Code Versioning and Quality

Robert Clements

MSDS Program

University of San Francisco

What to Expect

- Goal: to learn branching strategies, and the importance of code styling and quality.

- How: in the lab we will explore the use of git branches along with common python libraries for checking code quality.

Code Versioning and Branching

- We already know how to use git for version control
  - And Github as a central code repository (along with other nice features).

- Branching is how you might collaborate with other team members on the same codebase, without stepping on each other's toes
  - Many branching strategies, will be team-dependent.
  - Necessary when collaborating, not when working solo.

Branching

```
# script for doing stuff
```

test.py

main

Branching

```
# script for doing stuff
```

test.py

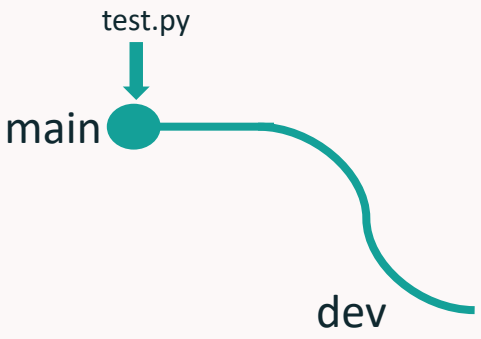
main

```
git commit –m ‘first commit’
```

Branching

script for doing stuff

```
git checkout -b dev
```



Branching

```
# script for doing stuff
```

test.py


```
# script for doing stuff

import pandas as pd
```

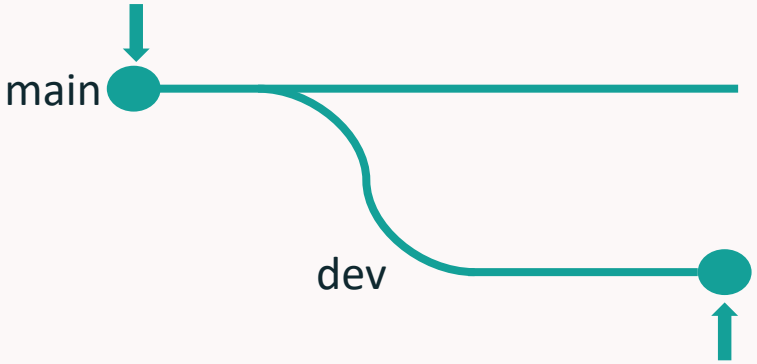
test.py

Branching

```
# script for doing stuff
```

test.py

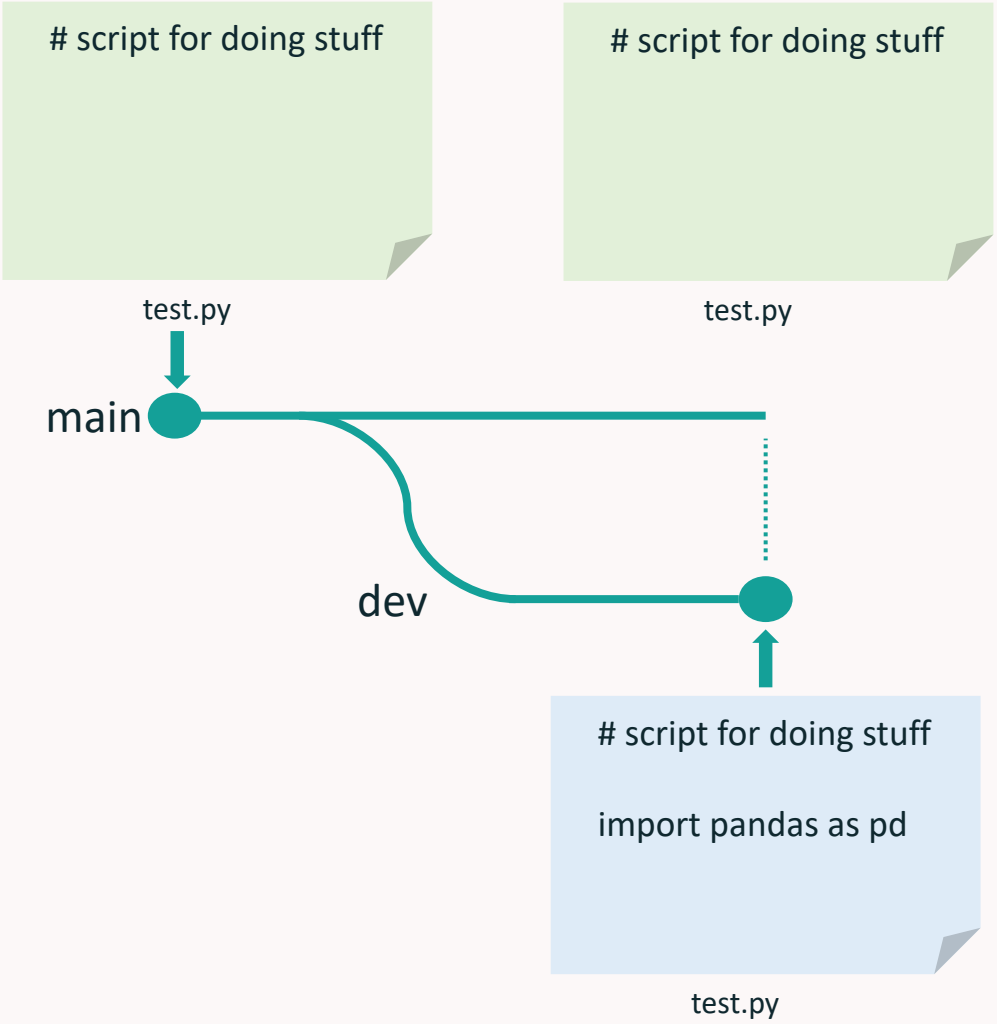
```
git commit –m ‘added library’
```



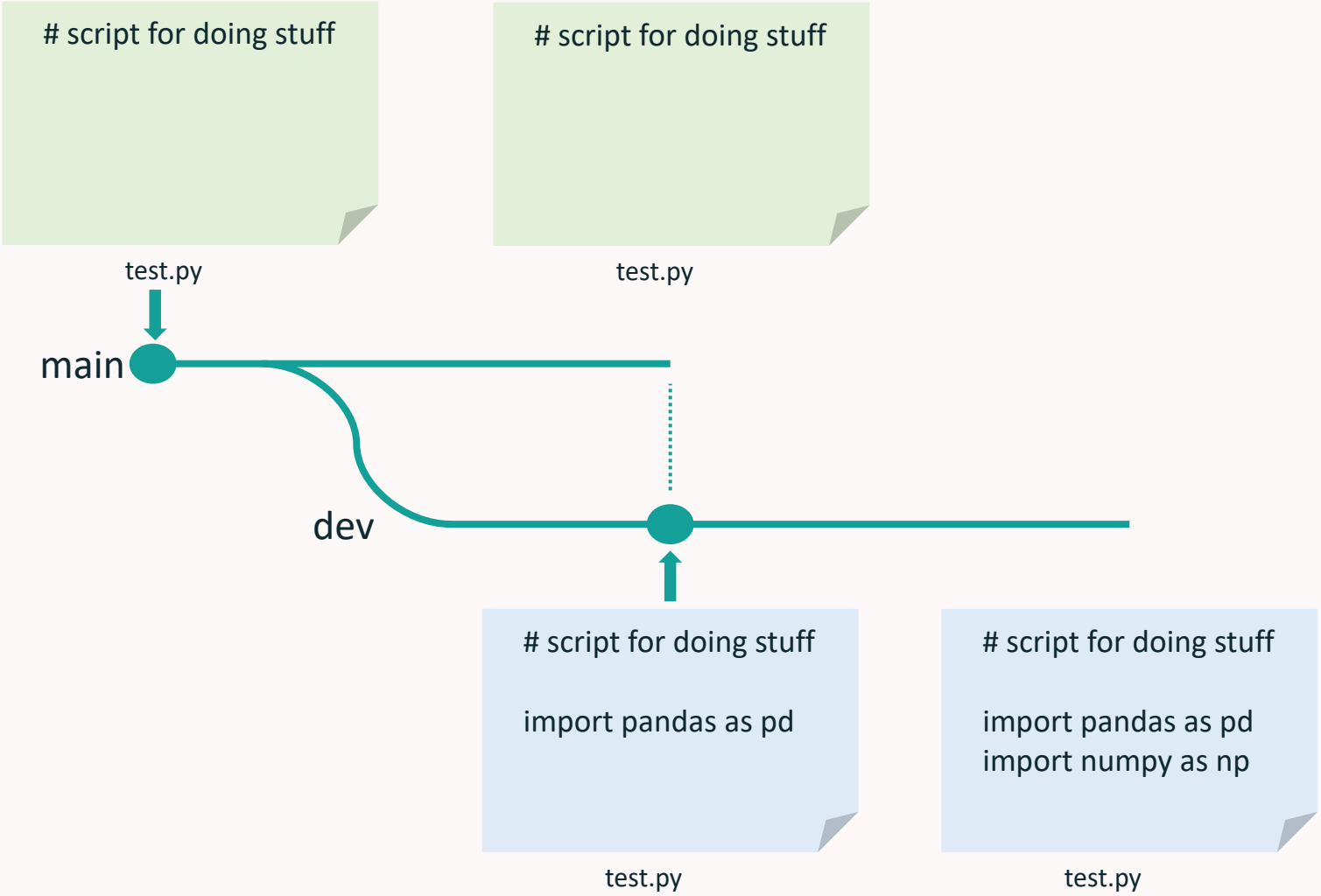

```
# script for doing stuff

import pandas as pd
```

test.py

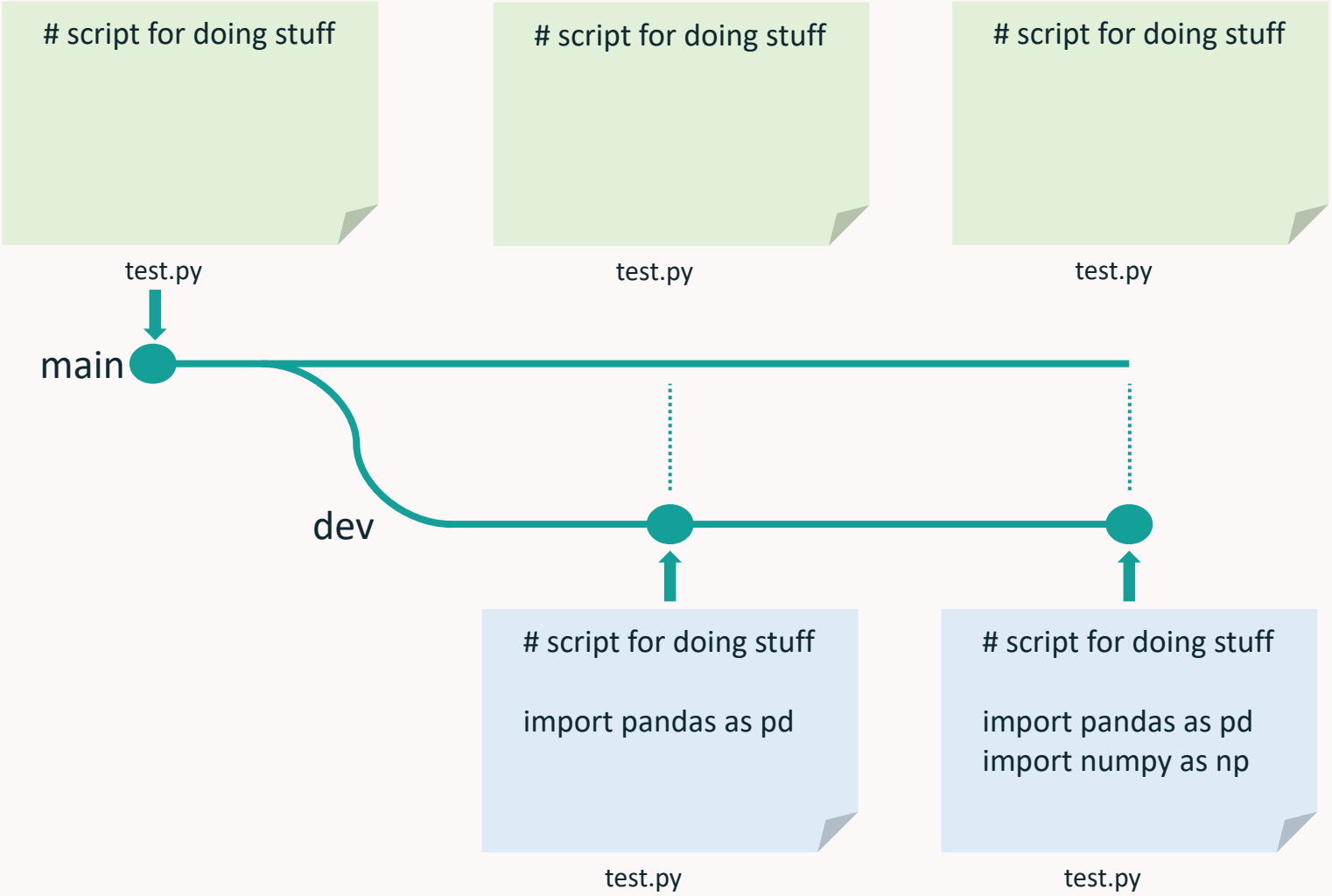
Branching


script for doing stuff
test.py
script for doing stuff
test.py
main
dev
# script for doing stuff
import pandas as pd
test.py

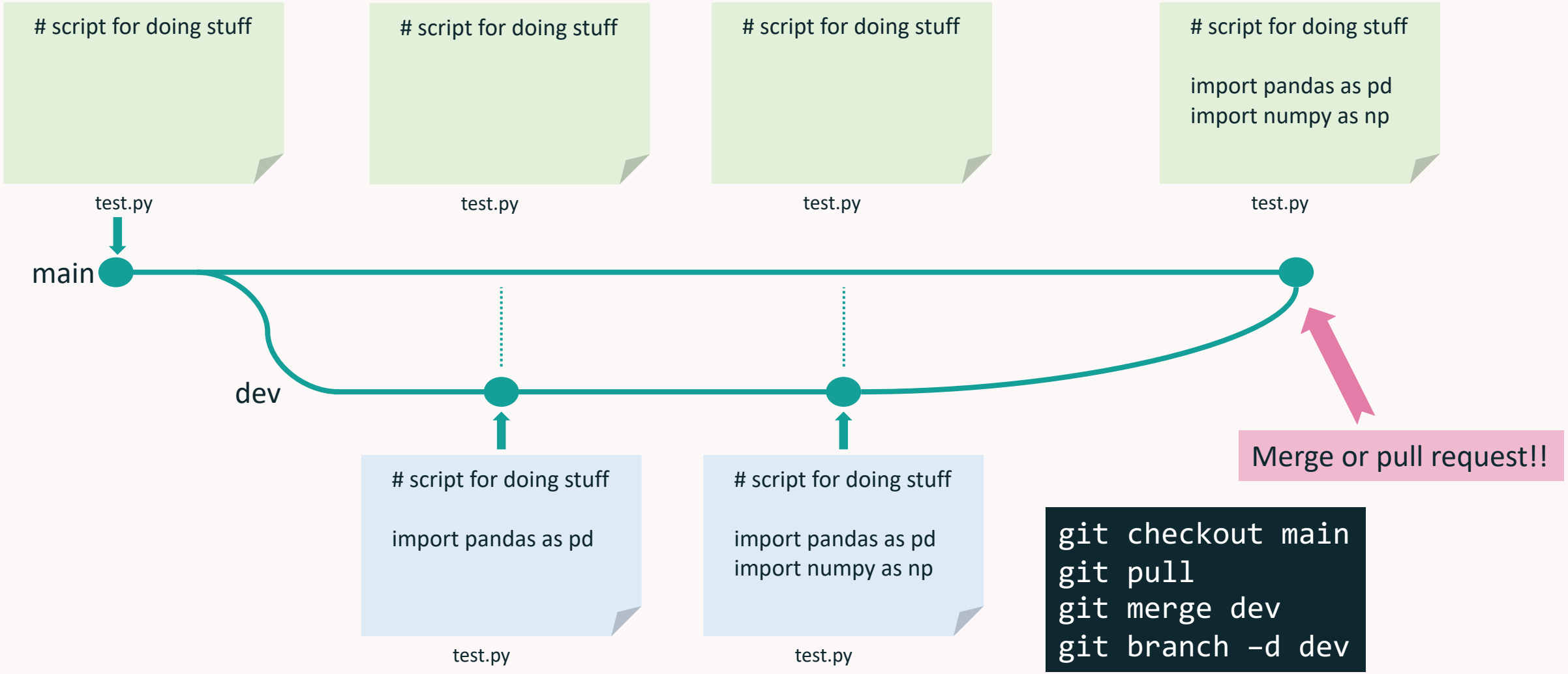
Branching


script for doing stuff
test.py
script for doing stuff
test.py
main
dev
# script for doing stuff
import pandas as pd
test.py
# script for doing stuff
import pandas as pd
import numpy as np
test.py

Branching

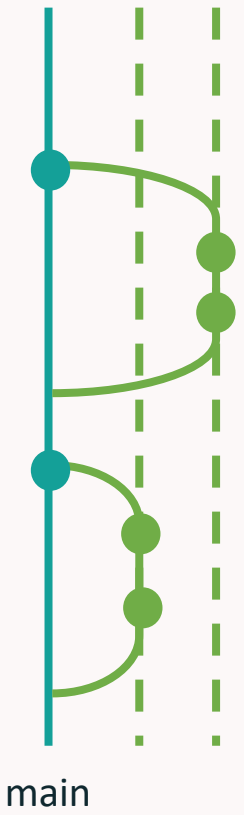

script for doing stuff
test.py
script for doing stuff
test.py
script for doing stuff
test.py
main
dev
# script for doing stuff
import pandas as pd
test.py
# script for doing stuff
import pandas as pd
import numpy as np
test.py

Branching



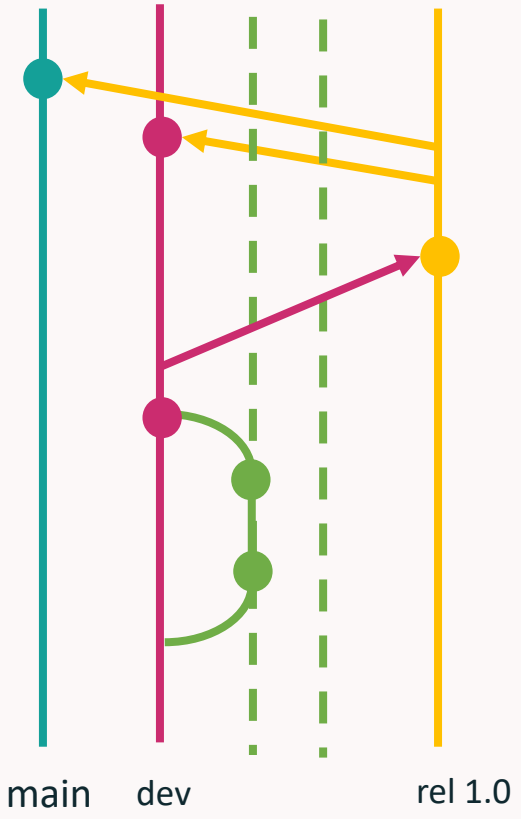

```
git checkout main
git pull
git merge dev
git branch -d dev
```

Branching Strategies



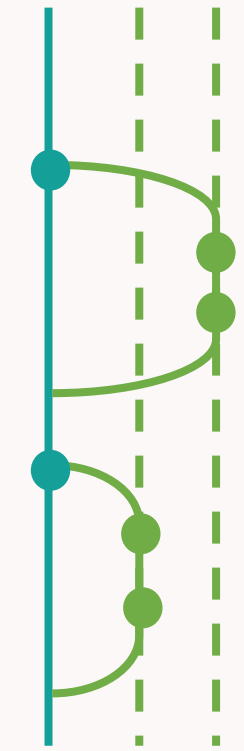
Github Flow

Trunk-based Development



GitFlow

Branching Strategies



- Feature branches off the main branch
  - You, as the DS, may work off of your own branch
- Merge or use pull request to merge to main branch
- Main branch is the “production” branch
- Bug can be introduced into production code since there is no separate “development” or “release” branch

Github Flow

Trunk-based Development

GitFlow

Branching Strategies

- No branches
- Frequent pushes to main branch (trunk)
- True CI/CD
- Only for very experienced devs

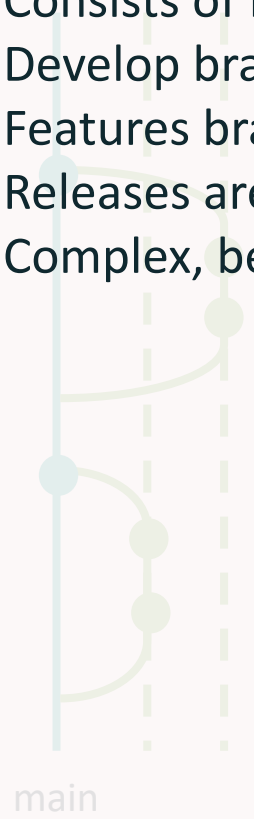
Github Flow

Trunk-based Development

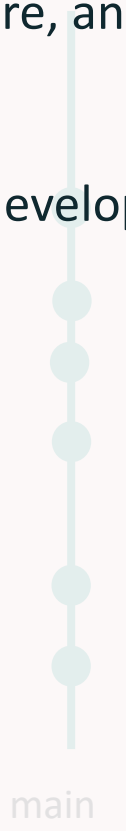
GitFlow

Branching Strategies

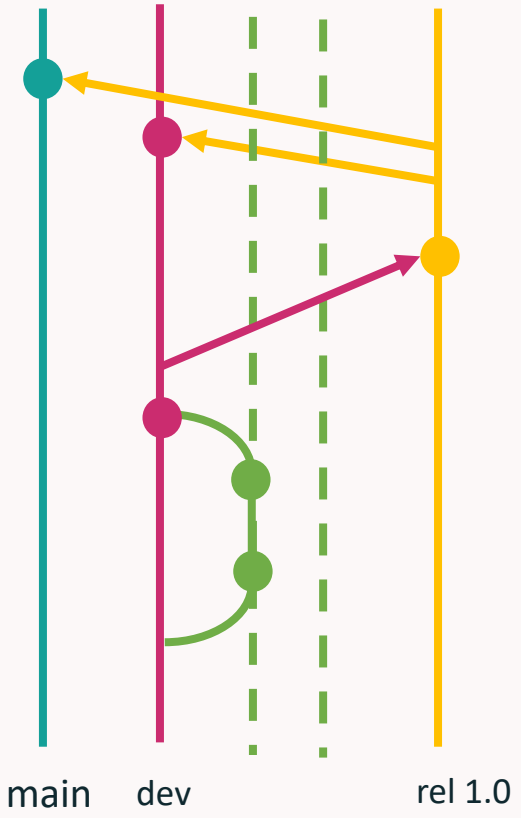
- Consists of Main, Develop, Release, Feature, and Hotfix branches
- Develop branches off of Main
- Features branch off of develop
- Releases are prepared and merged with develop and main
- Complex, best for very large teams



Github Flow



Trunk-based Development



GitFlow

Branching Demo

Code Styling

- Code styling is team-dependent
  - (Python Enhancement Proposal) PEP 8 is commonly used

```
# Correct:
spam(ham[1], {eggs: 2})
```

```
# Wrong:
spam( ham[ 1 ], { eggs: 2 } )
```

Screenshots from https://peps.python.org/pep-0008/

```
# Correct:

# Aligned with opening delimiter.
foo = long_function_name(var_one, var_two,
                         var_three, var_four)

# Add 4 spaces (an extra level of indentation) to distinguish arguments from the rest.
def long_function_name(
        var_one, var_two, var_three,
        var_four):
    print(var_one)

# Hanging indents should add a level.
foo = long_function_name(
    var_one, var_two,
    var_three, var_four)
```

```
# Wrong:

# Arguments on first line forbidden when not using vertical alignment.
foo = long_function_name(var_one, var_two,
    var_three, var_four)

# Further indentation required as indentation is not distinguishable.
def long_function_name(
    var_one, var_two, var_three,
    var_four):
    print(var_one)
```

Linting

- Linting: check code for errors (both programmatic and style)
  - There are many linters to choose from: pylint, flake8, pycodestyle, and others
  - Use linters within your IDE (pycharm, VSCode)
  - Configure with a pyproject.toml file

```
pip install pylint

pylint file.py
```

pyproject.toml

```
[tool.pylint.messages_control]
disable = [
"missing-final-newline",
"missing-function-docstring”,
. . .
]
```

Code Formatting

- Formatters: will format your code for you
  - isort – sorts your imports
  - Autopep8 – PEP 8 auto-formatting
  - Black – opinionated formatter

```
pip install black isort

black file.py

isort file.py
```

pyproject.toml

```
[tool.pylint.messages_control]
disable = [
"missing-final-newline",
"missing-function-docstring”,
. . .
]

[tool.black]
line-length = 100
target-version = ['py39']
skip-string-normalization = true

[tool.isort]
length_sort = true
```

Unit Tests

- Unit tests are a way of testing small independent pieces/units of code
  - You may have seen these in previous courses
  - Differ from regression and integration tests
- As a DS you may want to test:
  - Data types
  - Data exists
  - Data in proper range
  - Edge cases for functions (missing values, wrong types, etc.)
- pytest is commonly used
- Tools like deepchecks might be a good alternative to writing your own tests
- More on this later when we talk about CI/CD/CT

Linting and Formatting Demo

Branching and Code Quality Lab