git tutorial

What we'll cover

- Tech check
- Why?
- Basic operations
- Moving to GitHub
- Branching

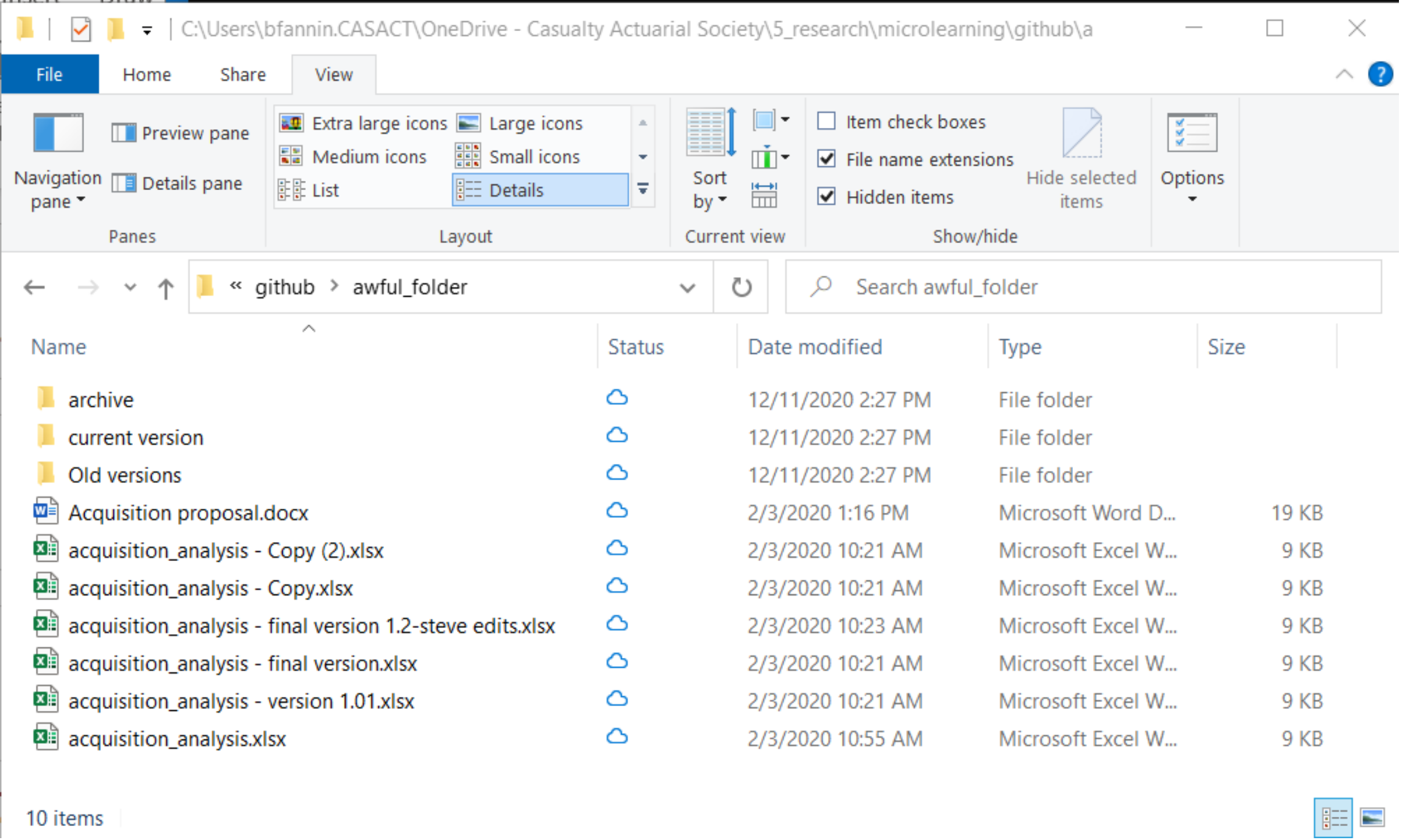
Tech check

1. Have you installed git?
2. Do you have one of the following:
   - RStudio
   - Atom
   - GitHub desktop
   - Sourcetree
   - Something else?
3. Make sure you can find a command prompt
   - Most decent IDEs make this easy

Why?

Alternatives

- Filename
- E-mail
- Office suite/cloud backup
- Track changes


C:\Users\bfannin.CASACT\OneDrive - Casualty Actuarial Society\5_research\microlearning\github\a
File
Home
Share
View
Navigation pane
Preview pane
Details pane
Panes
Extra large icons
Large icons
Medium icons
Small icons
List
Details
Layout
Sort by
Current view
Item check boxes
File name extensions
Hidden items
Hide selected items
Show/hide
Options
« github › awful_folder
Search awful_folder
Name
Status
Date modified
Type
Size
archive 12/11/2020 2:27 PM File folder
current version 12/11/2020 2:27 PM File folder
Old versions 12/11/2020 2:27 PM File folder
Acquisition proposal.docx 2/3/2020 1:16 PM Microsoft Word D... 19 KB
acquisition_analysis - Copy (2).xlsx 2/3/2020 10:21 AM Microsoft Excel W... 9 KB
acquisition_analysis - Copy.xlsx 2/3/2020 10:21 AM Microsoft Excel W... 9 KB
acquisition_analysis - final version 1.2-steve edits.xlsx 2/3/2020 10:23 AM Microsoft Excel W... 9 KB
acquisition_analysis - final version.xlsx 2/3/2020 10:21 AM Microsoft Excel W... 9 KB
acquisition_analysis - version 1.01.xlsx 2/3/2020 10:21 AM Microsoft Excel W... 9 KB
acquisition_analysis.xlsx 2/3/2020 10:55 AM Microsoft Excel W... 9 KB
10 items

Filename as a version control system

- Absolutely no non-manual governance
  - Naming convention is a social contract
  - No way to enforce this with technology
- Not foolproof at all
  - Nothing to stop the timestamp and filename from getting out of sync
  - For that matter, nothing stopping me from changing the timestamp of a file to anything I want
- Changes from one version to another are not at all clear
  - We love manual processes!
- Only applicable to one file
  - Could use naming convention for a directory, but complexity only magnifies

E-mail as a version control system

"I think I sent it on or about the 12th of June. Maybe in the morning. I know that I'd just eaten a burrito, but I can't remember if it was a *breakfast* burrito or a *regular* burrito."

Office suite/cloud backup

- Better than nothing, but falls well short
- Passive backup != version control
  - I want to be deliberate about changes. I know when I've made progress.
  - Some things don't need to be memorialized. They're noise and don't need to be backed up.
- What's a good restore point? Panning for gold
- No documentation of the evolution of the thinking. Must be (re)constructed by comparing successive versions.

Office suite/cloud track changes

- Only one file at a time
- Comments and changes disappear after they've been accepted.
- End goal is to have all of the changes removed from the document, giving us a "clean" copy.
- Collaboration on one copy of the file. My version == your version.
- Nonexistent for spreadsheets
- Nonexistent for scripts

Git to the rescue!

Getting started

1. Fire up Atom, RStudio, etc.
2. Create a project

Basic git operations

1. Create a new project
2. Create a new file
3. Stage
4. Commit
5. Make some changes
6. Revert changes
7. Ignore

If you're using Atom

If you're using RStudio

git init

1. Have a look at the project folder
2. Take a gander at the support in the IDE
3. Open a terminal prompt and run

```
git status
git log
git --help
git --version
git init -h
git init --help
```

Basic git operations

1. ~~Create a new project~~
2. Create a new file
3. Stage
4. Commit
5. Make some changes
6. Revert changes
7. Ignore

Create a new file

1. Code the following:
   ```
   print("Hello, world!")
   ```
2. Save the file as:
   ```
   hello.py
   ```
3. Open a terminal prompt and run
   ```
   git status
   ```

File status

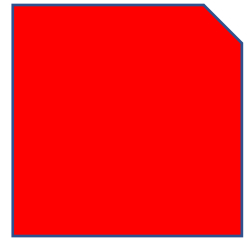
- When a file is first created it is not tracked.
  - If you stage it, it will be tracked.
  - If you commit it, after staging, that file will be tracked forever.
  - If you ignore it, you won't see it again.
- Once a file is being tracked, changes will show as being "not staged for commit"

Untracked

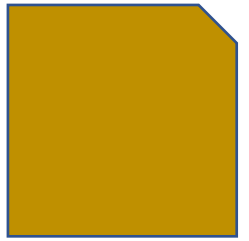
`git add`

Changes not staged

`git add`

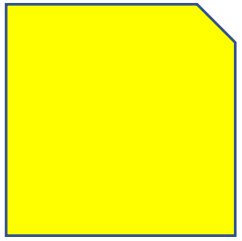


Ignored



Staged

`git commit`



`git reset`

Committed

`git checkout`
`git revert`

Untracked

Changes not staged

Ignored

`git add`

Staged

Committed

Basic git operations

1. ~~Create a new project~~
2. ~~Create a new file~~
3. Stage
4. Commit
5. Make some changes
6. Revert changes
7. Ignore

Stage changes

1. Open a terminal prompt and run

```
git status
git add my_file.py
git status
```

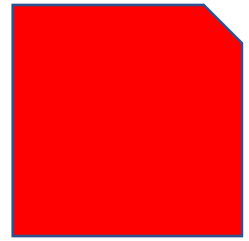
2. Also explore how this looks in your IDE

Untracked

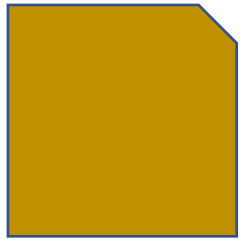
`git add`

Changes not staged

`git add`

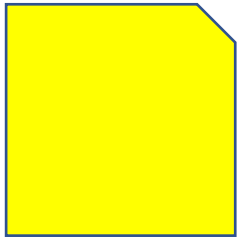


Ignored



Staged

`git commit`



`git reset`

Committed

`git checkout`
`git revert`

Stage vs. commit

- Stage
  - I'm pretty sure that I'm done making changes.
  - This is the set of changes that I'm planning to commit.
  - Necessary step before committing
- Commit
  - We're all good here. This is a unit of work and here's some commentary.
  - (Almost) no turning back

"It's like ham and eggs: the chicken is merely *involved* with the ham and eggs. The pig? He's committed."

Stage changes

1. Open a terminal prompt and run

```
git status
git add my_file.py
git status
```

2. Also explore how this looks in your IDE
3. Back at the terminal

```
git reset
git status
git add my_file.py
git status
```

Basic git operations

1. ~~Create a new project~~
2. ~~Create a new file~~
3. ~~Stage~~
4. Commit
5. Make some changes
6. Revert changes
7. Ignore

Untracked

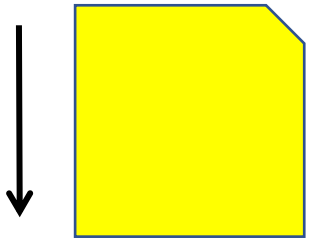
Changes not staged

Ignored

`git add`

Staged

`git commit`



Committed

Commit changes

1. Open a terminal prompt and run

```
git status
git commit –m "Initial commit for hello.py"
git status
```

2. Also explore how this looks in your IDE

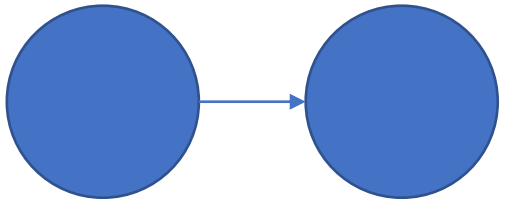
Stage vs. commit

- Stage
  - I'm pretty sure that I'm done making changes.
  - This is the set of changes that I'm planning to commit.
- Commit
  - We're all good here. This is a unit of work and here's some commentary.
  - (Almost) no turning back

Commit changes

1. Open a terminal prompt and run
   ```
   git log
   ```
2. Also explore how this looks in your IDE

Visual metaphor for commits



Basic git operations

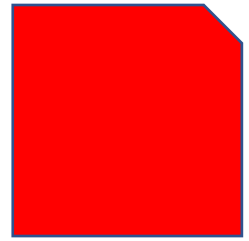
1. ~~Create a new project~~
2. ~~Create a new file~~
3. ~~Stage~~
4. ~~Commit~~
5. Make some changes
6. Revert changes
7. Ignore

Untracked

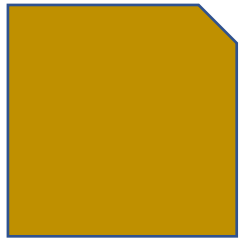
`git add`

Changes not staged

`git add`

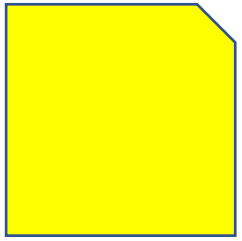


Ignored



Staged

`git commit`



`git reset`

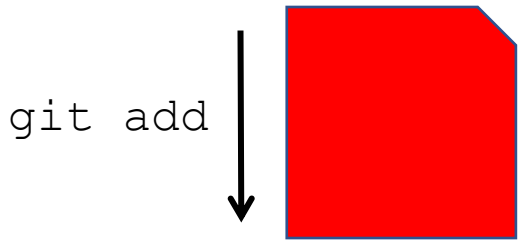
Committed

`git checkout`
`git revert`

Untracked

Changes not staged

Ignored



Staged

Committed

Make more changes

1. Edit your file in some way
2. Open a terminal prompt and run

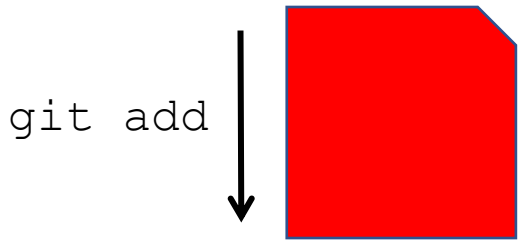
```
git status
git diff hello.py
git add hello.py
```

3. Also explore how this looks in your IDE

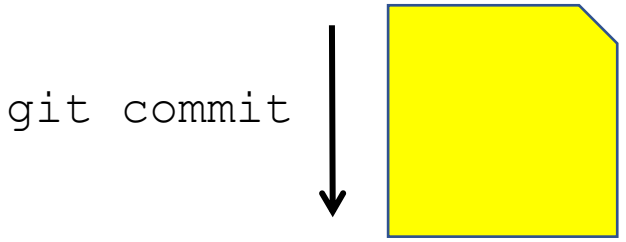
Untracked

Changes not staged

Ignored



Staged



Committed

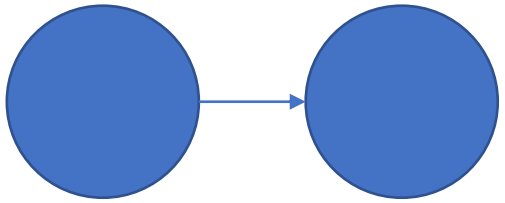
Commit the change

1. Edit your file in some way
2. Open a terminal prompt and run

```
git status
git commit –m “Made a change”
```

3. Also explore how this looks in your IDE

Version history!



Basic git operations

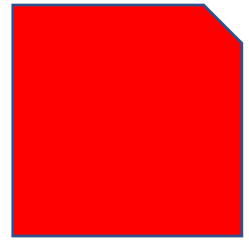
1. ~~Create a new project~~
2. ~~Create a new file~~
3. ~~Stage~~
4. ~~Commit~~
5. ~~Make some changes~~
6. Revert changes
7. Ignore

Untracked

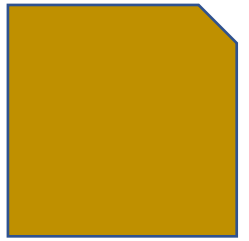
`git add`

Changes not staged

`git add`

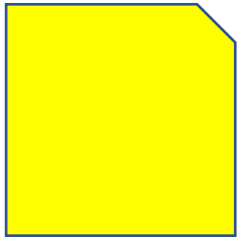


Ignored



Staged

`git commit`



`git reset`

Committed

`git checkout`
`git revert`

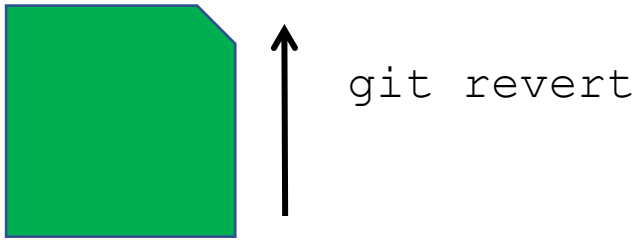
Untracked

Changes not staged

Ignored

Staged

Committed



`git revert`

Revert a commit

1. Open a terminal prompt and run

```
git status
git revert
git commit –m “Made a change”
```

2. Also explore how this looks in your IDE

Basic git operations

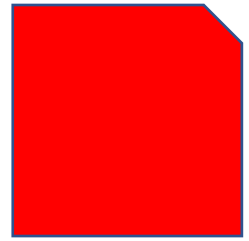
1. ~~Create a new project~~
2. ~~Create a new file~~
3. ~~Stage~~
4. ~~Commit~~
5. ~~Make some changes~~
6. ~~Revert changes~~
7. Ignore

Untracked

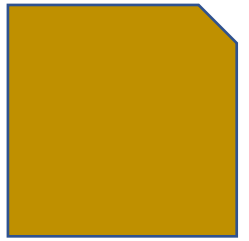
`git add`

Changes not staged

`git add`

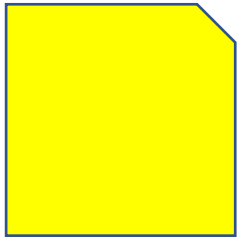


Ignored



Staged

`git commit`



`git reset`

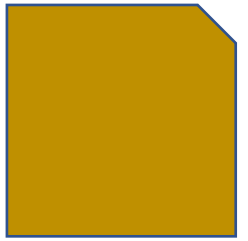
Committed

`git checkout`
`git revert`

Untracked

Changes not staged

Ignored



Staged

Committed

Ignoring files

1. Do this live

Basic git operations

1. ~~Create a new project~~
2. ~~Create a new file~~
3. ~~Stage~~
4. ~~Commit~~
5. ~~Make some changes~~
6. ~~Revert changes~~
7. ~~Ignore~~

Moving to GitHub

Version Control in the Cloud

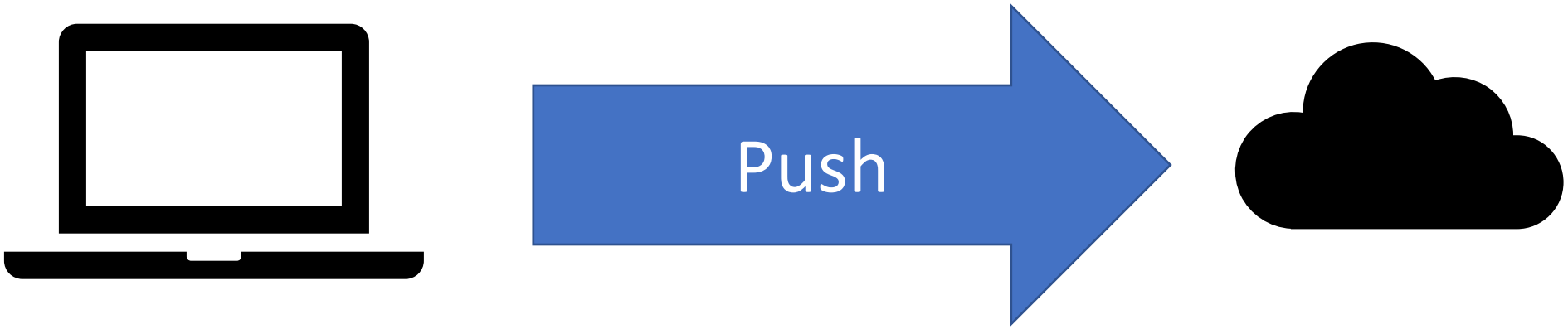
- Similar to DropBox, Google Docs, etc.
  - HOWEVER! No automatic sync to your device.
  - ALSO! Very limited cloud-only use. Most use cases assume manual sync between your device and the cloud.
- Also BitBucket, GitLab

Account Features

- Contribution activity
- My repositories
- Stars
- Followers
- Following

Repo Features

- Issue tracking
- Wiki
- Basic business intelligence
- Integrations

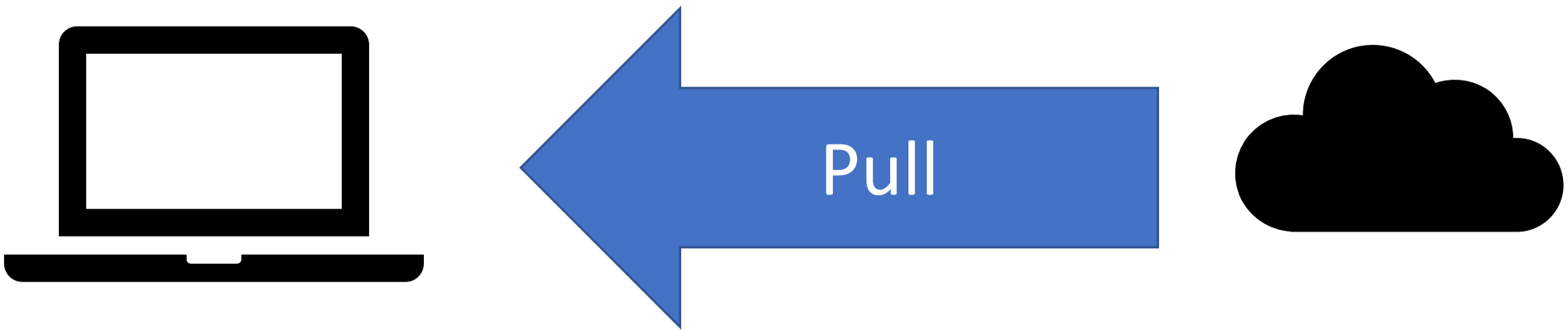

Push

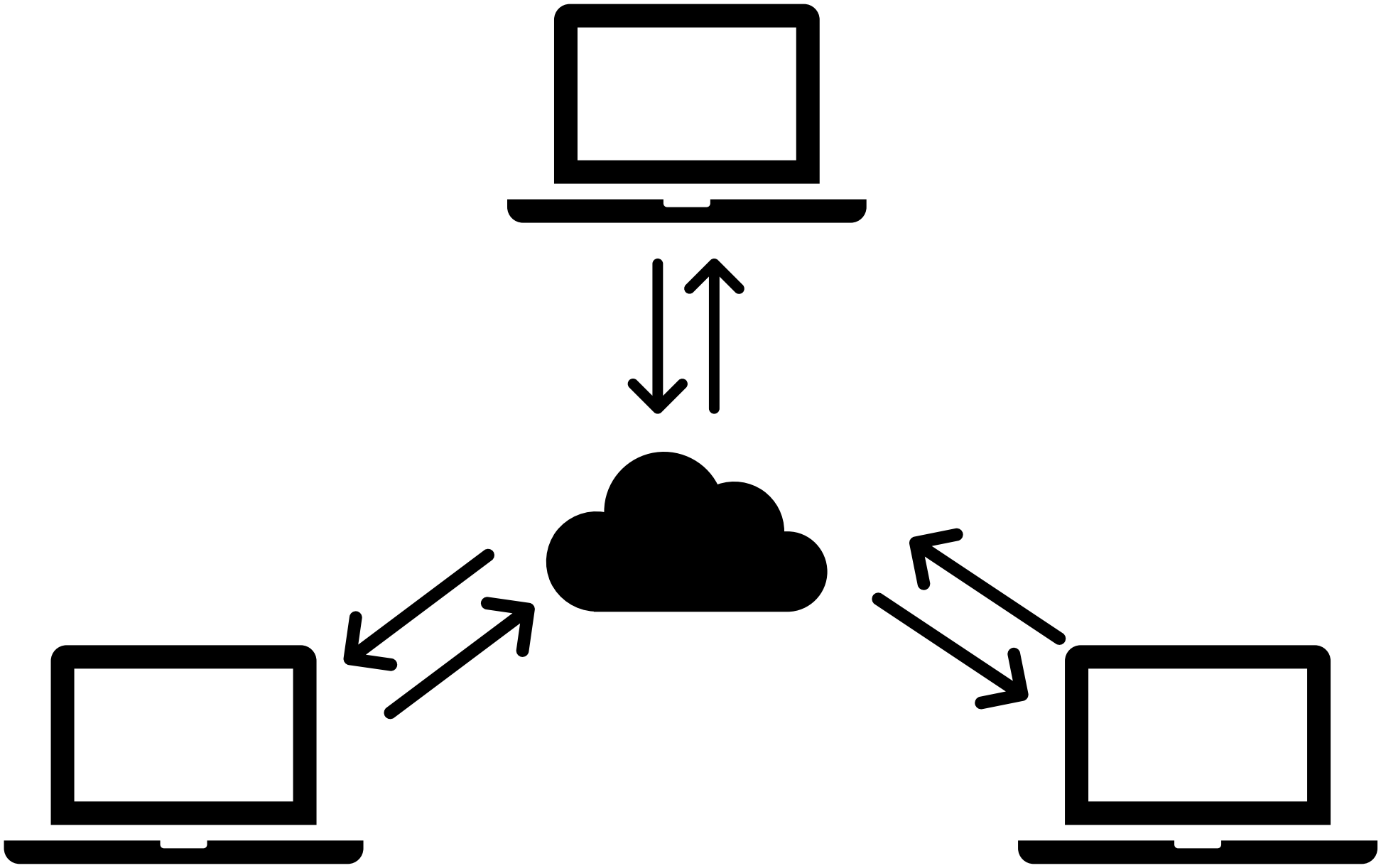
Push to the cloud

1. Ensure that we have a repo
2. Open a terminal prompt and run

```
git status
git remote -v
git remote add origin https://github.com/user/repo.git
git remote -v
git push origin
```

3. Also explore how this looks in RStudio and Atom


Pull



Update changes from the cloud

1. Open a terminal prompt and run

```
git status
git pull origin
git status
```

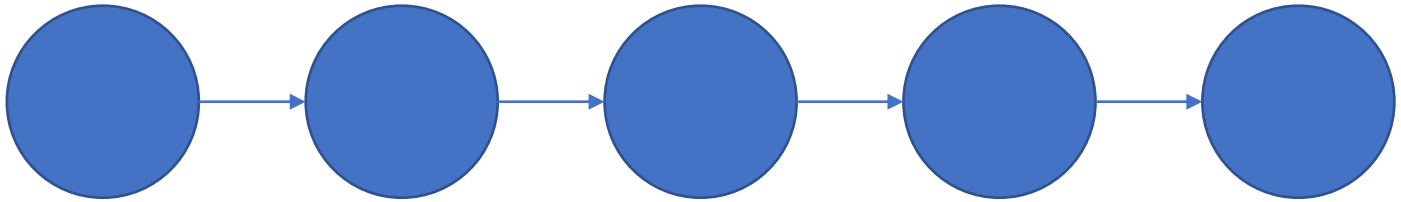
2. Also explore how this looks in your IDE

Branching

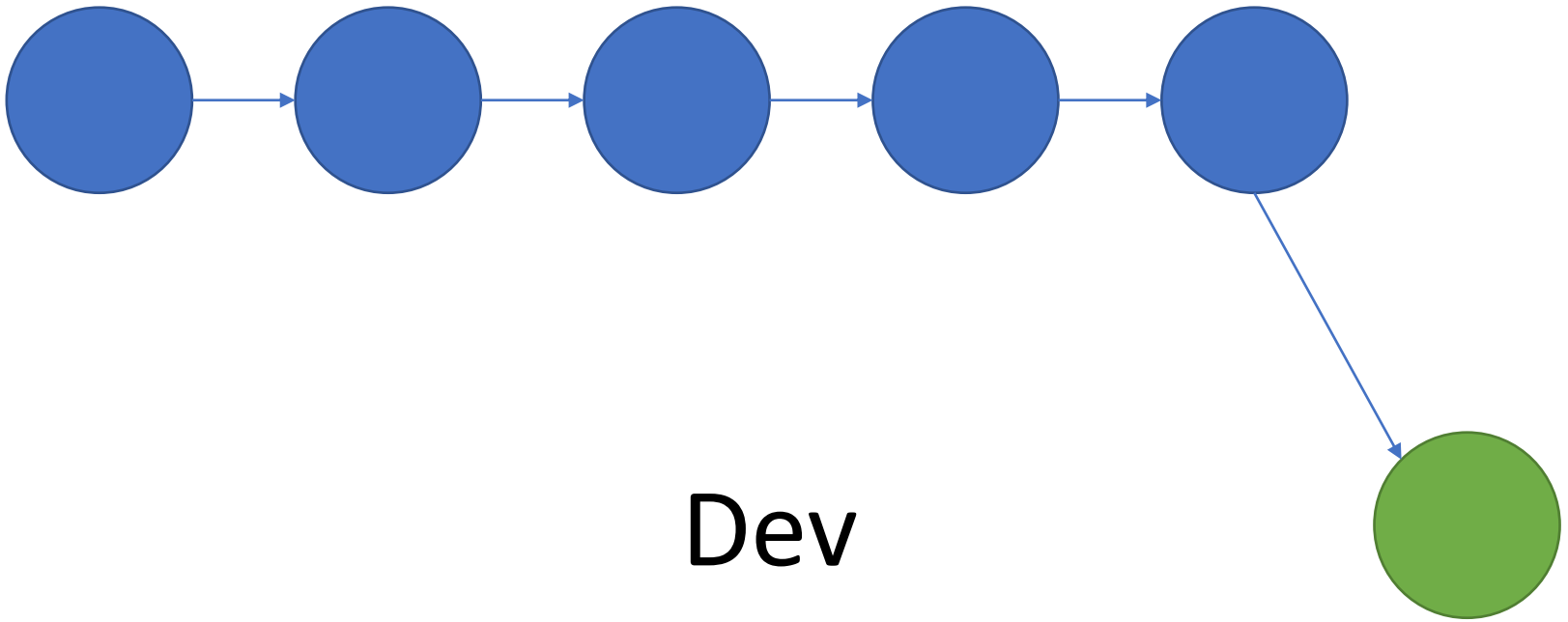
Branching

- Main or “clean” version of the code
- Branch for:
  - Feature development
  - Testing
  - Debugging/hotfixing

Main



Main



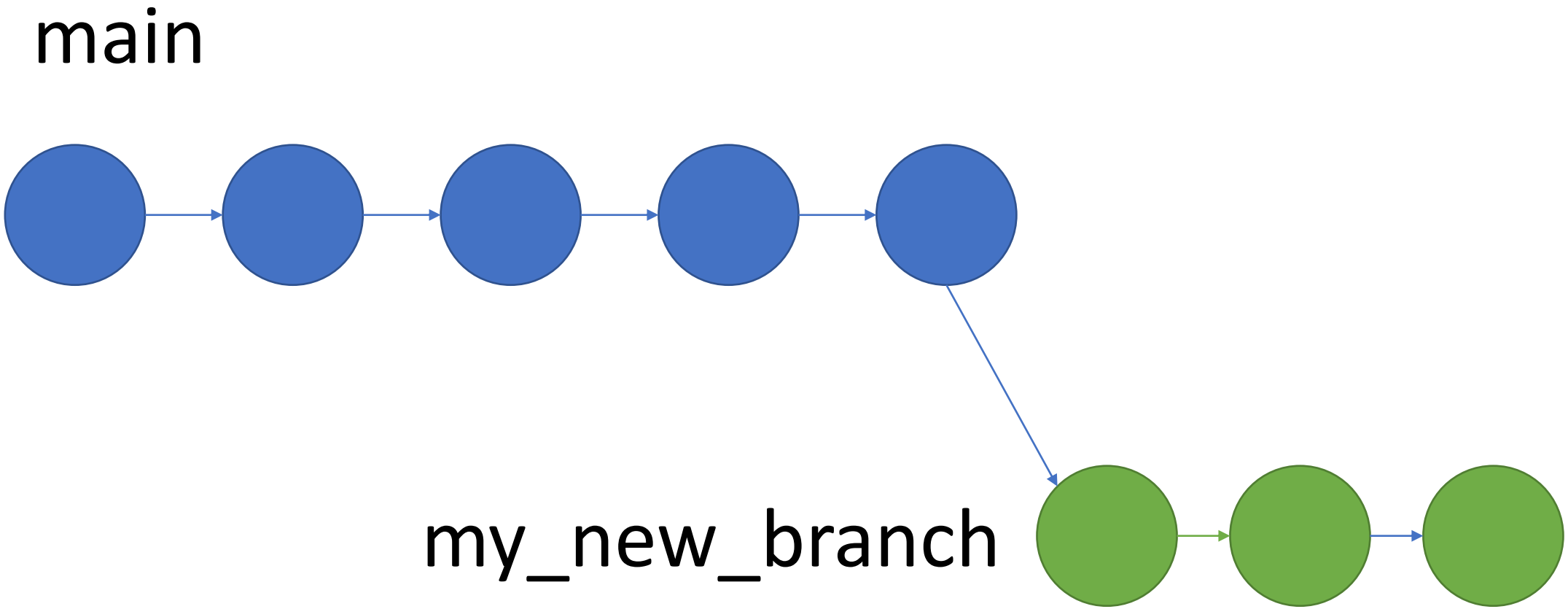
Dev

Create a new branch

1. Open a terminal prompt and run

```
git status
git branch -v
git checkout –b my_new_branch
git branch -v
git status
```

2. Also explore how this looks in your IDE


main
my_new_branch

Make some changes and commit

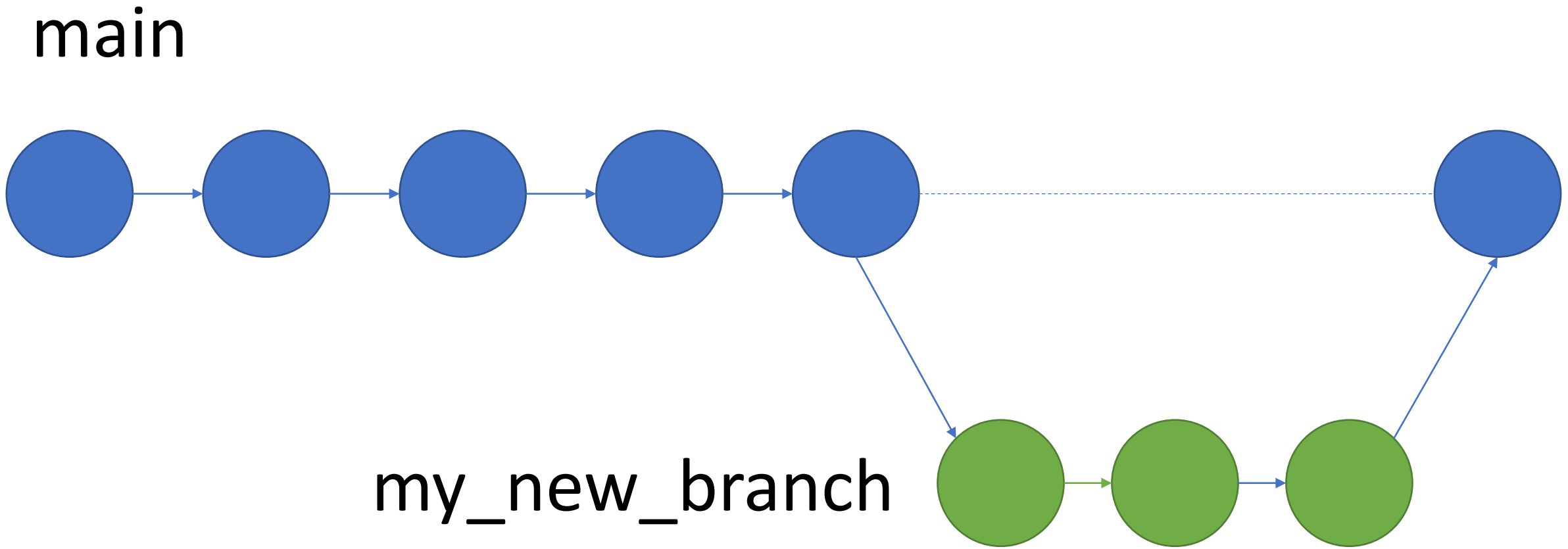
1. Make some changes to your file
2. Open a terminal prompt and run

```
git status
git add my_file.py
git commit –m “Testing out this thing”
```

3. Also explore how this looks in your IDE
4. Now run this

```
git checkout main
```

5. You’re looking at the other branch!

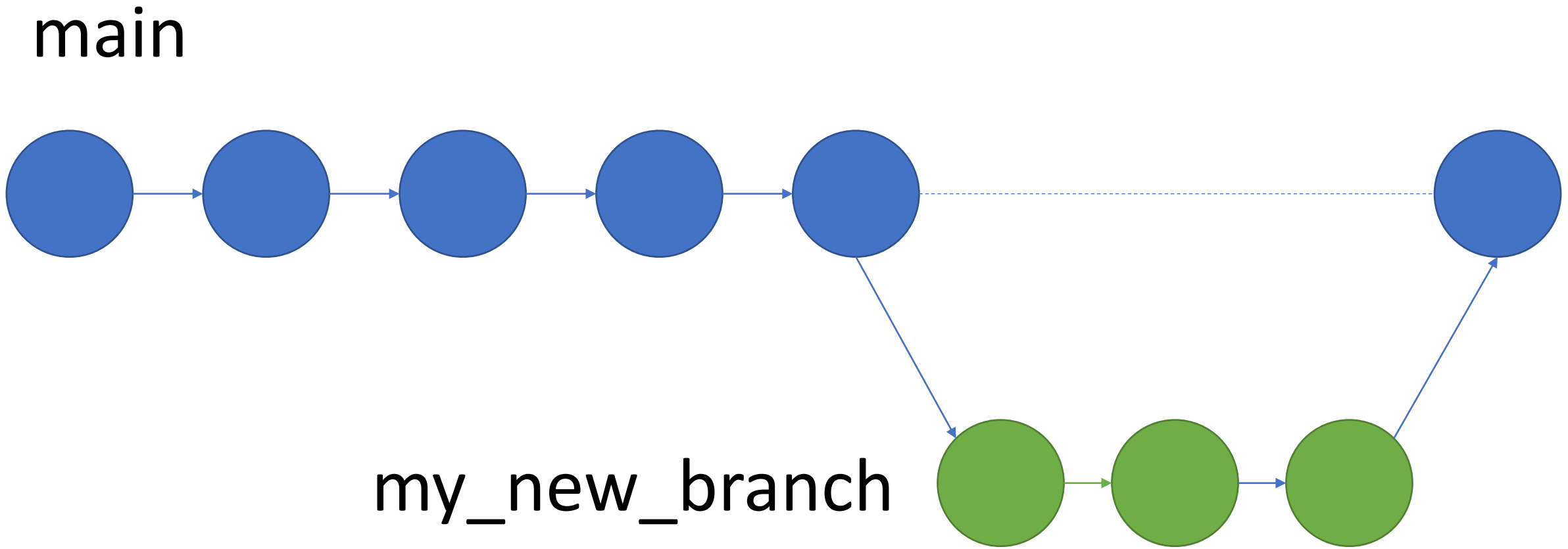

main
my_new_branch

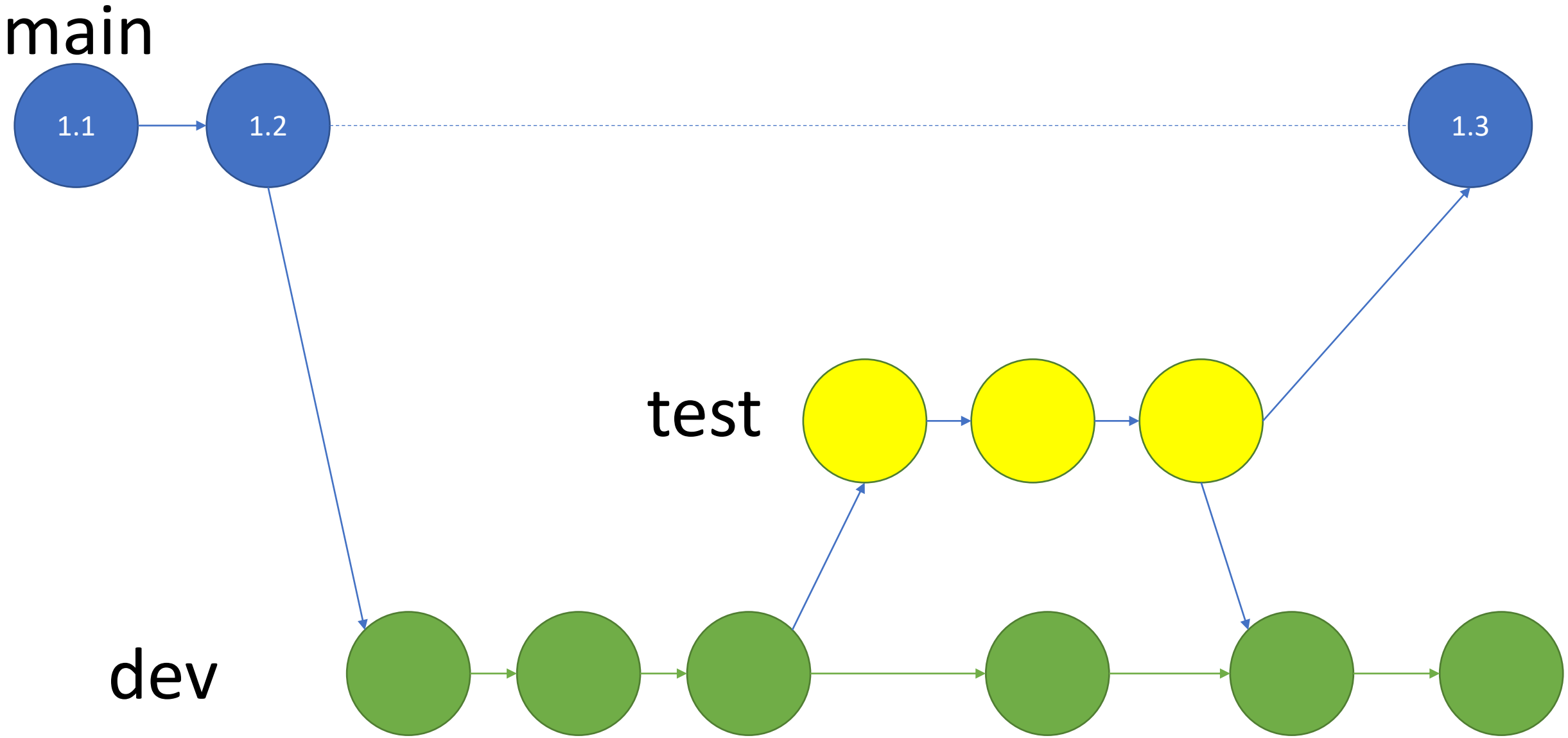
Merge development and main

1. Open a terminal prompt and run

```
git status
git checkout main
git merge my_new_branch
git branch -v
git status
```

2. Also explore how this looks in your IDE


main
my_new_branch


main
1.1
1.2
1.3
test
dev

Making a pull request

Steps

1. Fork the repository
2. Edit and commit
3. Submit the pull request