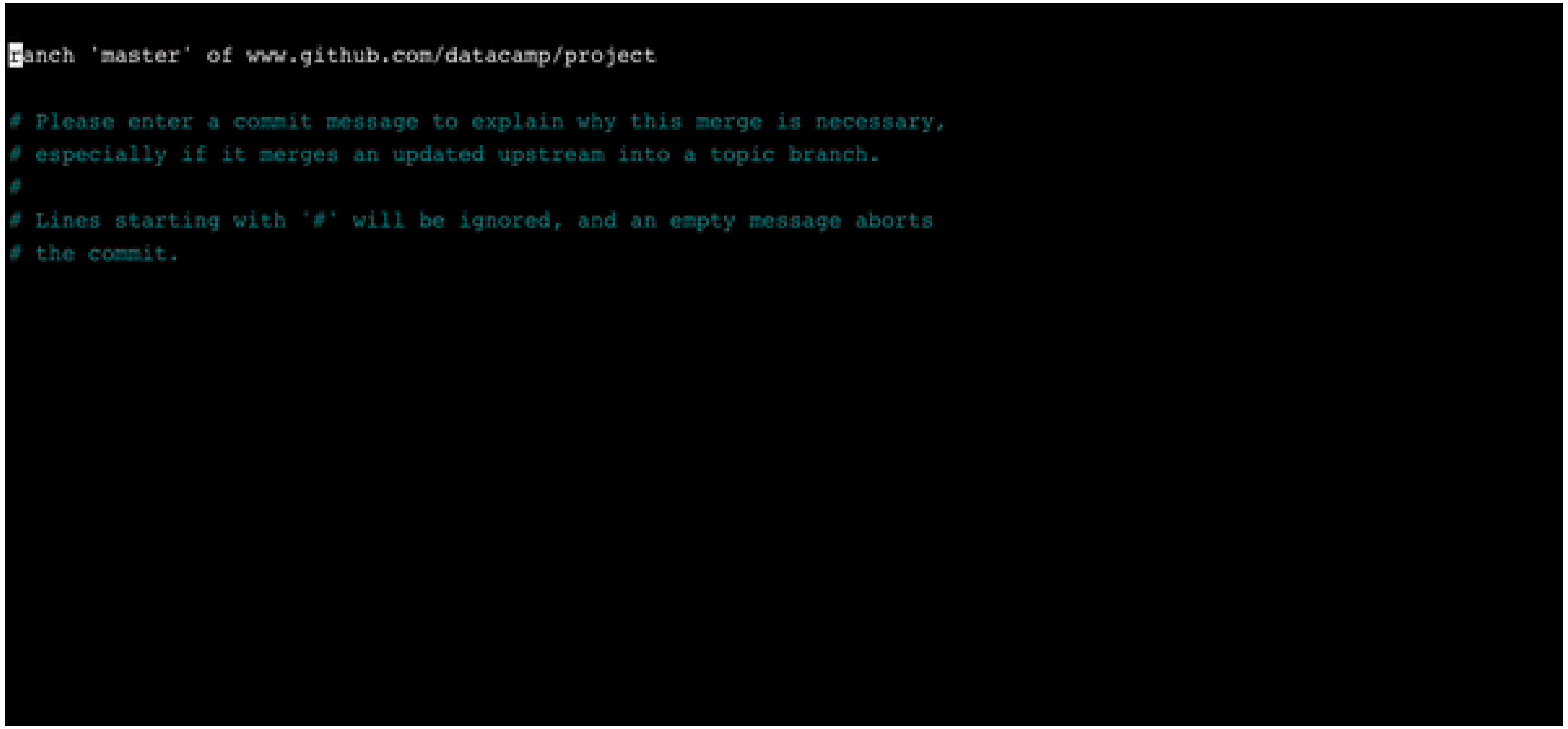
Reason(s) and suggestion(s)



```
To https://github.com/datacamp/project
 ! [rejected]        main -> main (non-fast-forward)
error: failed to push some refs to 'https://github.com/datacamp/project'
hint: Updates were rejected because the tip of your branch is behind
hint: its remote counterpart. Integrate the remote changes (e.g.
hint: 'git pull ..') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

Resolving a conflict

```
git pull origin main
```



```
ranch 'master' of www.github.com/datacamp/project

# Please enter a commit message to explain why this merge is necessary,
# especially if it merges an updated upstream into a topic branch.
#
# Lines starting with '#' will be ignored, and an empty message aborts
# the commit.
```

Avoid leaving a message

```
git pull --no-edit origin main
```

Resolving a conflict

```
Merge made by the 'recursive' strategy.
 report.md | 1 +
 1 file changed, 1 insertion(+)
```

- Default merge strategy for Git v0.99.9 - v2.33.0:

Pushing to the remote

```
git push origin main
```

```
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 10% (3/3), done.
Writing objects: 100% (3/3), 325 bytes | 325.00 KiB/s, done.
To https://github.com/datacamp/project
    01384d2..0a1dbf6  main -> main
```

Let's practice!

INTRODUCTION TO GIT

Congratulations!

INTRODUCTION TO GIT

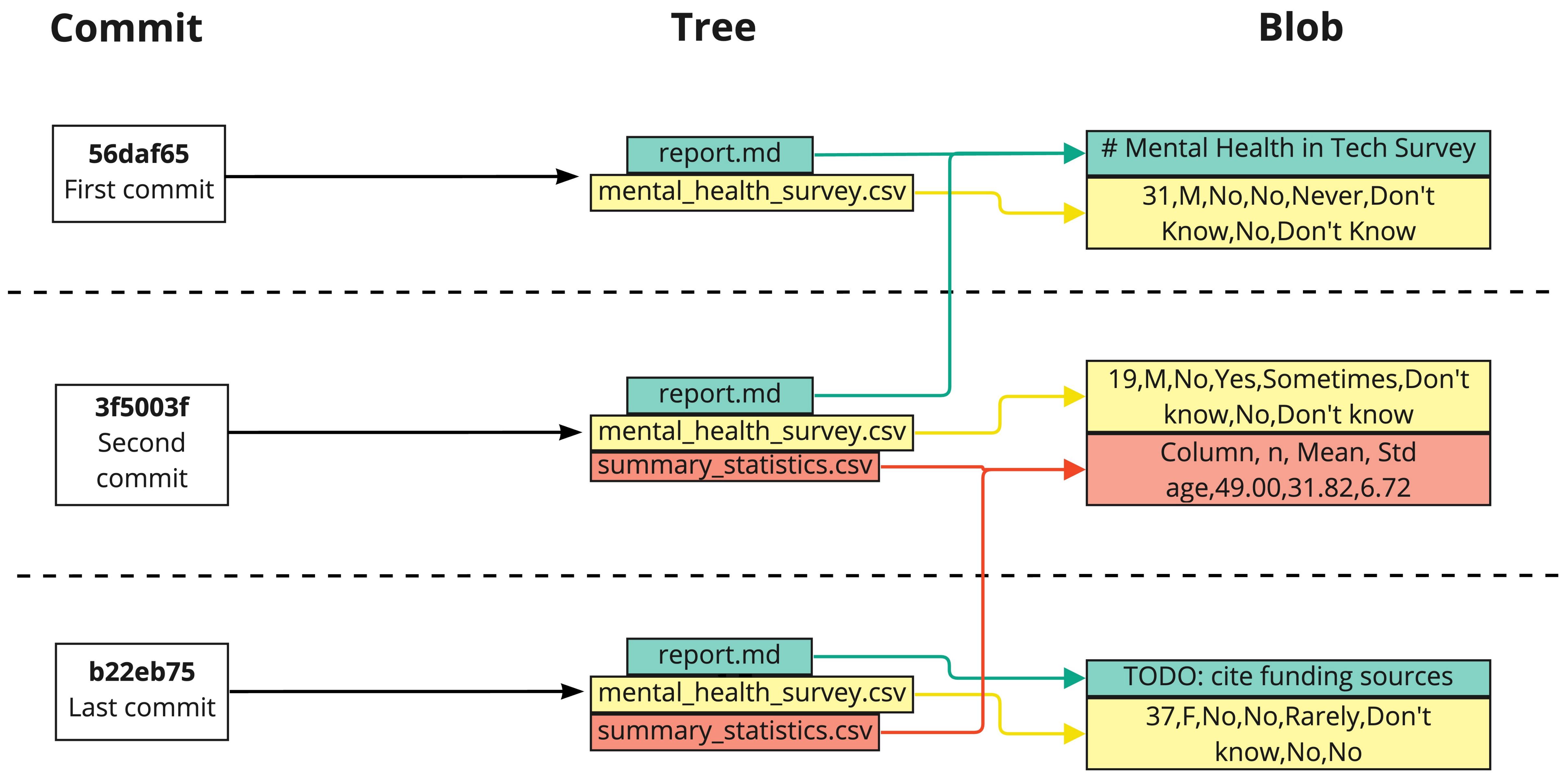
George Boorman
Curriculum Manager, DataCamp

What you've covered

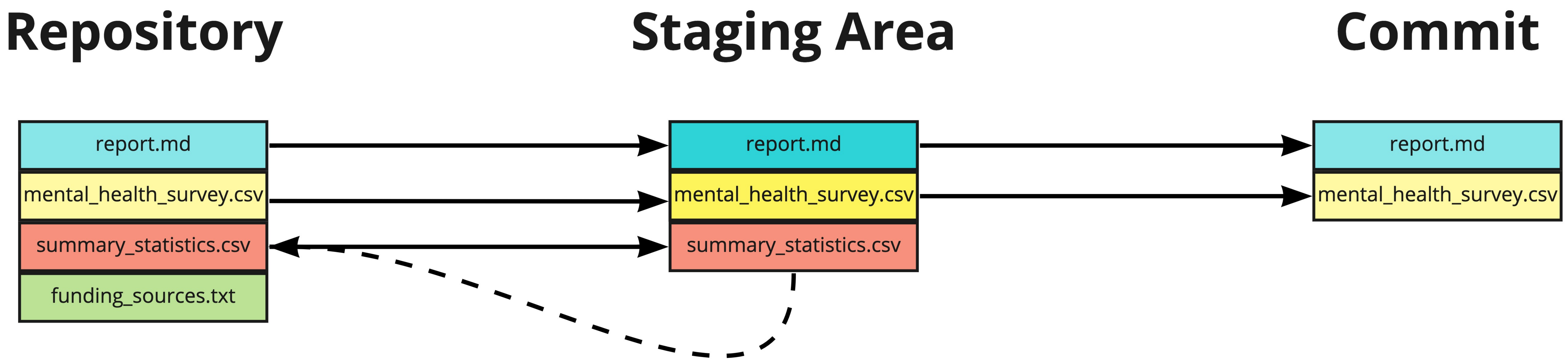
- What a version is
- Why version control is important
- Using Git to:
  - save files
  - compare files

What you've covered

- How Git stores data



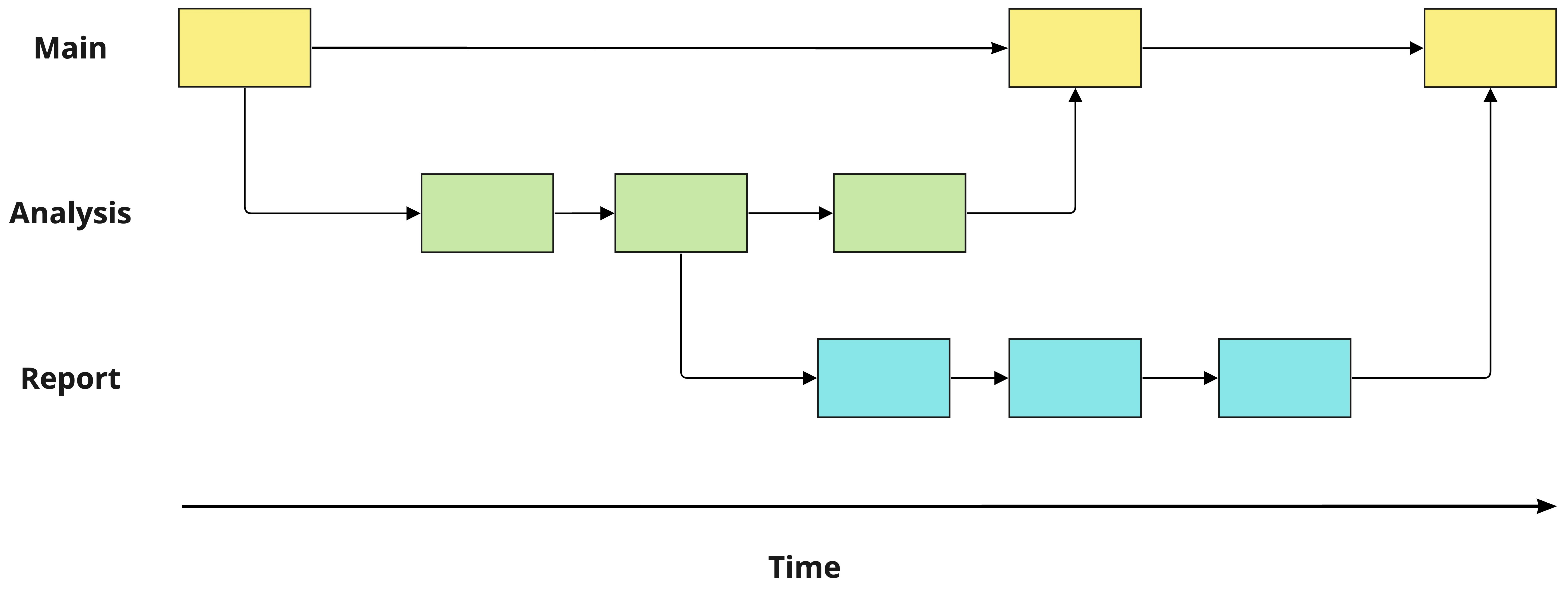
What you've covered



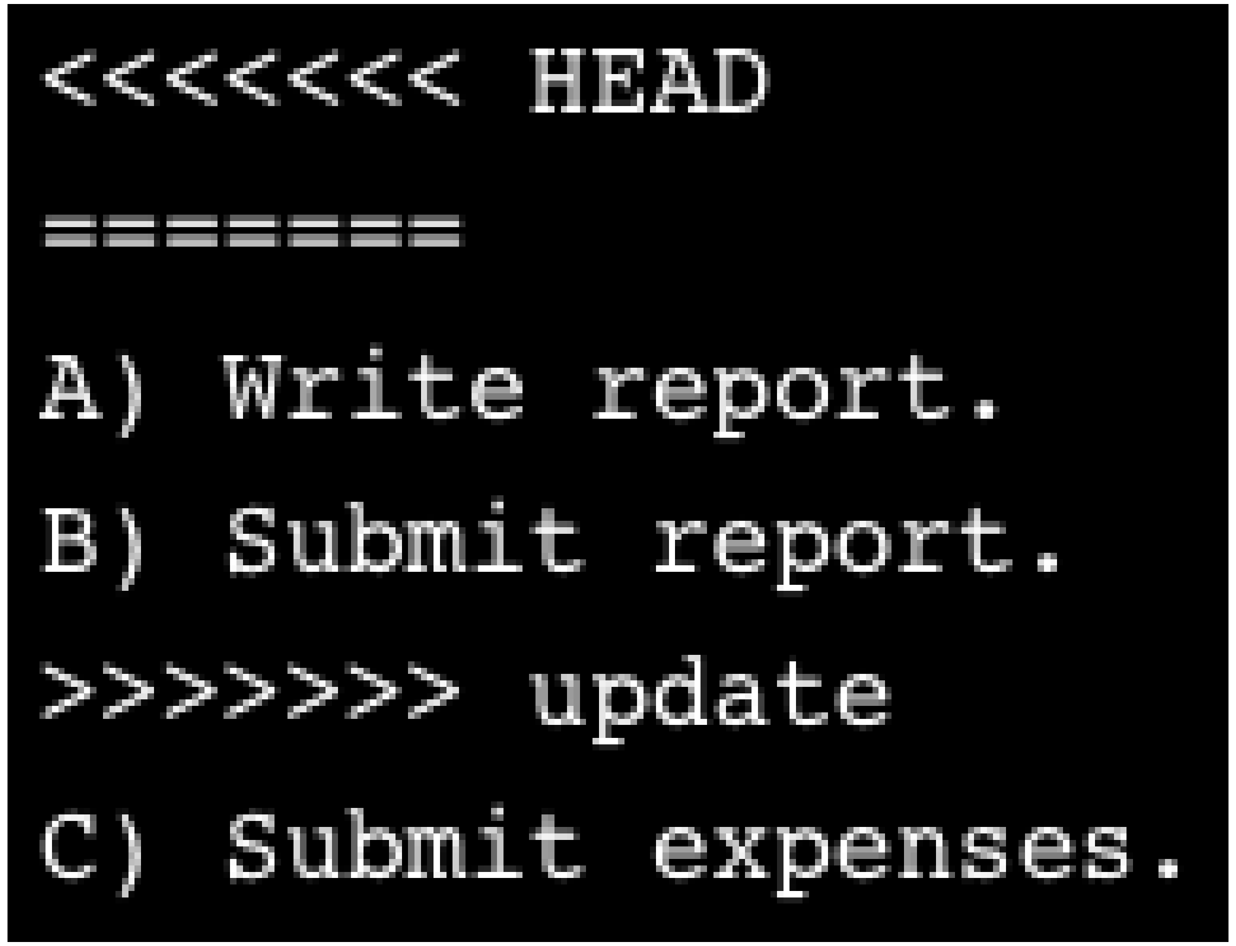
What you've covered

- Configuration

```
git config --global alias.unstage 'reset HEAD'
```

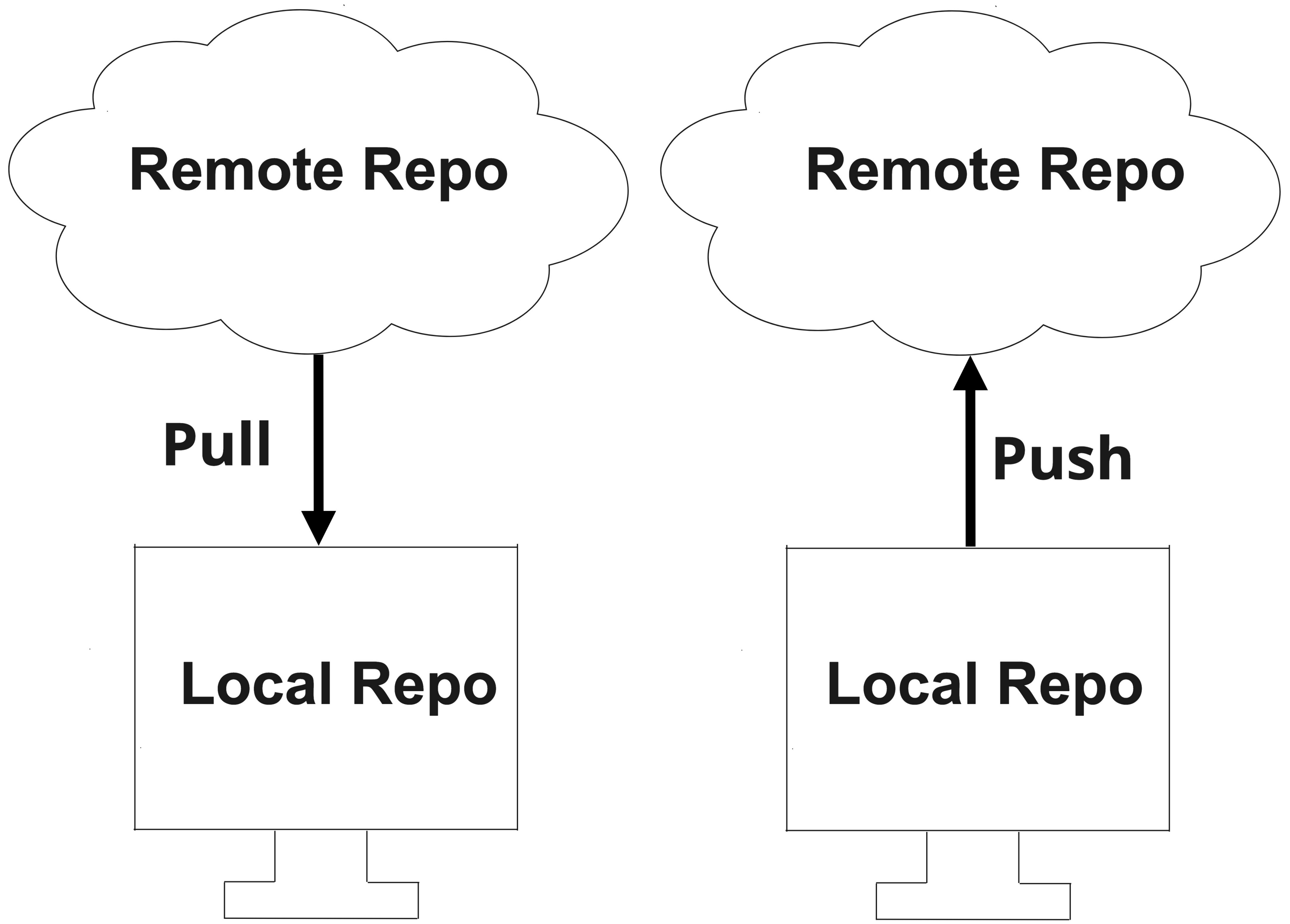


What you've covered



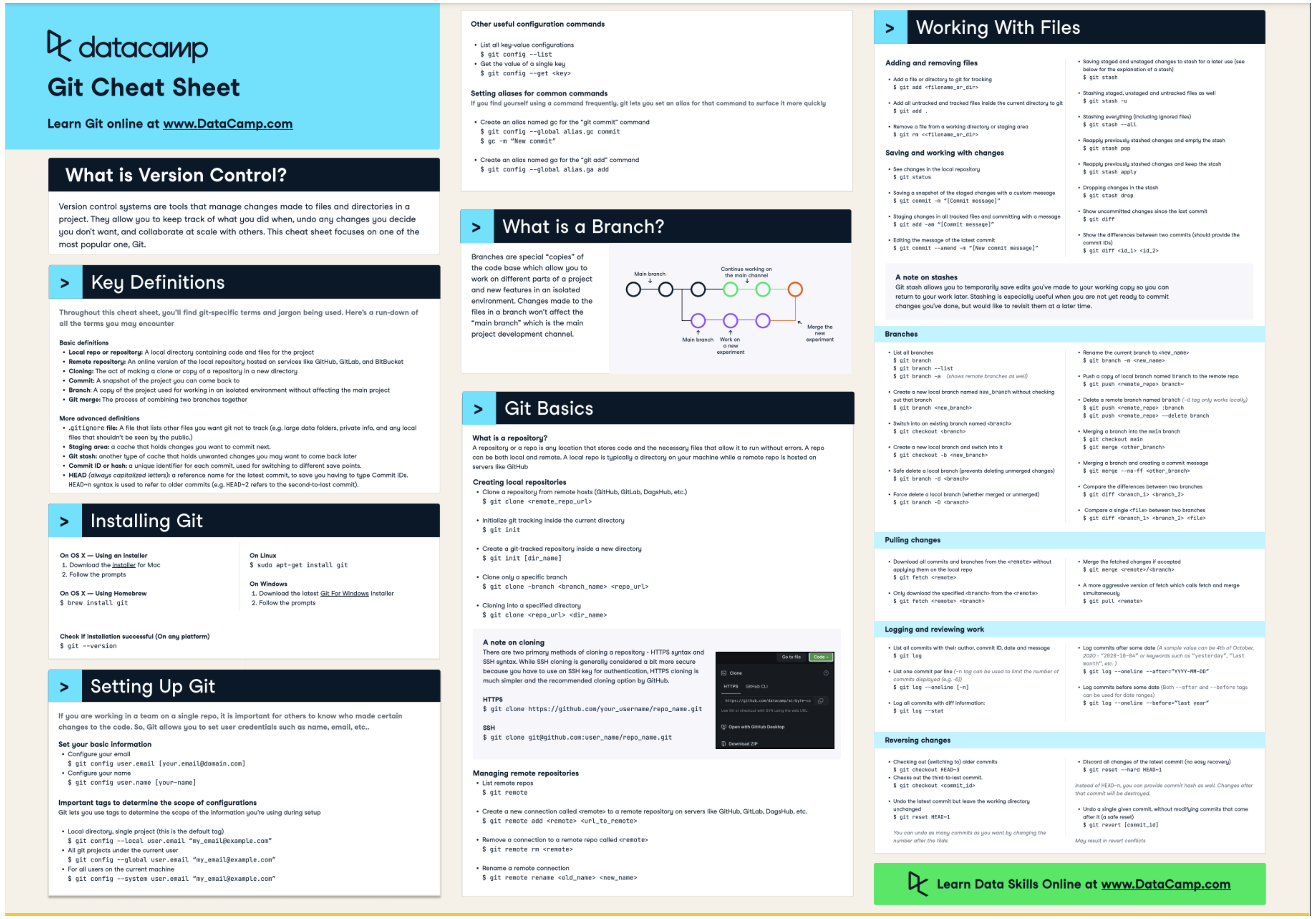
```
<<<<<<< HEAD
=======
A) Write report.
B) Submit report.
>>>>>>> update
C) Submit expenses.
```

What you've covered



Git cheat sheet

- https://www.datacamp.com/cheat-sheet/git-cheat-sheet



datacamp

Git Cheat Sheet

Learn Git online at www.DataCamp.com

What is Version Control?

Version control systems are tools that manage changes made to files and directories in a project. They allow you to keep track of what you did when, undo any changes you decide you don't want, and collaborate at scale with others. This cheat sheet focuses on one of the most popular one, Git.

> Key Definitions

Throughout this cheat sheet, you'll find git-specific terms and jargon being used. Here's a run-down of all the terms you may encounter

Basic definitions
- **Local repo or repository:** A local directory containing code and files for the project
- **Remote repository:** An online version of the local repository hosted on services like GitHub, GitLab, and BitBucket
- **Cloning:** The act of making a clone or copy of a repository in a new directory
- **Commit:** A snapshot of the project you can come back to
- **Branch:** A copy of the project used for working in an isolated environment without affecting the main project
- **Git merge:** The process of combining two branches together

More advanced definitions
- `.gitignore` **file:** A file that lists other files you want git not to track (e.g. large data folders, private info, and any local files that shouldn't be seen by the public.)
- **Staging area:** a cache that holds changes you want to commit next.
- **Git stash:** another type of cache that holds unwanted changes you may want to come back later
- **Commit ID or hash:** a unique identifier for each commit, used for switching to different save points.
- **HEAD** *(always capitalized letters)*: a reference name for the latest commit, to save you having to type Commit IDs. `HEAD~n` syntax is used to refer to older commits (e.g. `HEAD~2` refers to the second-to-last commit).

> Installing Git

On OS X — Using an installer
1. Download the installer for Mac
2. Follow the prompts

On OS X — Using Homebrew
```
$ brew install git
```

On Linux
```
$ sudo apt-get install git
```

On Windows
1. Download the latest Git For Windows installer
2. Follow the prompts

Check if installation successful (On any platform)
```
$ git --version
```

> Setting Up Git

If you are working in a team on a single repo, it is important for others to know who made certain changes to the code. So, Git allows you to set user credentials such as name, email, etc..

Set your basic information
- Configure your email
```
$ git config user.email [your.email@domain.com]
```
- Configure your name
```
$ git config user.name [your-name]
```

Important tags to determine the scope of configurations
Git lets you use tags to determine the scope of the information you're using during setup

- Local directory, single project (this is the default tag)
```
$ git config --local user.email "my_email@example.com"
```
- All git projects under the current user
```
$ git config --global user.email "my_email@example.com"
```
- For all users on the current machine
```
$ git config --system user.email "my_email@example.com"
```

Other useful configuration commands

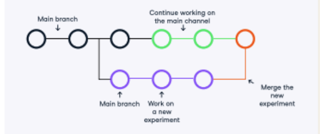
- List all key-value configurations
```
$ git config --list
```
- Get the value of a single key
```
$ git config --get <key>
```

Setting aliases for common commands
If you find yourself using a command frequently, git lets you set an alias for that command to surface it more quickly

- Create an alias named gc for the "git commit" command
```
$ git config --global alias.gc commit
$ gc -m "New commit"
```
- Create an alias named ga for the "git add" command
```
$ git config --global alias.ga add
```

> What is a Branch?

Branches are special "copies" of the code base which allow you to work on different parts of a project and new features in an isolated environment. Changes made to the files in a branch won't affect the "main branch" which is the main project development channel.



> Git Basics

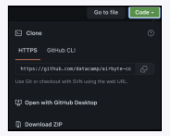
What is a repository?
A repository or a repo is any location that stores code and the necessary files that allow it to run without errors. A repo can be both local and remote. A local repo is typically a directory on your machine while a remote repo is hosted on servers like GitHub

Creating local repositories
- Clone a repository from remote hosts (GitHub, GitLab, DagsHub, etc.)
```
$ git clone <remote_repo_url>
```
- Initialize git tracking inside the current directory
```
$ git init
```
- Create a git-tracked repository inside a new directory
```
$ git init [dir_name]
```
- Clone only a specific branch
```
$ git clone -branch <branch_name> <repo_url>
```
- Cloning into a specified directory
```
$ git clone <repo_url> <dir_name>
```

A note on cloning
There are two primary methods of cloning a repository - HTTPS syntax and SSH syntax. While SSH cloning is generally considered a bit more secure because you have to use an SSH key for authentication, HTTPS cloning is much simpler and the recommended cloning option by GitHub.

HTTPS
```
$ git clone https://github.com/your_username/repo_name.git
```

SSH
```
$ git clone git@github.com:user_name/repo_name.git
```



```
https://github.com/datacamp/airbyte-co
```
Use Git or checkout with SVN using the web URL.

Managing remote repositories
- List remote repos
```
$ git remote
```
- Create a new connection called <remote> to a remote repository on servers like GitHub, GitLab, DagsHub, etc.
```
$ git remote add <remote> <url_to_remote>
```
- Remove a connection to a remote repo called <remote>
```
$ git remote rm <remote>
```
- Rename a remote connection
```
$ git remote rename <old_name> <new_name>
```

> Working With Files

Adding and removing files
- Add a file or directory to git for tracking
```
$ git add <filename_or_dir>
```
- Add all untracked and tracked files inside the current directory to git
```
$ git add .
```
- Remove a file from a working directory or staging area
```
$ git rm <<filename_or_dir>
```

Saving and working with changes
- See changes in the local repository
```
$ git status
```
- Saving a snapshot of the staged changes with a custom message
```
$ git commit -m "[Commit message]"
```
- Staging changes in all tracked files and committing with a message
```
$ git add -am "[Commit message]"
```
- Editing the message of the latest commit
```
$ git commit --amend -m "[New commit message]"
```
- Saving staged and unstaged changes to stash for a later use (see below for the explanation of a stash)
```
$ git stash
```
- Stashing staged, unstaged and untracked files as well
```
$ git stash -u
```
- Stashing everything (including ignored files)
```
$ git stash --all
```
- Reapply previously stashed changes and empty the stash
```
$ git stash pop
```
- Reapply previously stashed changes and keep the stash
```
$ git stash apply
```
- Dropping changes in the stash
```
$ git stash drop
```
- Show uncommitted changes since the last commit
```
$ git diff
```
- Show the differences between two commits (should provide the commit IDs)
```
$ git diff <id_1> <id_2>
```

A note on stashes
Git stash allows you to temporarily save edits you've made to your working copy so you can return to your work later. Stashing is especially useful when you are not yet ready to commit changes you've done, but would like to revisit them at a later time.

Branches
- List all branches
```
$ git branch
$ git branch --list
$ git branch -a   (shows remote branches as well)
```
- Create a new local branch named new_branch without checking out that branch
```
$ git branch <new_branch>
```
- Switch into an existing branch named <branch>
```
$ git checkout <branch>
```
- Create a new local branch and switch into it
```
$ git checkout -b <new_branch>
```
- Safe delete a local branch (prevents deleting unmerged changes)
```
$ git branch -d <branch>
```
- Force delete a local branch (whether merged or unmerged)
```
$ git branch -D <branch>
```
- Rename the current branch to <new_name>
```
$ git branch -m <new_name>
```
- Push a copy of local branch named branch to the remote repo
```
$ git push <remote_repo> branch~
```
- Delete a remote branch named branch *(-d tag only works locally)*
```
$ git push <remote_repo> :branch
$ git push <remote_repo> --delete branch
```
- Merging a branch into the main branch
```
$ git checkout main
$ git merge <other_branch>
```
- Merging a branch and creating a commit message
```
$ git merge --no-ff <other_branch>
```
- Compare the differences between two branches
```
$ git diff <branch_1> <branch_2>
```
- Compare a single <file> between two branches
```
$ git diff <branch_1> <branch_2> <file>
```

Pulling changes
- Download all commits and branches from the <remote> without applying them on the local repo
```
$ git fetch <remote>
```
- Only download the specified <branch> from the <remote>
```
$ git fetch <remote> <branch>
```
- Merge the fetched changes if accepted
```
$ git merge <remote>/<branch>
```
- A more aggressive version of fetch which calls fetch and merge simultaneously
```
$ git pull <remote>
```

Logging and reviewing work
- List all commits with their author, commit ID, date and message
```
$ git log
```
- List one commit per line *(-n tag can be used to limit the number of commits displayed (e.g. -5))*
```
$ git log --oneline [-n]
```
- Log all commits with diff information:
```
$ git log --stat
```
- Log commits after some date *(A sample value can be 4th of October, 2020 - "2020-10-04" or keywords such as "yesterday", "last month", etc.)*
```
$ git log --oneline --after="YYYY-MM-DD"
```
- Log commits before some date (Both --after and --before tags can be used for date ranges)
```
$ git log --oneline --before="last year"
```

Reversing changes
- Checking out (switching to) older commits
```
$ git checkout HEAD~3
```
- Checks out the third-to-last commit.
```
$ git checkout <commit_id>
```
- Undo the latest commit but leave the working directory unchanged
```
$ git reset HEAD~1
```
You can undo as many commits as you want by changing the number after the tilde.

- Discard all changes of the latest commit (no easy recovery)
```
$ git reset --hard HEAD~1
```
Instead of HEAD~n, you can provide commit hash as well. Changes after that commit will be destroyed.

- Undo a single given commit, without modifying commits that come after it (a safe reset)
```
$ git revert [commit_id]
```
May result in revert conflicts

Learn Data Skills Online at www.DataCamp.com

Thank you!

INTRODUCTION TO GIT