



arm

TDT4102

Arm-ing yourself for
exceptional circumstances

Einar Johan Trøan Sømåen

2023-03-20

Einar Johan Trøan Sømåen

- Siv. Ing. Datateknikk (2015)
- Undass/Vitass i TDT4102 (5 år)
 - Videreutviklet øvingsopplegget.
- Principal Engineer & Team Lead Arm Norway A/S (siden 2015)
 - Verifikasjon og testing av Mali GPUer.

Arm

- Hardware «Intellectual Property» selskap
- Spesialisert på lav-energi
 - Mobiltelefoner/tablets
 - Mikrokontrollere
 - Datasentre
- Arm Norway
 - Mali/Immortalis grafikkprosessorer
 - Verdens mest solgte grafikkprosessor
 - Soft- og hardwareutvikling

Jobbe hos Arm?

- Ingen sommerjobber ledig i år, men kanskje neste år?
 - peter.salvesen@arm.com
- Stillinger lyses ut her:
 - careers.arm.com

Hva gjør jeg hos Arm?

— «Toplevel verifikasjon»

- Teste at GPUen fungerer «i sin helhet»

— Teste?

- Utvikle systemer for testing
- Utvikle input til testingen
- Planlegge hvilke tester som trengs

arm

Testing

Hva, hvorfor, hvordan?

Hva er testing?

— Kode som sjekker korrektheten til annen kode

Hvorfor teste?

- Oppdage bugs
 - Mye billigere å fikse bugs tidlig
- Bekrefte at man har oppfylt krav
 - Har man spesifikke krav til programmet må man kunne vise til at de er dekket
- Forstå problemet
 - Konkretiser hva man skal oppnå
 - Hjelper på problemløsingen

Hvordan teste?

—|— Positiv testing

- Gyldig input

—|— Negativ testing

- Ugyldig input (teste feilhåndtering)
- Sikkerhet

Hvordan teste (positiv testing)?

— Håndskrevne tester

- Hjelper godt på tenkingen
- «Testdreven utvikling»
- Tester bare det vi selv kommer på

Håndskreven test: testGetSurname() (1)

— Start med testen

- Hjelper godt på tenkingen
- «Testdreven utvikling»

```
void testGetSurname() {  
    assert(  
        stringIsEqual(getSurname("James Bond"), "Bond")  
    );  
}
```

Håndskreven test: testGetSurname() (2)

```
bool stringIsEqual(string value, string expected) {  
    if (value != expected) {  
        cout << "'" << value << "' != '" << expected << "'" << endl;  
        return false;  
    }  
    return true;  
}  
  
void testGetSurname() {  
    assert(  
        stringIsEqual(getSurname("James Bond"), "Bond")  
    );  
}
```


Håndskreven test: testGetSurname() (3)

```
string getSurname(string fullName) {  
    auto index = fullName.find_last_of(" ");  
    return fullName.substr(index);  
}
```

Håndskreven test: testGetSurname() (4)

```
string getSurname(string fullName) {  
    auto index = fullName.find_last_of(" ");  
    return fullName.substr(index);  
}
```

' Bond' != 'Bond'

**Assertion failed: (stringIsEqual(getSurname("James Bond"), "Bond")), function
testGetSurname, file main.cpp, line 88.**

Program ended with exit code: 9

Håndskreven test: testGetSurname() (5)

— Fant en feil (vi antok noe om find_last_of)

- Korrekt versjon nedenfor

```
string getSurname(string fullName) {  
    auto index = fullName.find_last_of(" ") + 1;  
    return fullName.substr(index);  
}
```

Håndskreven test: testGetSurname() (6)

- Behold testen så reintroduserer vi ikke den feilen igjen.
- Så legger man til flere tester, f.eks
 - «Madonna»
 - «Leonardo da Vinci»

Hvordan teste (positiv testing)?

— Tilfeldig (random) tester

- Hvis det er lett å generere input/sjekk resultat
- Kan sjekke ting vi ikke selv kommer på
- Reproduserbarhet blir en utfordring (seed)
- Både gyldig/ugyldig input

Random testing (positiv test) (1)

— Teste sorteringsfunksjon

- Kan generere tilfeldig input
- Kan sjekke: Økende indeks skal ha \geq i verdi

```
vector<int> ourSortFunction(vector<int> vec) {  
    std::sort(vec.begin(), vec.end());  
    return vec;  
}
```

Random testing (positiv test) (2)

```
vector<int> generateRandomVector(int numElements) {  
    vector<int> testData;  
    for (int i = 0; i < numElements; i++) {  
        testData.push_back(rand());  
    }  
    return testData;  
}
```

Random testing (positiv test) (3)

```
void checkSorted(vector<int> &vec) {  
    for (int i = 1; i < vec.size(); i++) {  
        assert(vec[i] >= vec[i - 1]);  
    }  
}
```


Random testing (positiv test) (4)

```
void testOurSortFunction() {  
    // Generate input  
    int numElements = rand() % 100;  
    vector<int> testData = generateRandomVector(numElements);  
  
    auto sorted = ourSortFunction(testData); // auto det samme som vector<int>  
    checkSorted(sorted);  
}
```

Random testing (positiv test) (5)

```
int main(int argc, const char * argv[]) {  
    srand(42);  
    for (int i = 0; i < 1000; i++) {  
        testOurSortFunction();  
    }  
}
```

Random testing (positiv test) (6)

```
vector<int> generateRandomVector(int numElements) {
    vector<int> testData;
    for (int i = 0; i < numElements; i++) {
        testData.push_back(rand());
    }
    return testData;
}

void checkSorted(vector<int> &vec) {
    for (int i = 1; i < vec.size(); i++) {
        assert(vec[i] >= vec[i - 1]);
    }
}

void testOurSortFunction() {
    // Generate input
    int numElements = rand() % 100;
    vector<int> testData = generateRandomVector(numElements);

    auto sorted = ourSortFunction(testData); // auto det samme som vector<int>
    checkSorted(sorted);
}
```

```
int main(int argc, const char * argv[]) {
    srand(42);
    for (int i = 0; i < 1000; i++) {
        testOurSortFunction();
    }
}
```

Hva sammenligner vi?

- Sammenligne med referanseimplementasjon
 - Hvis man har en annen implementasjon, sjekk mot den
 - Eksempel fra min masteroppgave: Prototype i C++ testet mot VHDL-kode
 - Annen implementasjon (eldre/treigere)

Negativ testing?

— Ugyldig input

- Definisjonsspørsmål, må definere hva resultatet blir selv for ugyldig input
- `getSurname("1")`

— Kan i mange tilfeller definere resultater der verdirommet har et meningsfylt svar

- `getSurname("1") -> "1"` // Vi kan definere at svaret alltid er siste «ord»

Negativ testing?

— Ugyldig input

- Definisjonsspørsmål, må definere hva resultatet blir selv for ugyldig input
- `getAreaOfCircleWithRadius(-2)`

— Kan i mange tilfeller definere resultater der verdirommet har et meningsfylt svar

- `getAreaOfCircleWithRadius(-2) -> 0` // Vi kan definere at vi runder radius «opp» til 0
- `getAreaOfCircleWithRadius(-2) -> -1` // Vi kan definere at ugyldig svar alltid gir -1
- `getAreaOfCircleWithRadius(-2) -> 4π` // $-4 * -4 * \pi$
- `getAreaOfCircleWithRadius(-2) -> -4π` // Hva nå enn negativt areal betyr?

Negativ testing?

— Ugyldig input

- Definisjonsspørsmål, må definere hva resultatet blir selv for ugyldig input
- `divideNumberBy(1, 0)`

— Kan i mange tilfeller definere resultater der verdirommet har et meningsfylt svar

- `divideNumberBy(1, 0)` // Kan ikke definere divisjon på 0 til 0, og ingen plass i verdirom
- Må ha en annen måte å signalisere denne feilen på

Negativ testing?

— Ugyldig input

- Definisjonsspørsmål, må definere hva resultatet blir selv for ugyldig input
- `readThirdLineOfFile("finnes_ikke.txt")`

— Kan i mange tilfeller definere resultater der verdirommet har et meningsfylt svar

- `readThirdLineOfFile("finnes_ikke.txt")` // Har ikke ledig plass i verdirom, tredje linje kunne vært tom.
- Trenger en annen måte å håndtere dette på

arm

Exceptions

Feil? Vi skriver da vel ikke feil?

— Vi gjør alltid antagelser

- Fila man prøver å åpne finnes
- Verdiene man får inn er gyldige $(v-1)/0$
- Brukerinput
- Nettverkstilkobling

— I en perfekt verden kunne vi hatt kontroll på alt dette

- I praksis må vi ta høyde for det

Feilhåndtering

— Det finnes mange måter for en funksjon å si ifra at noe gikk feil:

- Sette av deler av verdirommet (return nullptr, evt return -1)
- Returnere mer enn en verdi:
 - `bool foo(int &result, int input);`
 - `std::pair<int, bool> bar();`
 - `glGetError()`

— Men:

- Hva om man blander forskjellige måter å varsle om feil?
- Hva om man trenger å håndtere «dype» feil?

— Løsning: Exceptions

Unntak (Exceptions)

- Lar oss si ifra at noe «unntaksvis» har skjedd
- Avbryter ryddig, lar oss eventuelt håndtere problemet
- Lar oss fokusere på normaltilfellet

Eksempel: Unntak i STL

- `std::vector` har to måter å lese elementer på:
 - `std::vector::operator[]` - kaster ikke unntak
 - `std::vector::at[]` - kaster unntak

Eksempel: Unntak i STL

— `std::vector` har to måter å lese elementer på:

- `std::vector::operator[]` - kaster ikke unntak
- `std::vector::at[]` - kaster unntak

```
std::vector<int> test;  
test.push_back(42);  
cout << test[1] << endl; // Farlig, udefinert oppførsel  
cout << test.at(1) << endl;
```

Eksempel: Unntak i STL

— Hva sier egentlig feilmeldinga?

```
std::vector<int> test;  
test.push_back(42);  
cout << test[1] << endl; // Farlig, udefinert oppførsel  
cout << test.at(1) << endl;
```

0

```
libc++abi: terminating with uncaught exception of type std::out_of_range: vector  
terminating with uncaught exception of type std::out_of_range: vector  
(11db)
```


Unntakssyntaks - try-catch

— Try-catch

- Try: «Prøv dette, sikkerhetsnettet finner du under»
- Catch: «Sikkerhetsnett for feil av en viss type»

Unntakssyntaks - try-catch

Try-catch

- Try: «Prøv dette, sikkerhetsnettet finner du under»
- Catch: «Sikkerhetsnett for feil av en viss type»

```
std::vector<int> test;  
try {  
    cout << test.at(1) << endl;  
} catch (std::out_of_range &e) {  
    // Håndter feil  
}
```

Unntakssyntaks - throw

— throw

- Noe gikk galt, kast et unntak

```
int divideNumberBy(int dividend, int divisor) {  
    if (divisor == 0) {  
        throw DivisorIsZeroException();  
    } else {  
        return dividend / divisor;  
    }  
}
```

Unntakssyntaks - kombinert

- NB: Kodeflyt, vi «hopper» ut fra der vi var når vi throw-er, og kommer ikke tilbake
- «Mer kode» vil aldri bli kjørt hvis vi thrower

```
try {  
    if (divisor == 0) {  
        throw DivisorIsZeroException();  
    }  
    // Mer kode  
} catch (DivisorIsZeroException &e) {  
    // Håndter feil  
}
```

arm

Typen unntak

Egne typer unntak

- Kan kaste hva som helst, også klasser vi har definert selv
- Beskrivende navn er nyttig
- Nyttig informasjon («beskrivelse av feil»)
- Arv

Catch av forskjellige typer unntak

- Fanger unntak basert på type
- Sjekkes ovenfra og ned
 - NB: Subtyper før supertyper
- catch(...)
- Kan kaste unntak herifra også

```
try {  
    // Kode  
} catch (std::logic_error &e) {  
    // Håndter logic-error  
} catch (std::exception &e) {  
    // Håndter std::exception  
} catch (...) {  
    // Håndter alt annet  
}
```

Praktisk bruk

- Throw by value
- Catch by reference

std::exception

- Definert i <exception>
- En del vanlige unntakstyper i <stdexcept>
- std::runtime_error
 - std::overflow_error
 - std::underflow_error
- std::logic_error
 - std::domain_error
 - std::out_of_range
 - std::invalid_argument
- what()

Eksempel: Catching på type (1)

```
try {  
    functionThatThrows();  
} catch (std::logic_error &e) {  
    cout << "logic_error: " << e.what() << endl;  
} catch (std::exception &e) {  
    cout << "exception: " << e.what() << endl;  
} catch (...) {  
    cout << "Unknown exception" << endl;  
}
```

Eksempel: Catching på type (2)

```
int functionThatThrows() {  
    throw std::logic_error("Logic_error");  
}  
  
try {  
    functionThatThrows();  
} catch (std::logic_error &e) {  
    cout << "logic_error: " << e.what() << endl;  
} catch (std::exception &e) {  
    cout << "exception: " << e.what() << endl;  
} catch (...) {  
    cout << "Unknown exception" << endl;  
}
```

logic_error: Logic_error

Eksempel: Catching på type (3)

```
int functionThatThrows() {  
    throw std::exception();  
}  
  
try {  
    functionThatThrows();  
} catch (std::logic_error &e) {  
    cout << "logic_error: " << e.what() << endl;  
} catch (std::exception &e) {  
    cout << "exception: " << e.what() << endl;  
} catch (...) {  
    cout << "Unknown exception" << endl;  
}
```

exception: std::exception

Eksempel: Catching på type (4)

```
int functionThatThrows() {  
    throw 42;  
}  
  
try {  
    functionThatThrows();  
} catch (std::logic_error &e) {  
    cout << "logic_error: " << e.what() << endl;  
} catch (std::exception &e) {  
    cout << "exception: " << e.what() << endl;  
} catch (...) {  
    cout << "Unknown exception" << endl;  
}
```

Unknown exception

NB: Rekkefølge

```
int functionThatThrows() {  
    throw logic_error("logic_error");  
}  
  
try {  
    functionThatThrows();  
} catch (std::exception &e) {  
    cout << "exception: " << e.what() << endl;  
} catch (std::logic_error &e) {  
    cout << "logic_error: " << e.what() << endl;  
} catch (...) {  
    cout << "Unknown exception" << endl;  
}
```

exception: logic_error

Destruktører

—|— No throw

arm

Unntakssikkerhet

Hva skjer når en funksjon kaster unntak

— Stack unwind

- Funksjonen «rydder» og avslutter
- Fortsetter «opp kallkjeden» til vi møter en catch
- Eventuelt kræsjer programmet hvis vi ikke møter en catch

Lekkasje



— Foto: Pixabay (CC0)

Hvis vi ikke har unntakssikker kode, så kan vi «lekke ressurser»

- Minne (manuelt allokert)
- Filer
- Nettverkstilkoblinger
- Andre ressurser

Unntakssikker kode

- No except
 - Kaster ikke unntak
- Weak exception guarantee
 - Garanterer at vi er i gyldig tilstand
- Strong exception guarantee
 - Forsøket har ingen effekt ved unntak

Teknikker for unntakssikker kode

- Unngå new/delete
- Resource Acquisition Is Initialization (RAII)
 - Scope-Bound Resource Management
 - «Automatisk opprydding»
- Smart-pekere (shared_ptr/unique_ptr)

Eksempel (Unntakssikkerhet)

```
class NetworkHandle {  
public:  
    ~NetworkHandle() {  
        cout << "Destructor gets called" << endl;  
    }  
};
```

Eksempel (Unntakssikkerhet)

```
void functionWithNewDelete() {  
    NetworkHandle *handle = new NetworkHandle();  
    throw std::runtime_error("Feilmelding");  
    // Kode  
    delete handle;  
}
```



Code will never be executed

Eksempel (Unntakssikkerhet)

```
void functionWithSharedPtr() {  
    shared_ptr<NetworkHandle> handle =  
        make_shared<NetworkHandle>();  
    throw std::runtime_error("Feilmelding");  
    // Kode  
  
    // At end of scope, destructor of shared_ptr gets called  
    // which also deletes the NetworkHandle  
}
```

Eksempel (Unntakssikkerhet)

```
cout << "With new/delete" << endl;
try {
    functionWithNewDelete();
} catch (std::exception &e) {
    cout << e.what() << endl;
}
cout << "-----" << endl;
cout << "With shared_ptr" << endl;
try {
    functionWithSharedPtr();
} catch (std::exception &e) {
    cout << e.what() << endl;
}
```

**With new/delete
Feilmelding**

**With shared_ptr
Destructor gets called
Feilmelding
End**

Program ended with exit code: 0

arm

Oppsummering

Oppsummering (Exceptions)

- Uhånderte unntak avslutter programmer ditt
 - Ikke nødvendigvis ryddig
- Try-Throw-Catch
 - Throw by value
 - Catch by reference
- Subtyper før supertyper (catch ovenfra og ned)
- Destruktører er nothrow

Oppsummering (Testing)

- Test tidlig, test ofte
 - Gjerne skriv testen først
- Positive tester
 - Håndskrevne
 - Random
- Negative tester
 - Ugyldig input
 - Definer oppførsel

arm

Git

Git - versjonskontroll

Git - versjonskontroll

—|— Lar deg spore endringene dine

Git - versjonskontroll

- Lar deg spore endringene dine
 - Og årsaken til de

Git - versjonskontroll

- Lar deg spore endringene dine
 - Og årsaken til de
- Lar deg samarbeide med andre

Git - versjonskontroll

- Lar deg spore endringene dine
 - Og årsaken til de
- Lar deg samarbeide med andre
 - Kod hver for seg, slå sammen endringer

Git - versjonskontroll

- Lar deg spore endringene dine
 - Og årsaken til de
- Lar deg samarbeide med andre
 - Kod hver for seg, slå sammen endringer
- Backup, Github etc.

Git - versjonskontroll

- Lar deg spore endringene dine
 - Og årsaken til de
- Lar deg samarbeide med andre
 - Kod hver for seg, slå sammen endringer
- Backup, Github etc.
- Lar deg spole tilbake når ting går skeis

Git - repository

Git - repository

— Man har koden sin i et «git repository»

Git - repository

- Man har koden sin i et «git repository»
 - Forenklet: En versjonskontrollert mappe

Git - repository

- Man har koden sin i et «git repository»
 - Forenklet: En versjonskontrollert mappe
 - Et slikt repository inneholder en sekvens med «commits»

Git - repository

- Man har koden sin i et «git repository»
 - Forenklet: En versjonskontrollert mappe
 - Et slikt repository inneholder en sekvens med «commits»
 - En commit er et øyeblikksbilde/snapshot av koden din

Git - repository

- Man har koden sin i et «git repository»
 - Forenklet: En versjonskontrollert mappe
 - Et slikt repository inneholder en sekvens med «commits»
 - En commit er et øyeblikksbilde/snapshot av koden din
 - I praksis ser vi på commits i form av hva som endret seg

Git - repository

- Man har koden sin i et «git repository»
- Forenklet: En versjonskontrollert mappe
- Et slikt repository inneholder en sekvens med «commits»
- En commit er et øyeblikksbilde/snapshot av koden din
- I praksis ser vi på commits i form av hva som endret seg

—	<pre>5 int main(int argc, char **argv) { 6- cout << "Hello World" << endl; 7 return 0; 8 }</pre>		<pre>5 int main(int argc, char **argv) { 6+ cout << "Hallo Verden" << endl; 7 return 0; 8 }</pre>
---	--	--	---

Git - commits

Git - commits

— Vi lager commits ved å «stage» endringer

Git - commits

- Vi lager commits ved å «stage» endringer
- I praksis kunne vi commitet alt hver gang, men ofte er det et subsett vi vil commite.

Git - commits

- Vi lager commits ved å «stage» endringer
- I praksis kunne vi commitet alt hver gang, men ofte er det et subsett vi vil commite.
- Dette lar oss se over endringene før vi commiter

Git - commits

- Vi lager commits ved å «stage» endringer
 - I praksis kunne vi commitet alt hver gang, men ofte er det et subsett vi vil commite.
 - Dette lar oss se over endringene før vi commiter
- Demo!

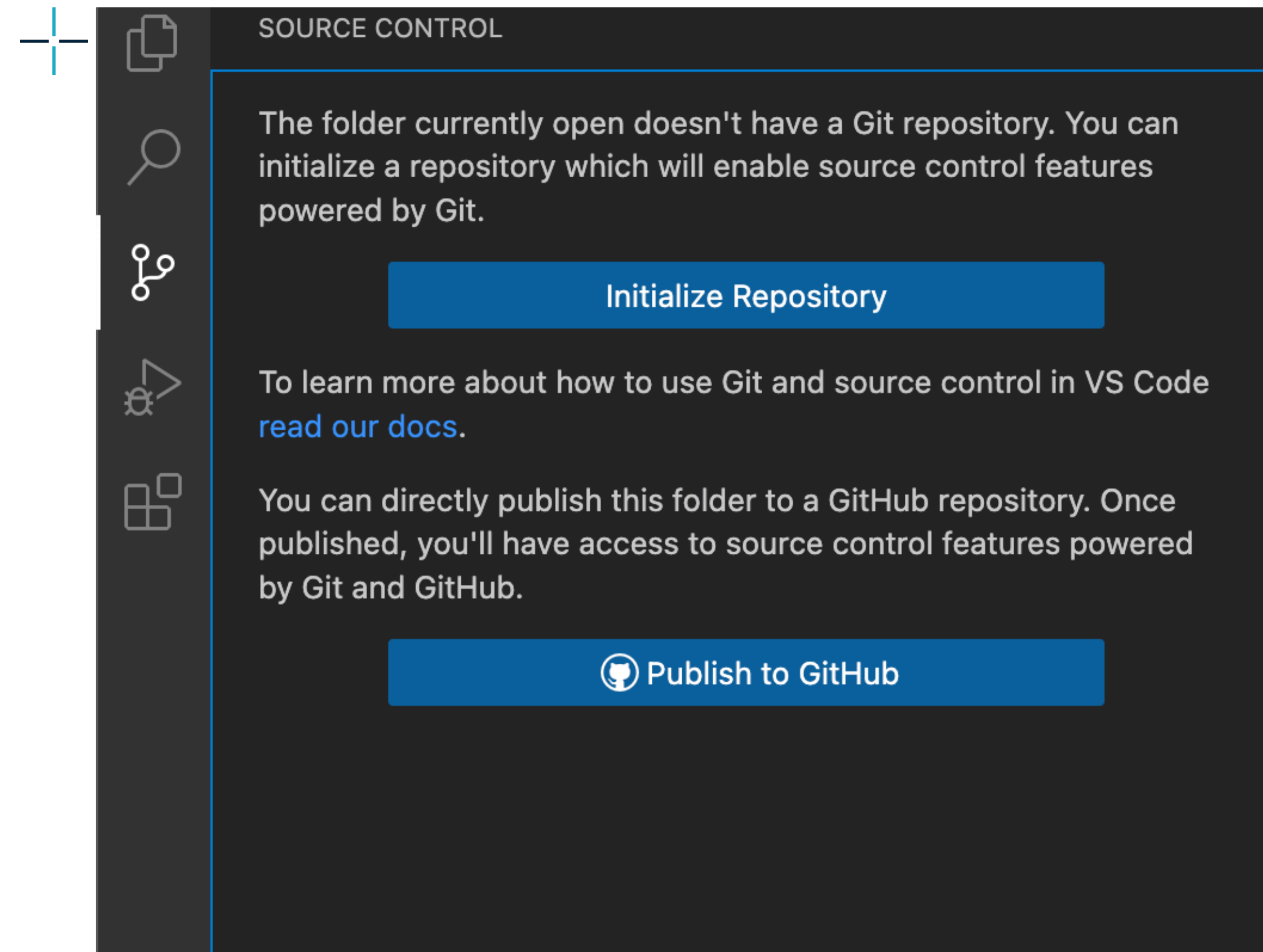
Git - commits (Demo)

Git - commits (Demo)

— Helt generelt eksempel, File -> Open Folder -> Ny Mappe

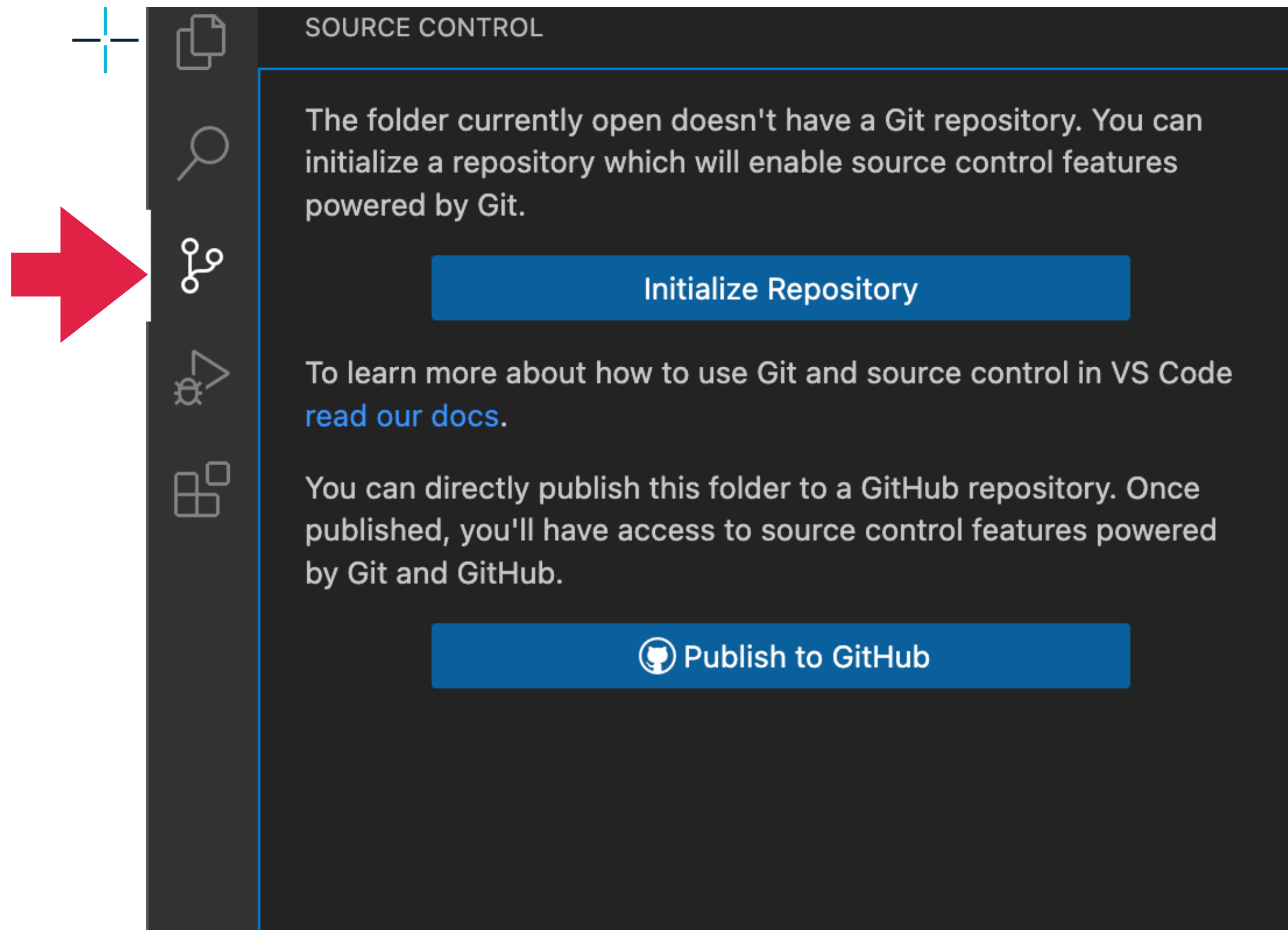
Git - commits (Demo)

+ Helt generelt eksempel, File -> Open Folder -> Ny Mappe



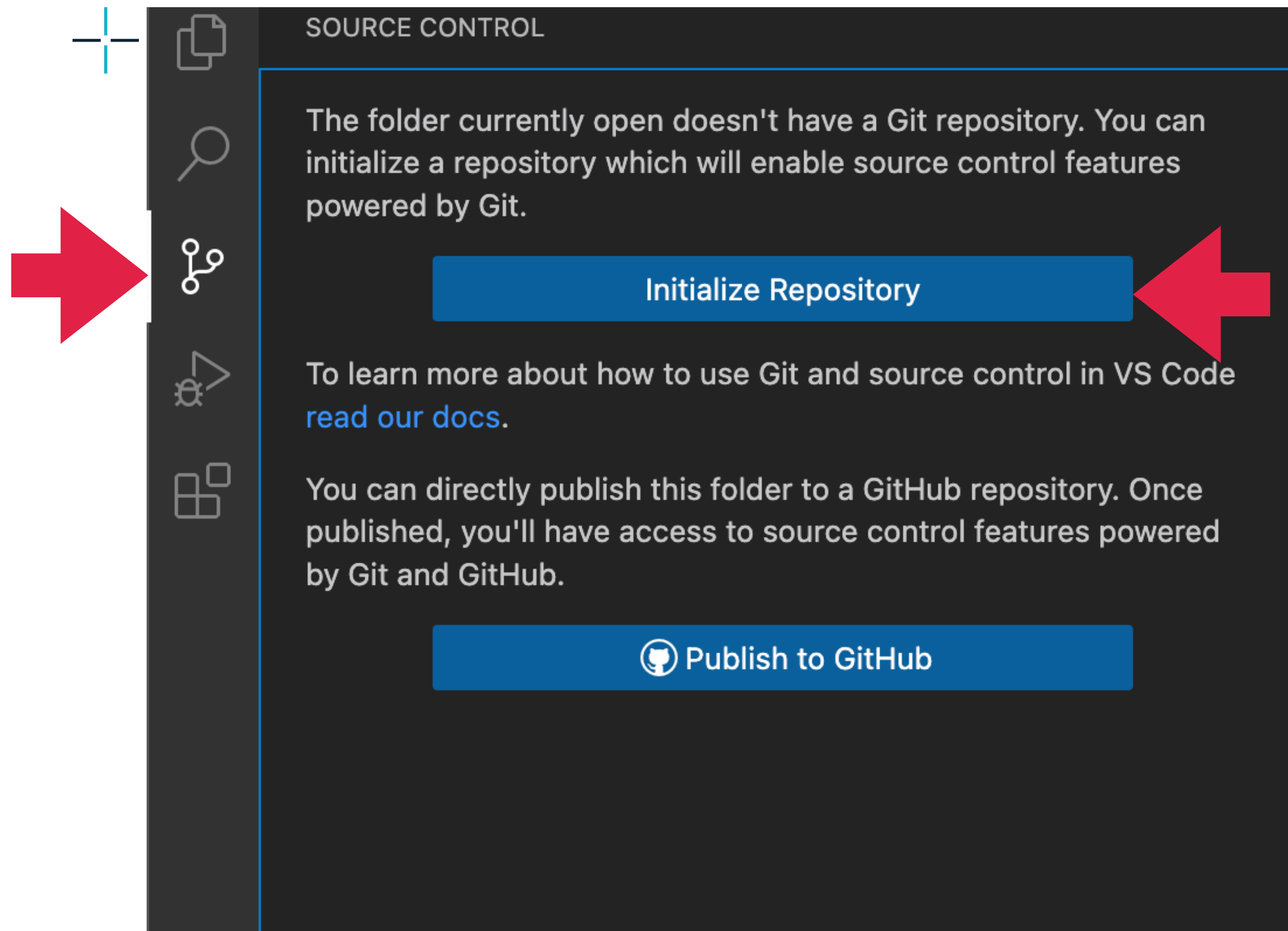
Git - commits (Demo)

+ Helt generelt eksempel, File -> Open Folder -> Ny Mappe



Git - commits (Demo)

+ Helt generelt eksempel, File -> Open Folder -> Ny Mappe



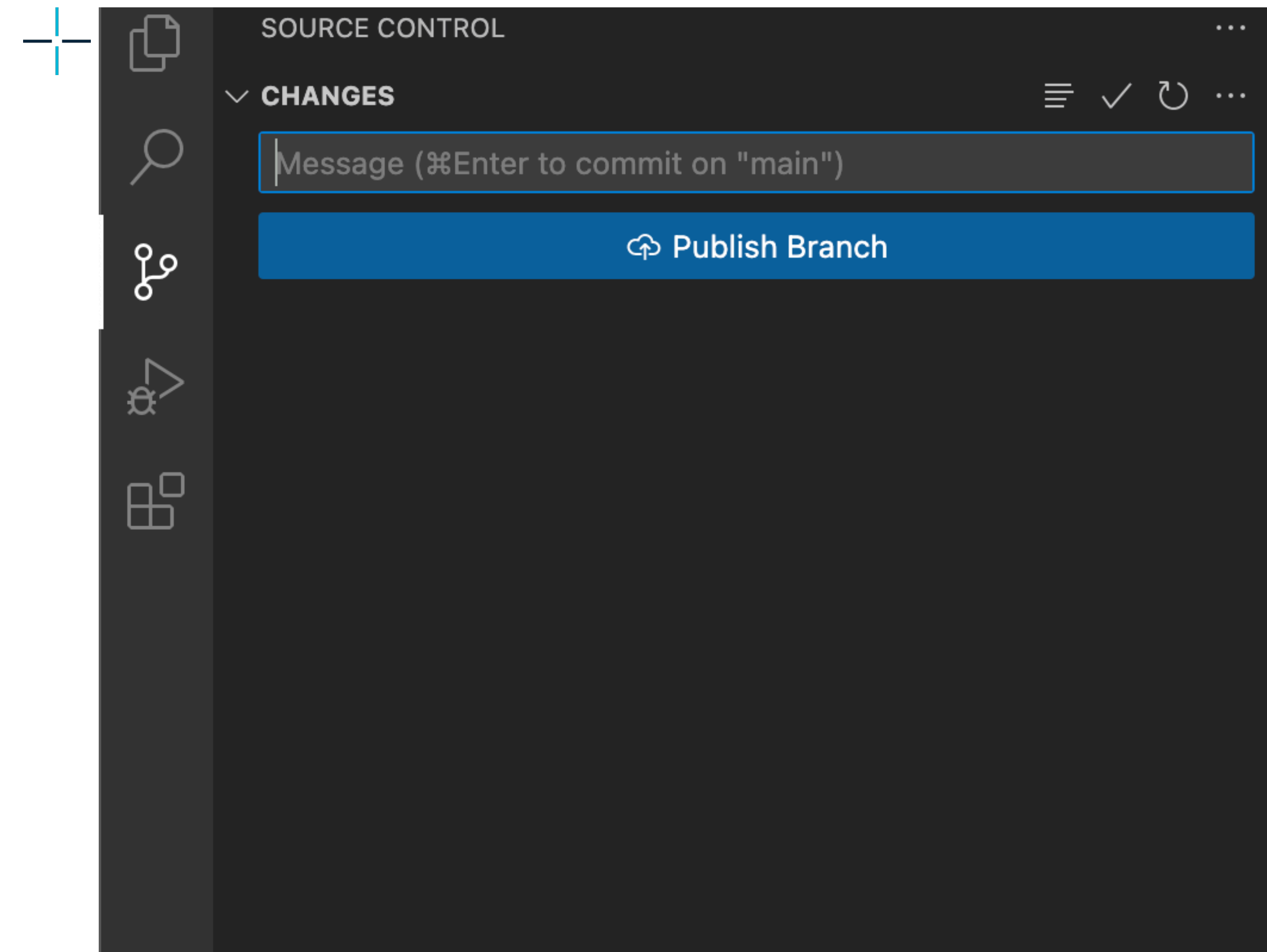
Git - commits (Demo)

Git - commits (Demo)

— Nå har vi et repos, og vi kan begynne å «commite»

Git - commits (Demo)

+ Nå har vi et repos, og vi kan begynne å «commite»



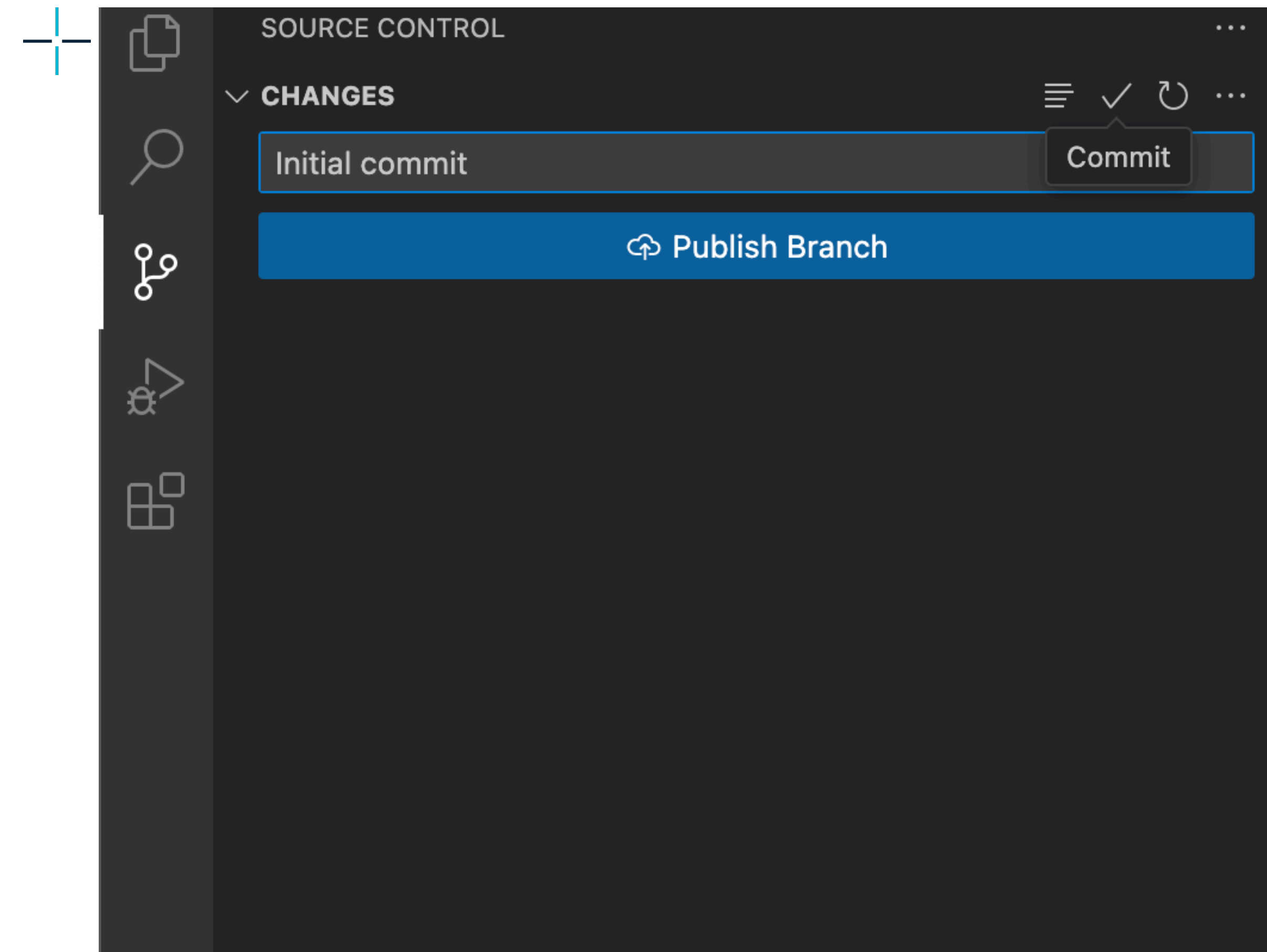
Git - commits (Demo)

Git - commits (Demo)

— Nå har vi et repos, og vi kan begynne å «commite»

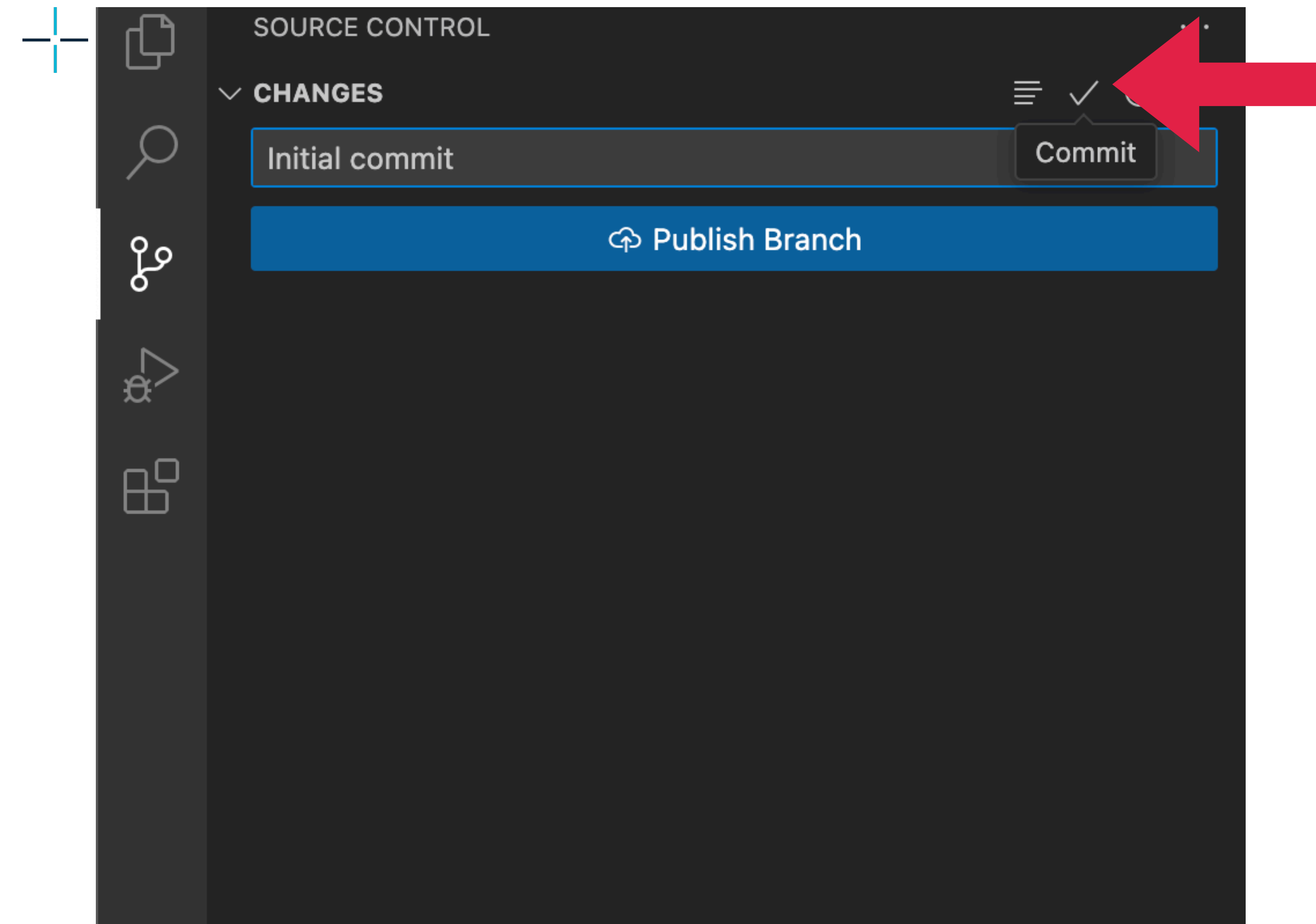
Git - commits (Demo)

+ Nå har vi et repos, og vi kan begynne å «commite»



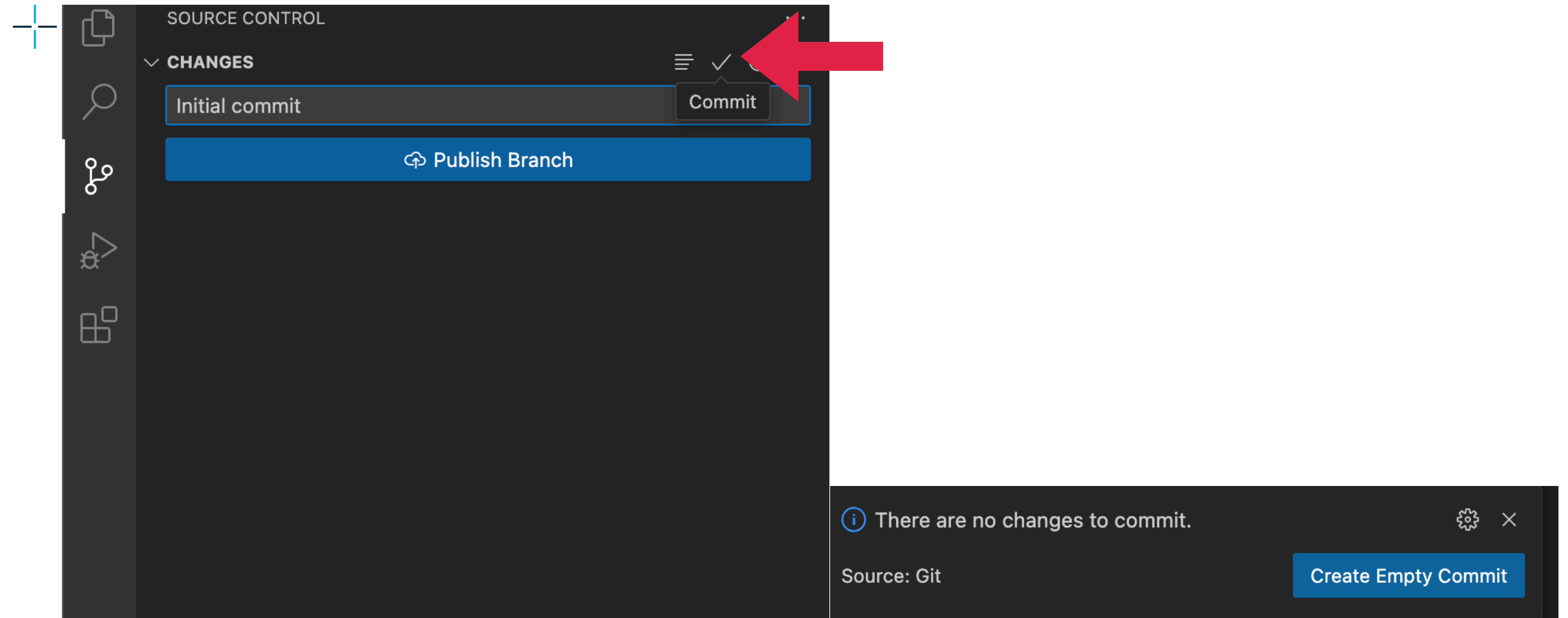
Git - commits (Demo)

+ Nå har vi et repos, og vi kan begynne å «commite»



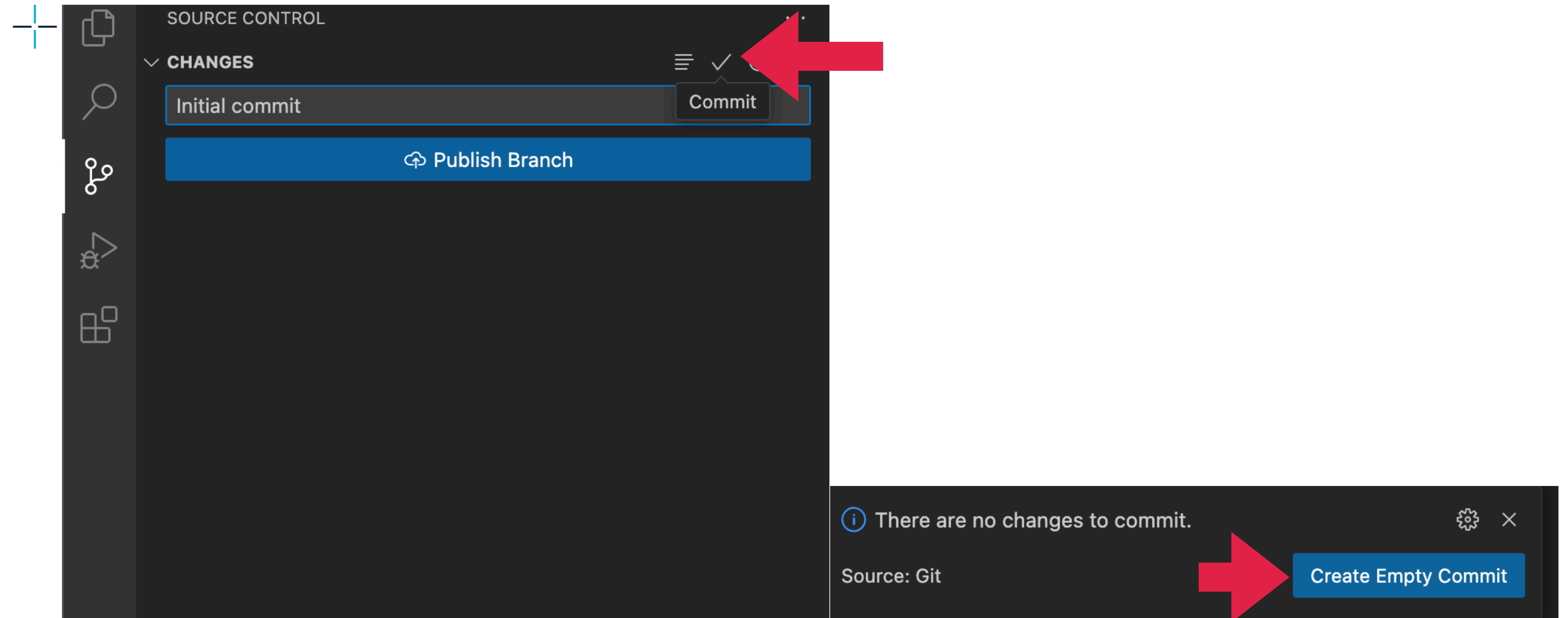
Git - commits (Demo)

+ Nå har vi et repos, og vi kan begynne å «commite»



Git - commits (Demo)

+ Nå har vi et repos, og vi kan begynne å «commite»



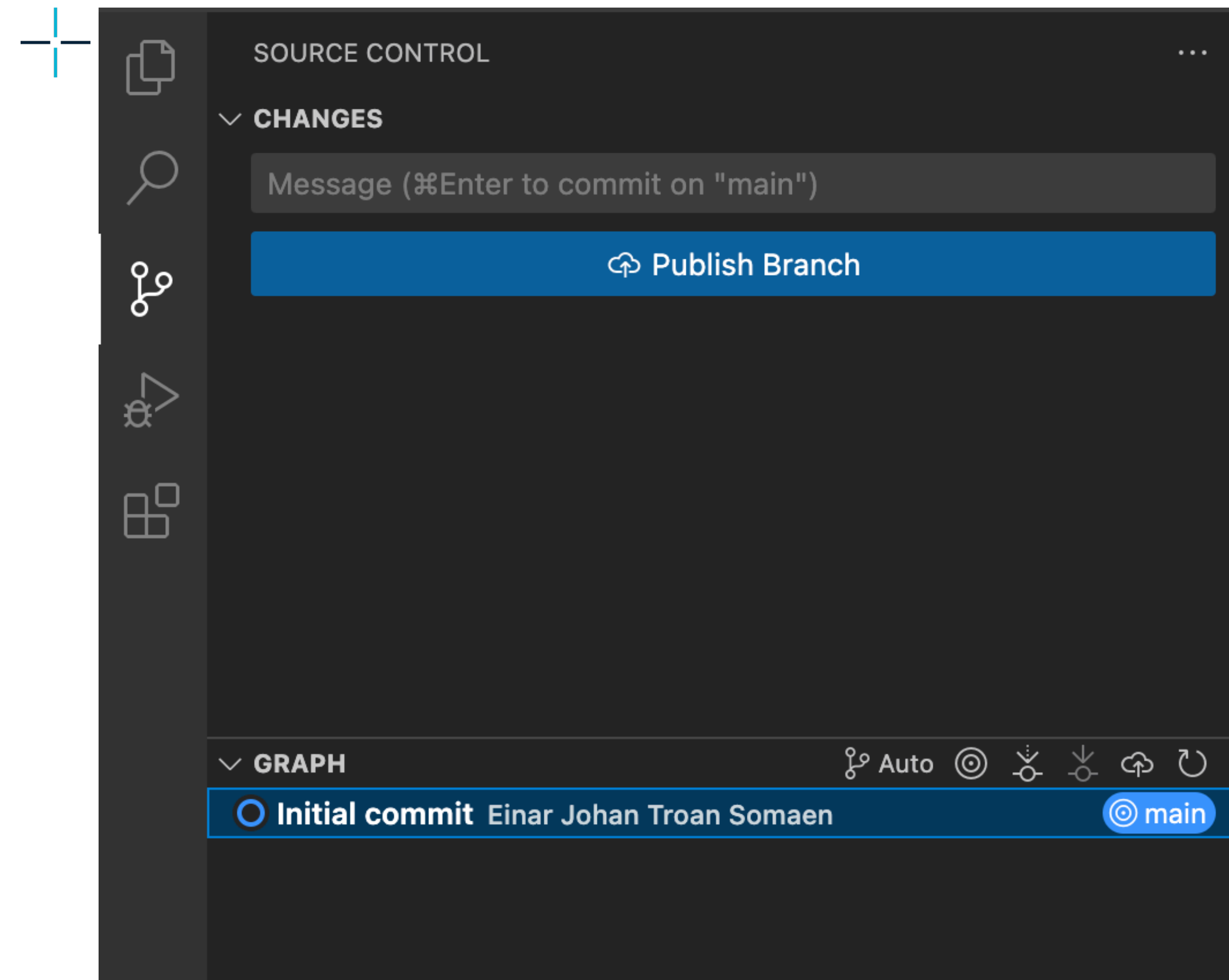
Git - commits (Demo)

Git - commits (Demo)

— Første commit:

Git - commits (Demo)

+ Første commit:



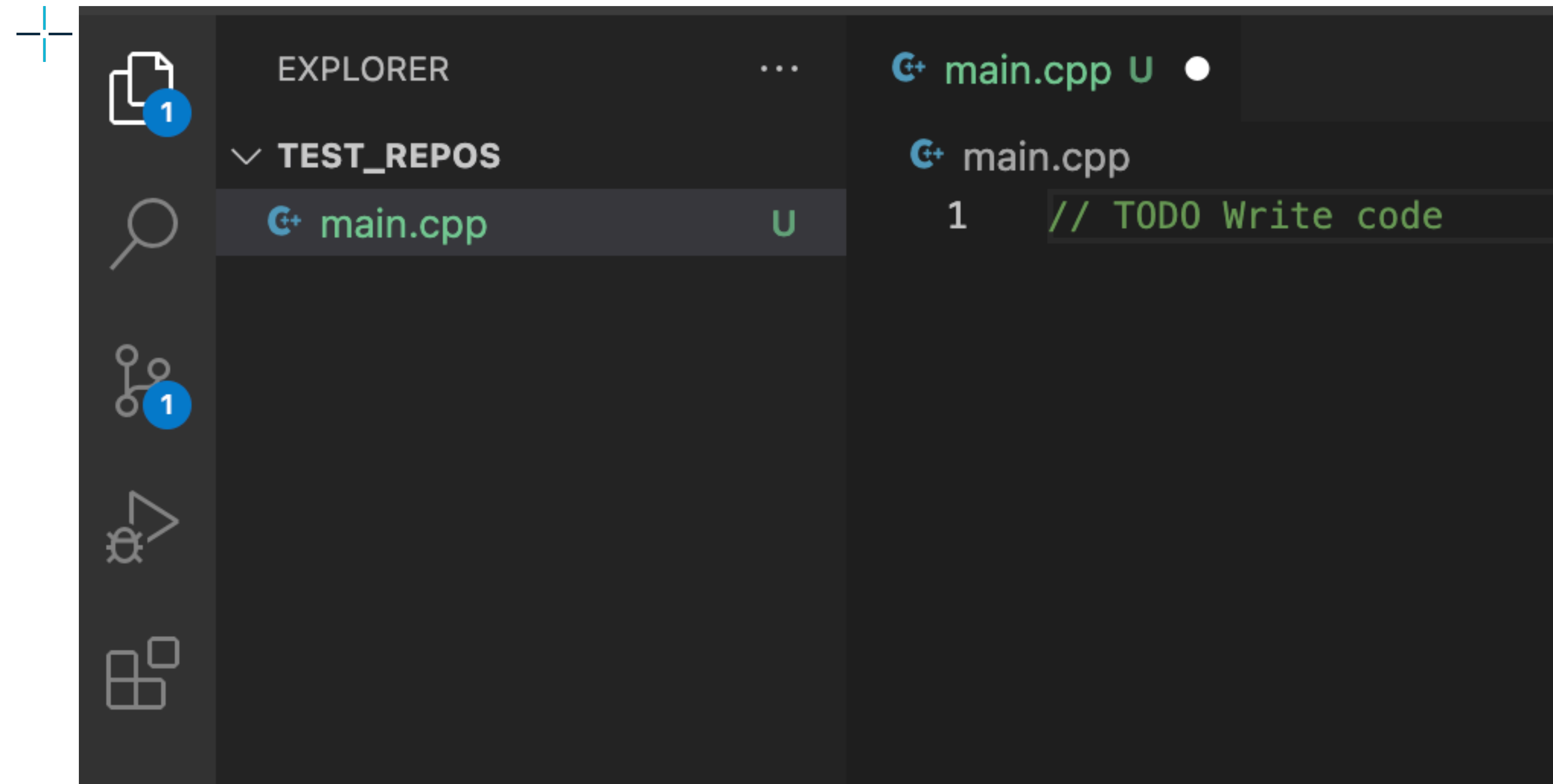
Git - commits (Demo)

Git - commits (Demo)

— La oss legge til en fil

Git - commits (Demo)

+ La oss legge til en fil



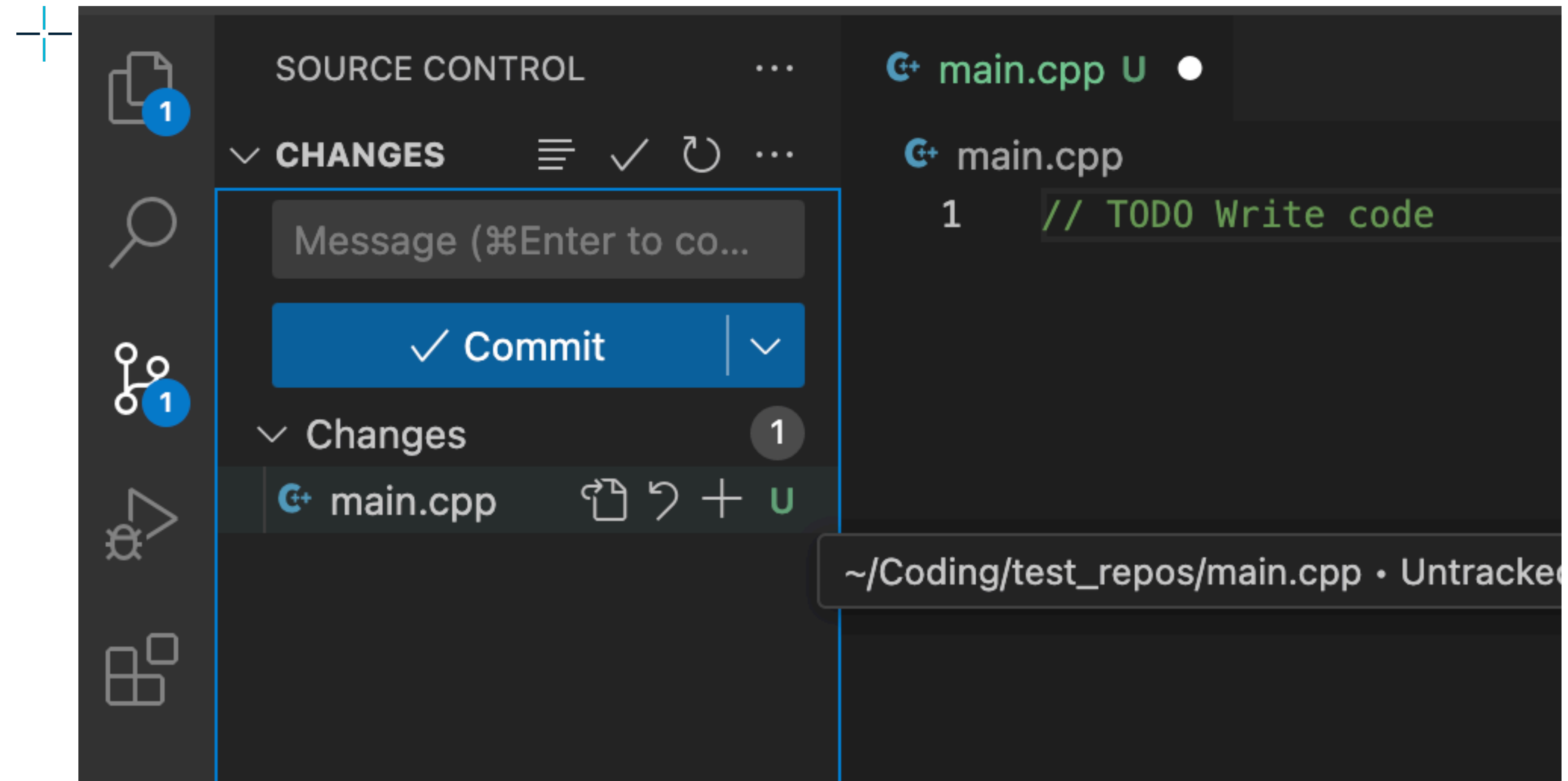
Git - commits (Demo)

Git - commits (Demo)

— La oss legge til en fil

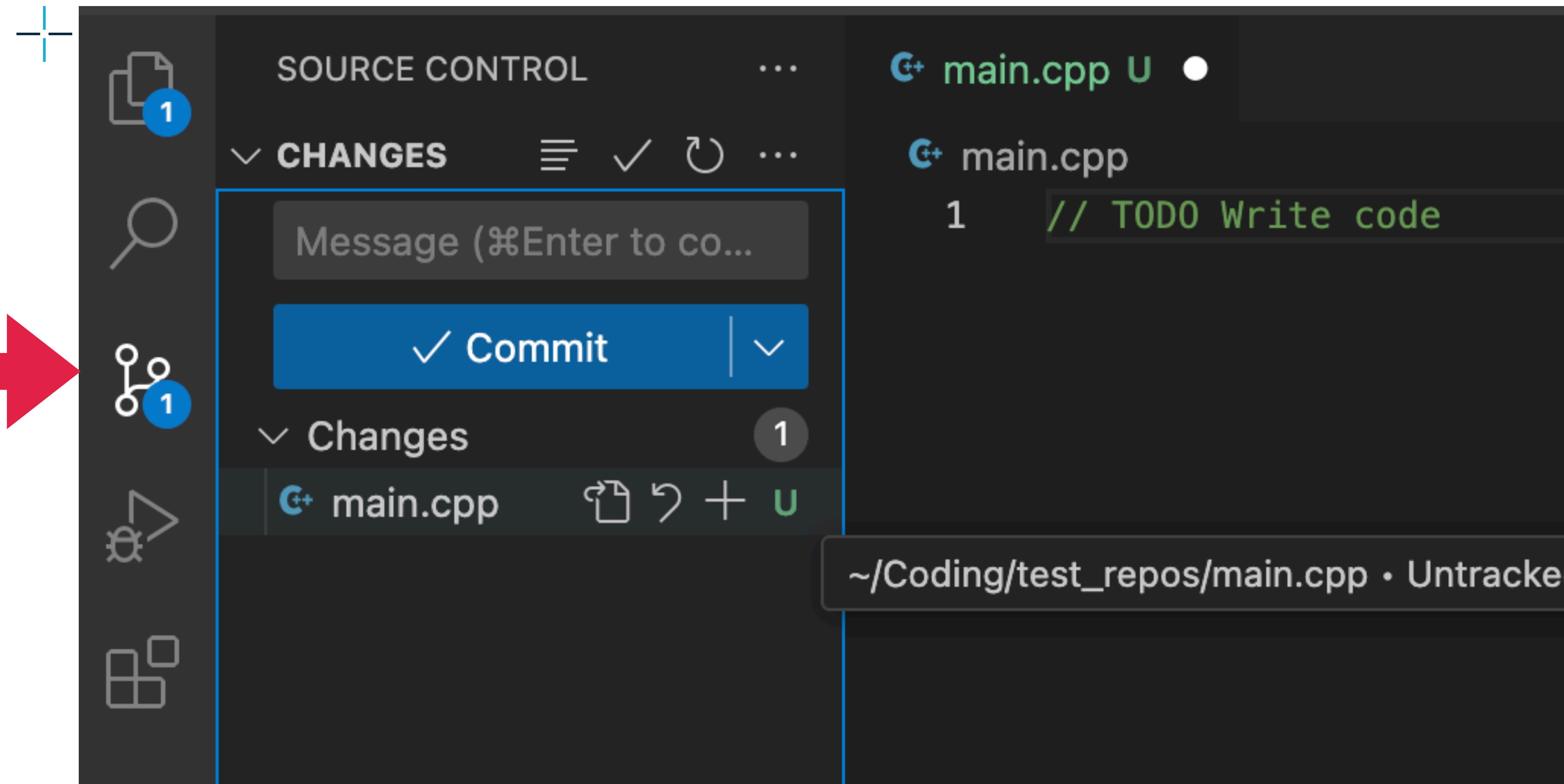
Git - commits (Demo)

+ La oss legge til en fil



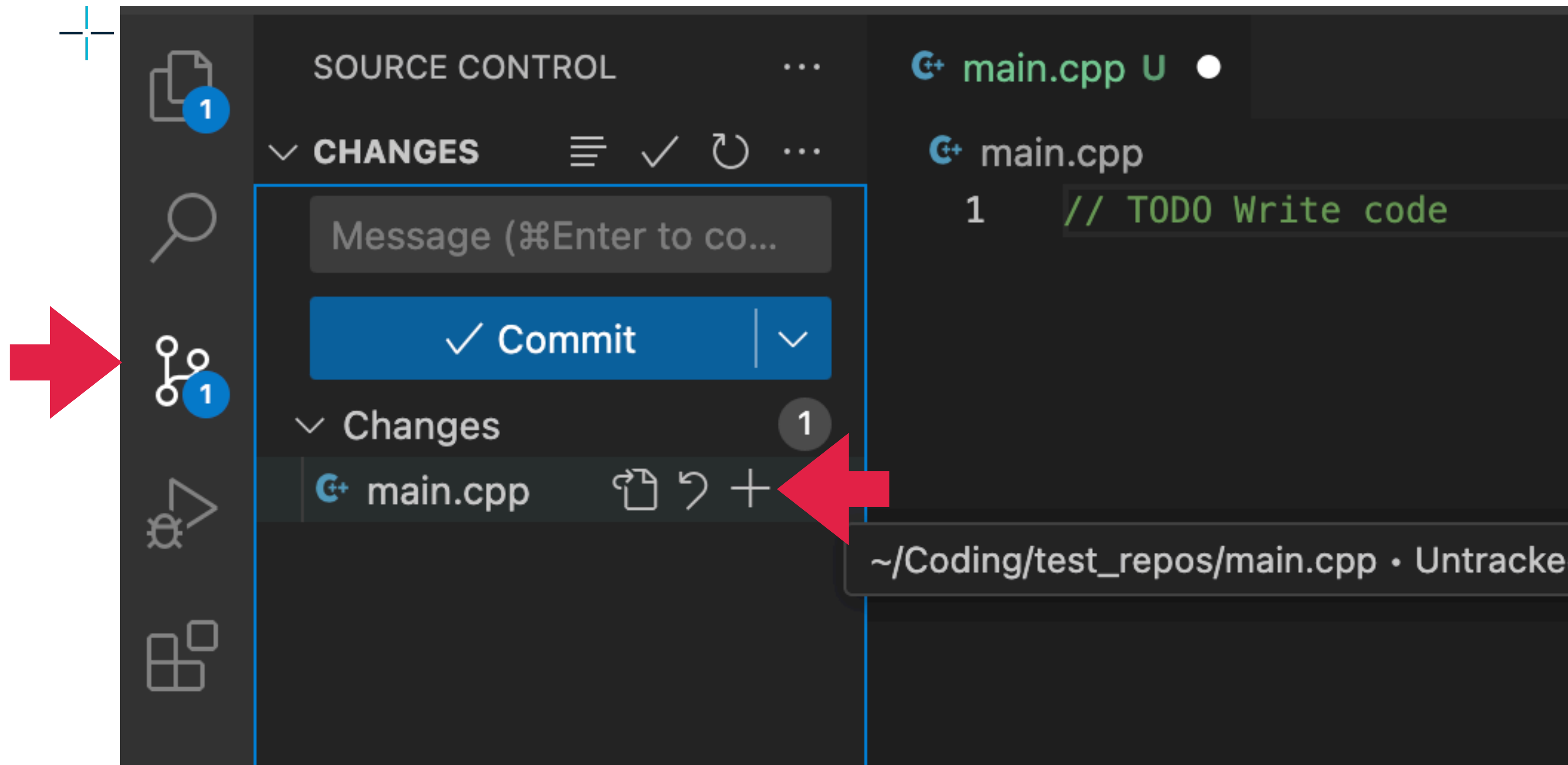
Git - commits (Demo)

+ La oss legge til en fil



Git - commits (Demo)

+ La oss legge til en fil



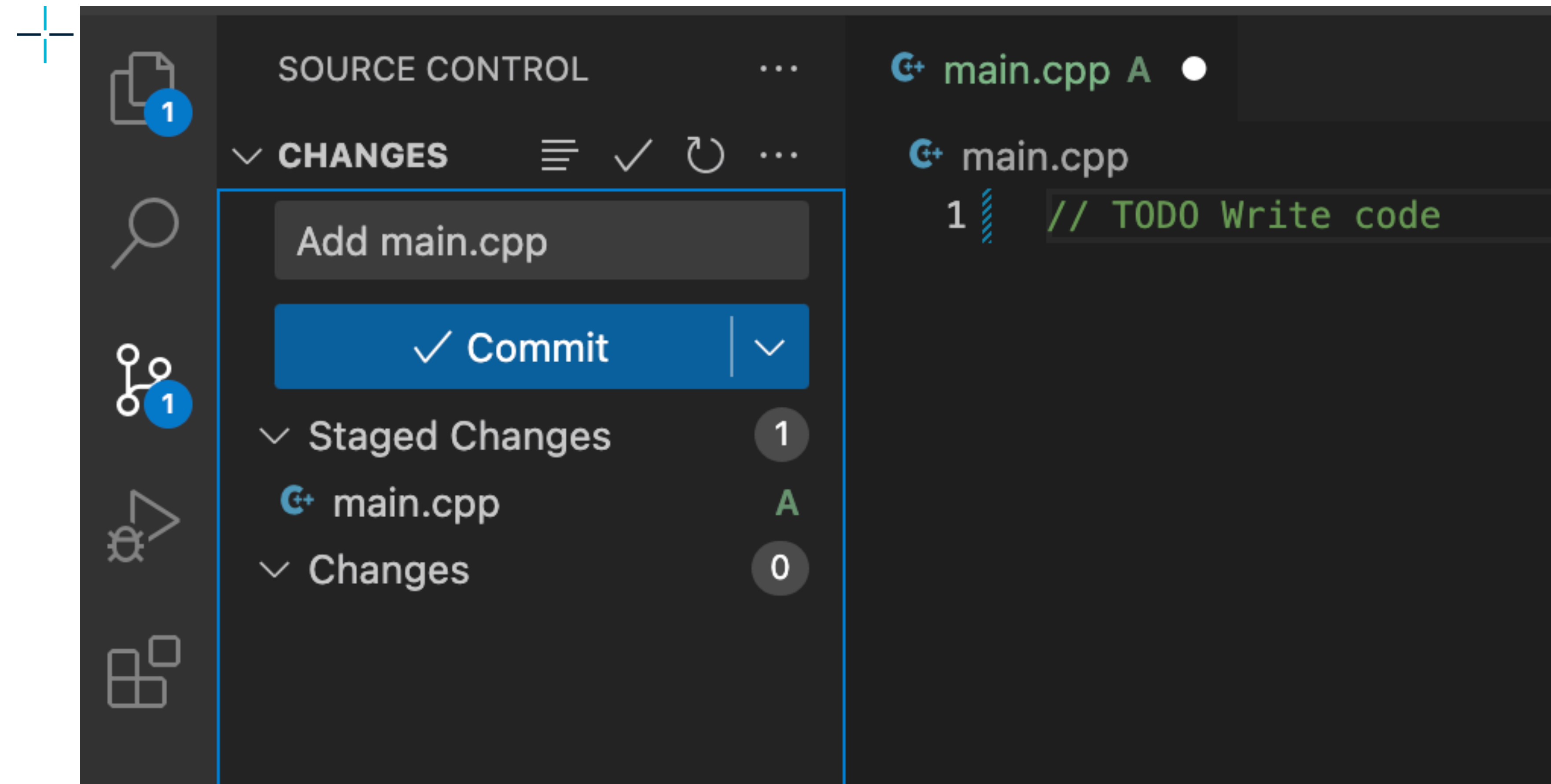
Git - commits (Demo)

Git - commits (Demo)

— La oss legge til en fil

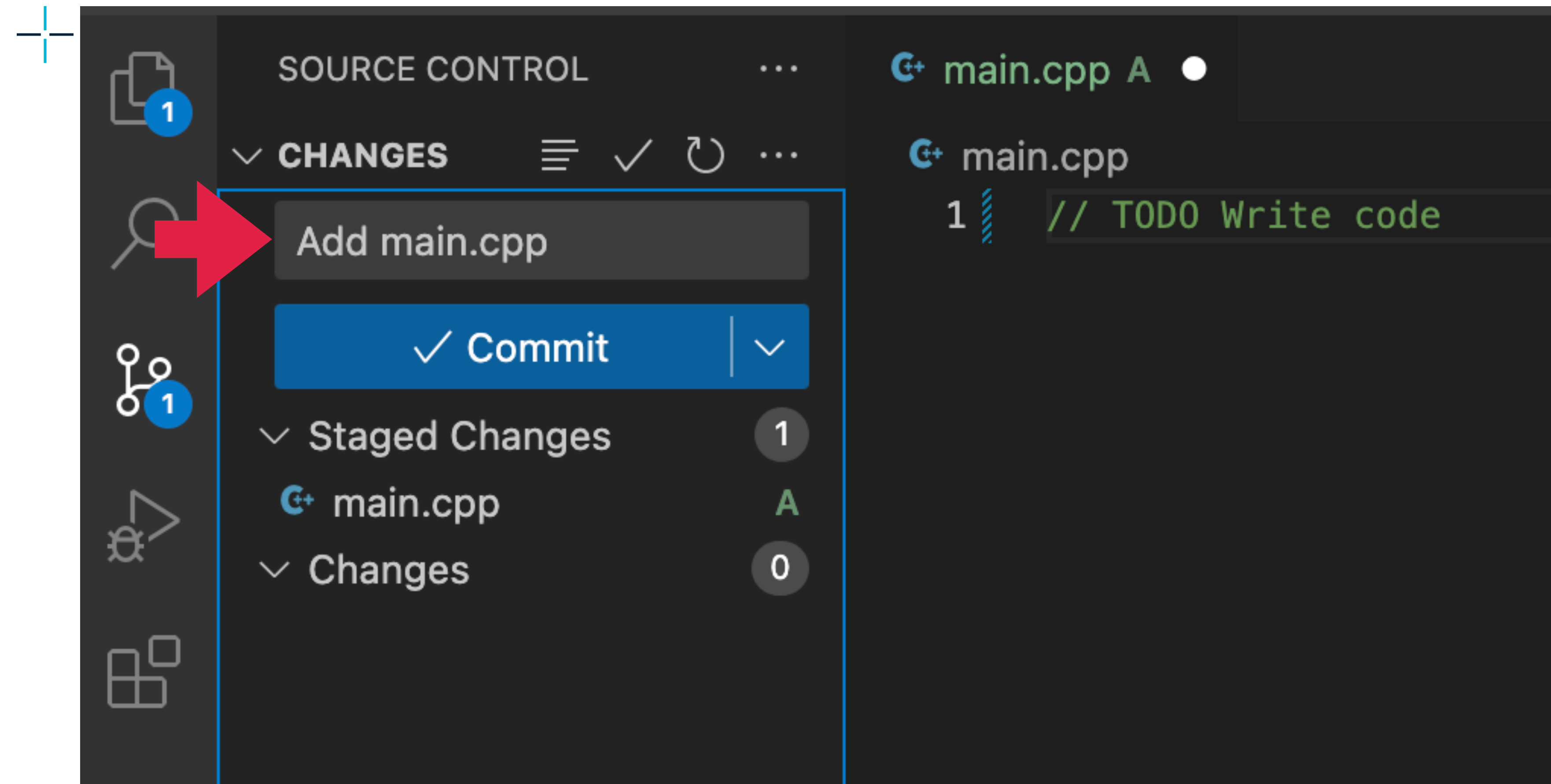
Git - commits (Demo)

+ La oss legge til en fil



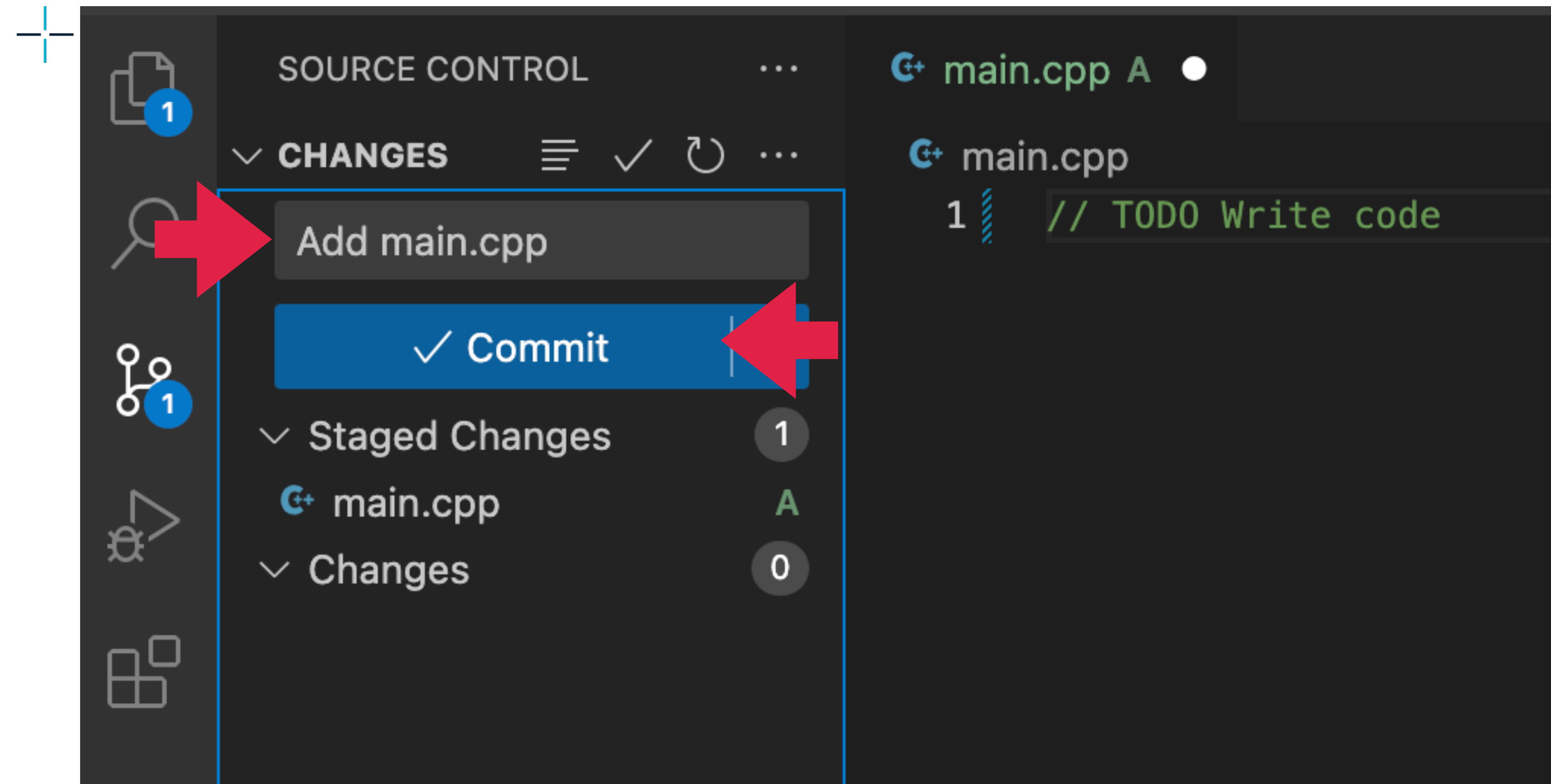
Git - commits (Demo)

+ La oss legge til en fil



Git - commits (Demo)

+ La oss legge til en fil



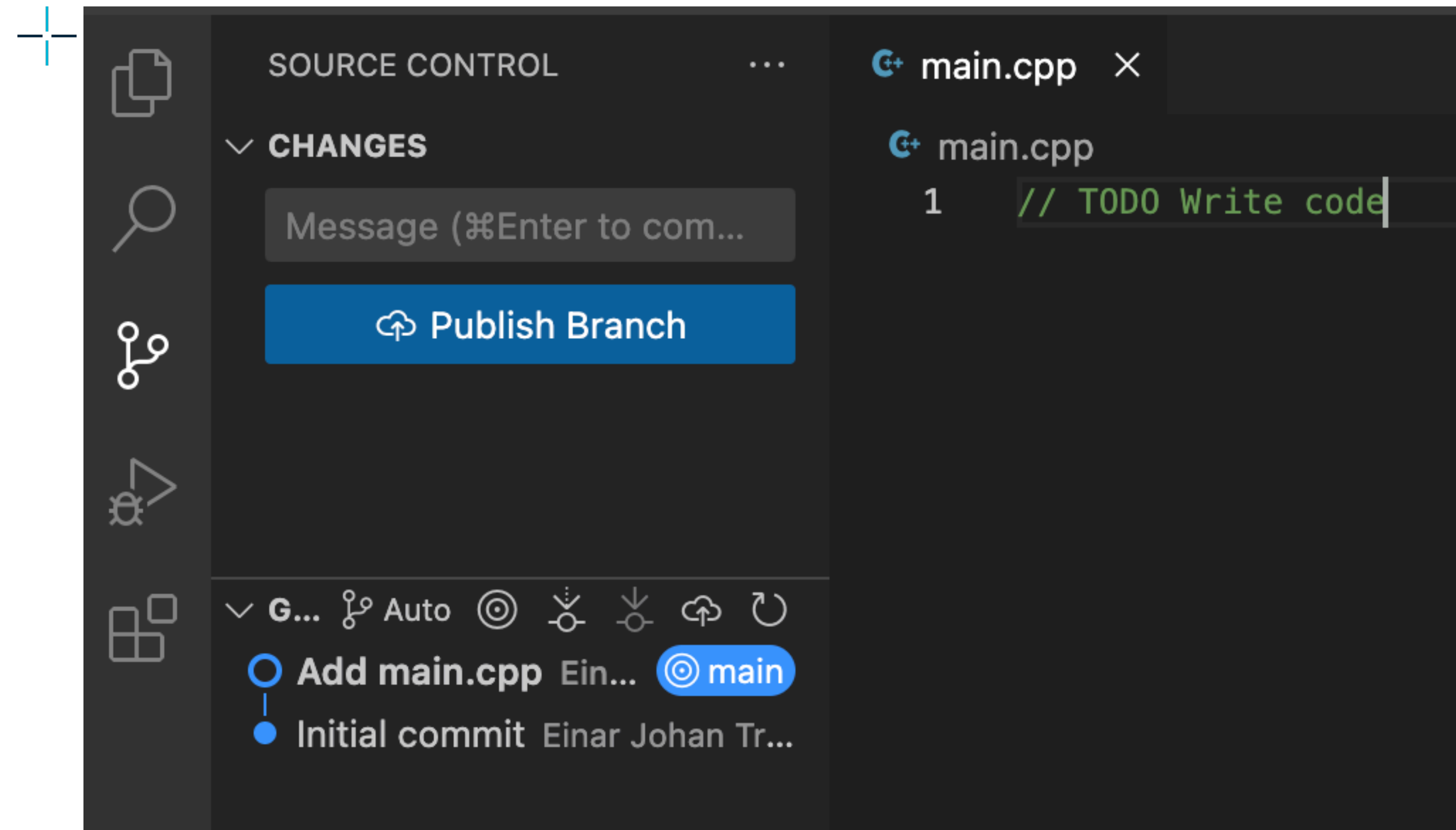
Git - commits (Demo)

Git - commits (Demo)

— La oss legge til en fil

Git - commits (Demo)

+ La oss legge til en fil



Git - commits (Demo)

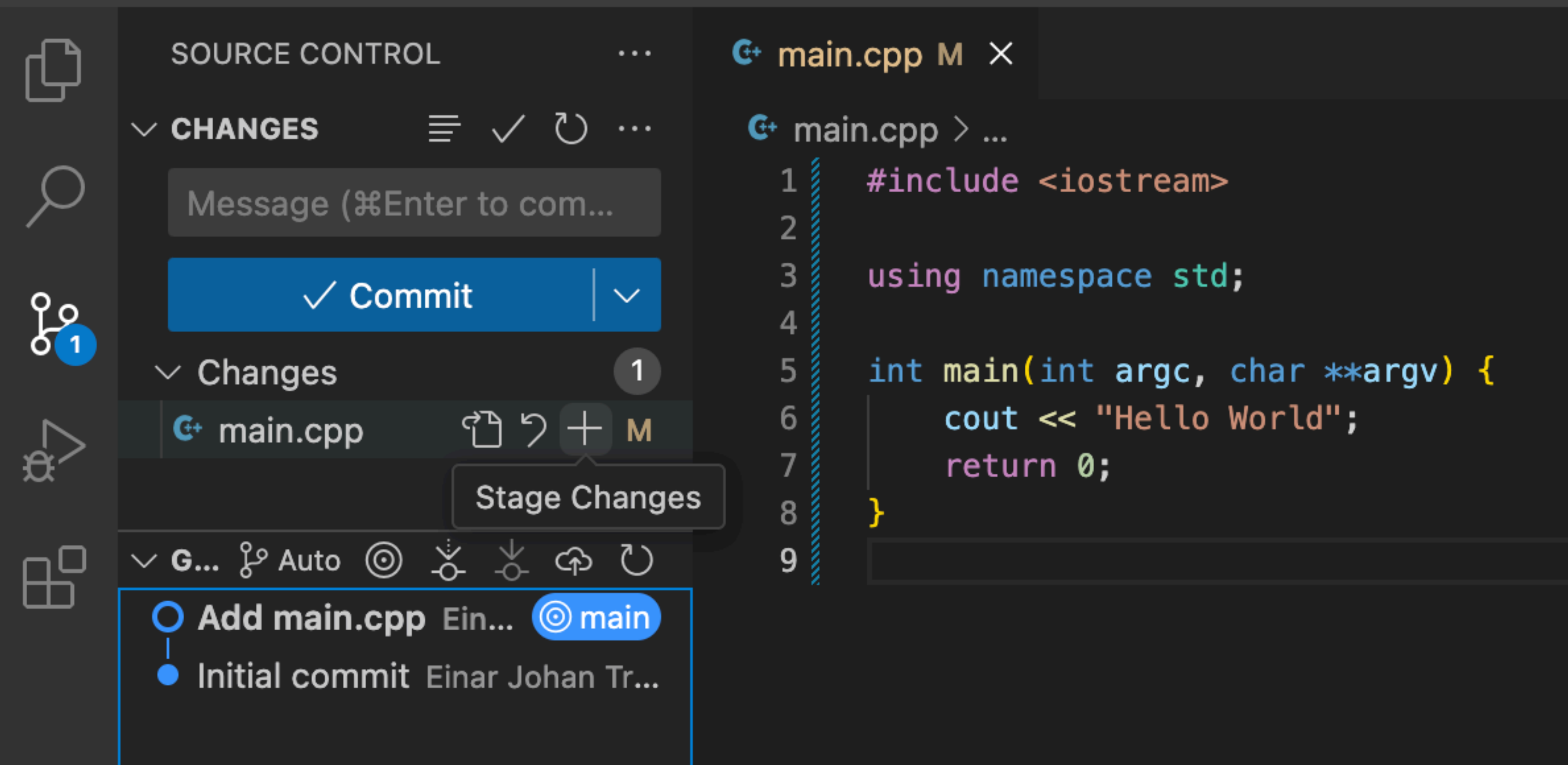
Git - commits (Demo)

— La oss legge til kode i denne fila

Git - commits (Demo)

+ La oss legge til kode i denne fila

+



The screenshot displays the Visual Studio Code interface with the Source Control panel on the left and the main editor on the right.

Source Control Panel (Left):

- SOURCE CONTROL** header.
- CHANGES** section with a message input field "Message (⌘Enter to com...".
- Commit** button.
- Changes** list showing **main.cpp** with a **Stage Changes** button.
- Commit History** section showing **Add main.cpp** and **Initial commit**.

Main Editor (Right):

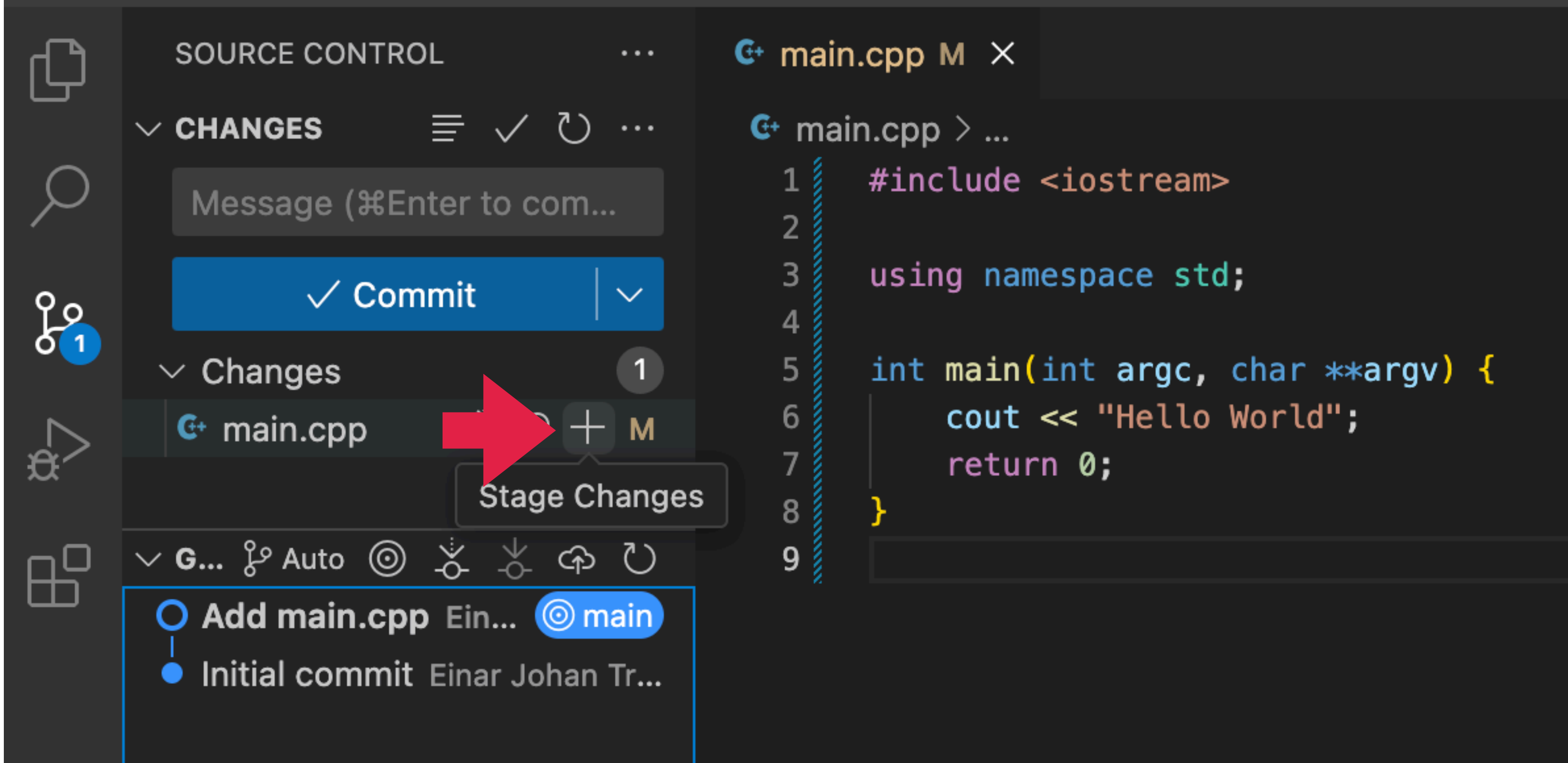
- File: **main.cpp**.
- Code content:

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main(int argc, char **argv) {
6      cout << "Hello World";
7      return 0;
8  }
9
```

Git - commits (Demo)

+ La oss legge til kode i denne fila

+



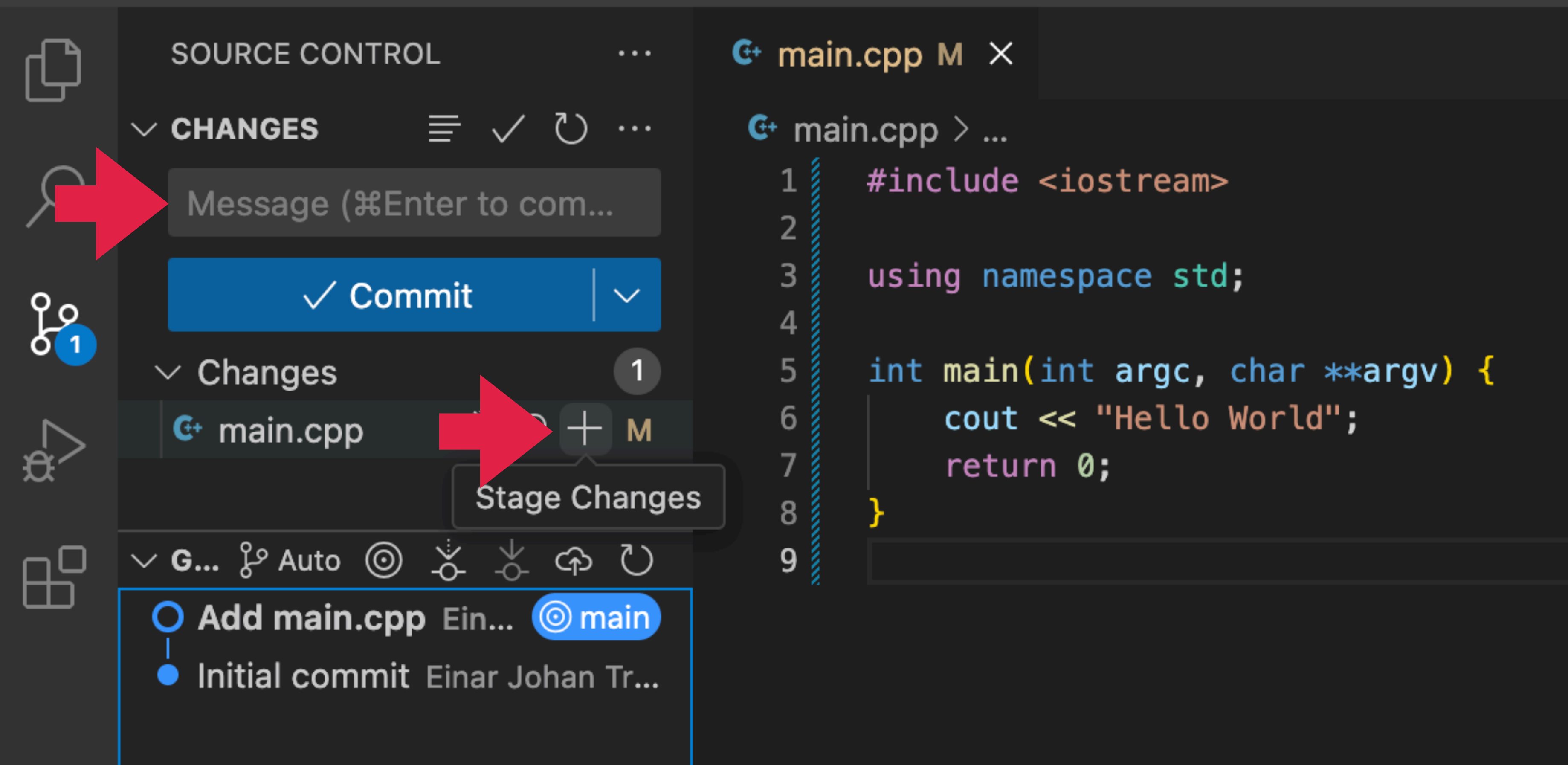
The screenshot shows the Visual Studio Code interface. On the left, the 'SOURCE CONTROL' panel is open, displaying the 'CHANGES' section. A red arrow points to the 'Stage Changes' button, which is labeled with a plus sign and the letter 'M'. Below this, the 'Changes' list shows 'main.cpp' with a plus sign and the letter 'M'. The 'Commit' button is visible above the changes list. The main editor area shows the 'main.cpp' file with the following code:

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main(int argc, char **argv) {
6      cout << "Hello World";
7      return 0;
8  }
9
```

Git - commits (Demo)

+ La oss legge til kode i denne fila

+



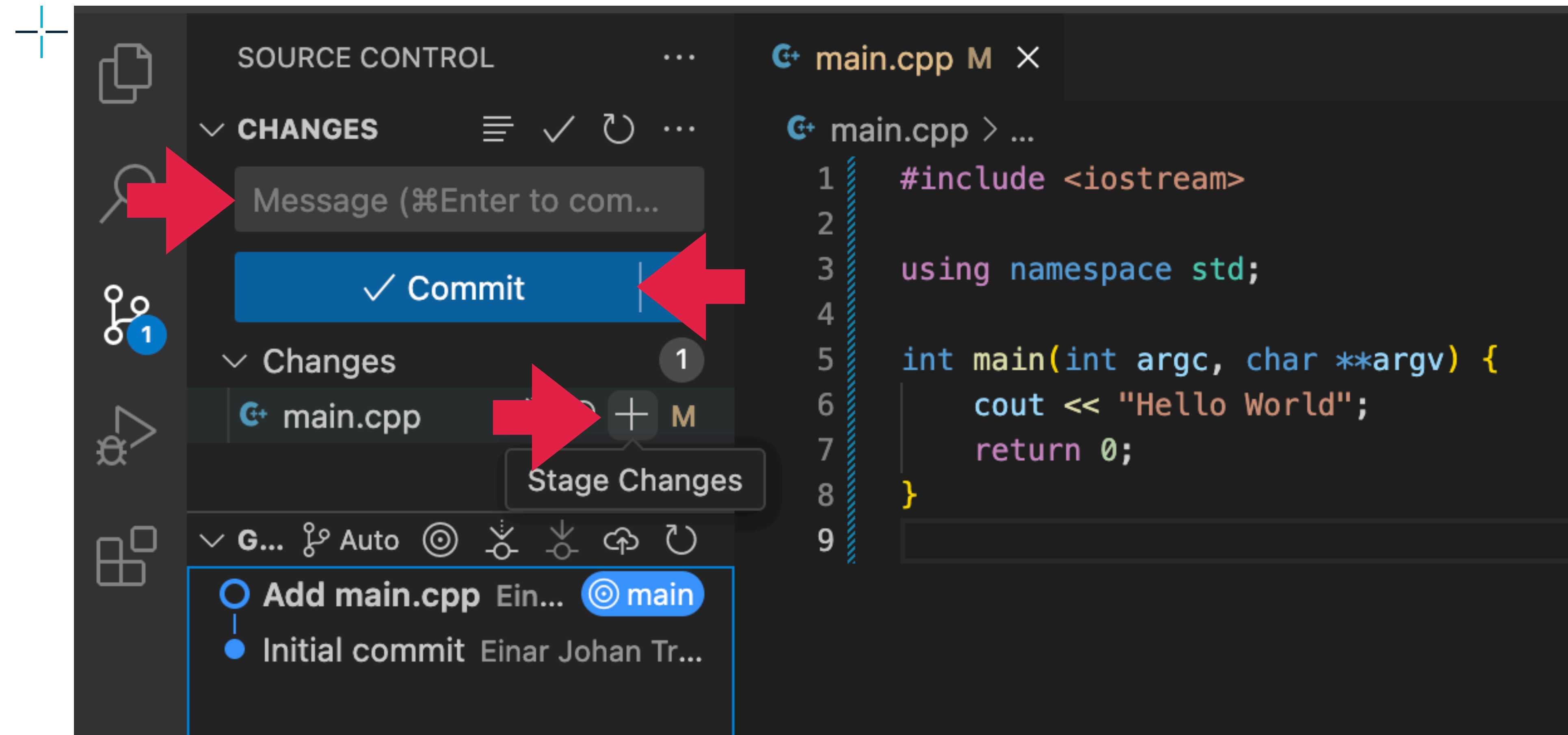
The screenshot shows the Visual Studio Source Control interface. On the left, the 'SOURCE CONTROL' pane is open, displaying the 'CHANGES' section. A red arrow points to the 'Commit' button. Below it, the 'Changes' section shows 'main.cpp' with a '+' icon and a tooltip that says 'Stage Changes'. Another red arrow points to this '+' icon. At the bottom, the 'Commit' dialog is open, showing the commit message 'Initial commit Einar Johan Tr...' and the 'main' branch selected.

main.cpp M X

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main(int argc, char **argv) {
6      cout << "Hello World";
7      return 0;
8  }
9
```

Git - commits (Demo)

— La oss legge til kode i denne fila



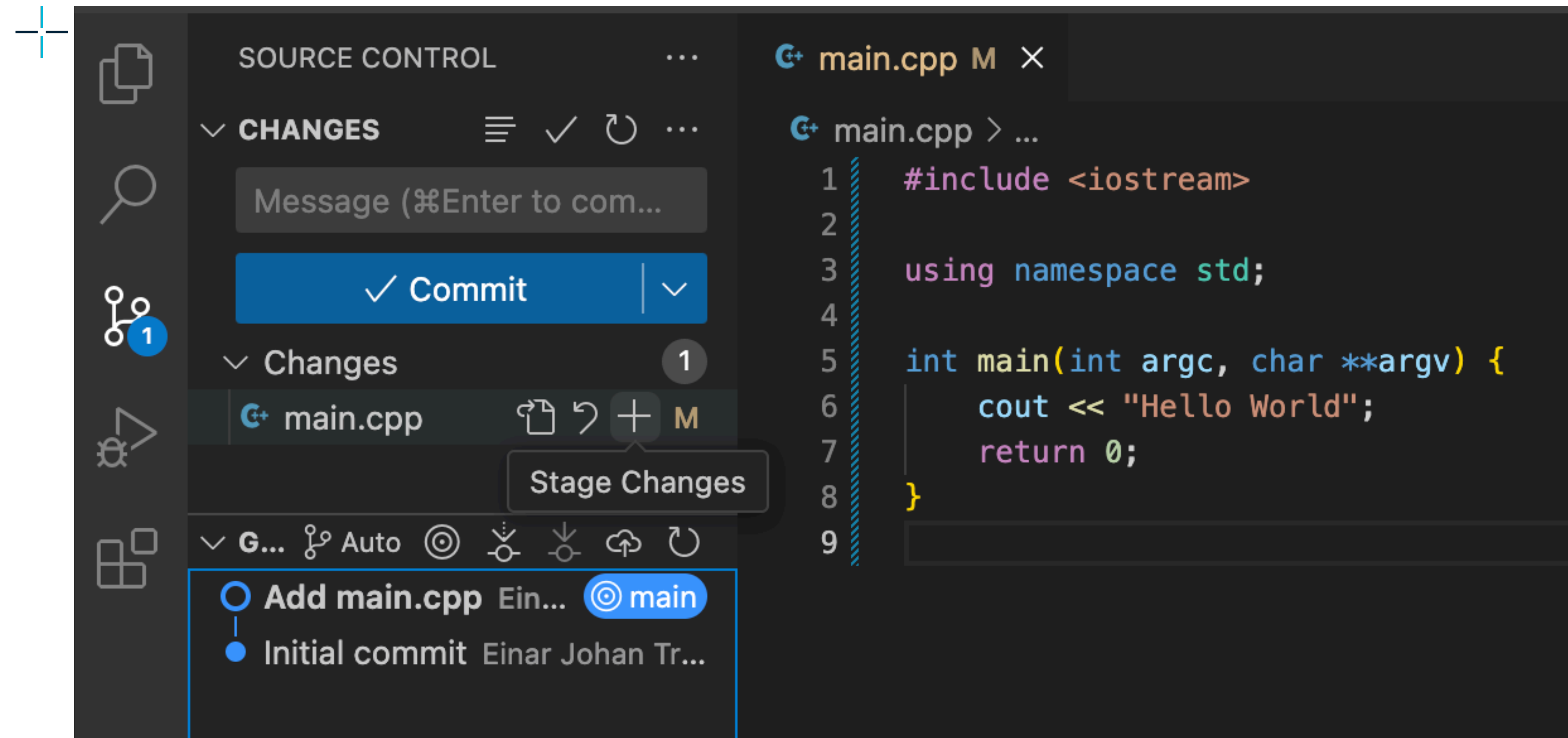
Git - commits (Demo)

Git - commits (Demo)

— La oss legge til kode i denne fila

Git - commits (Demo)

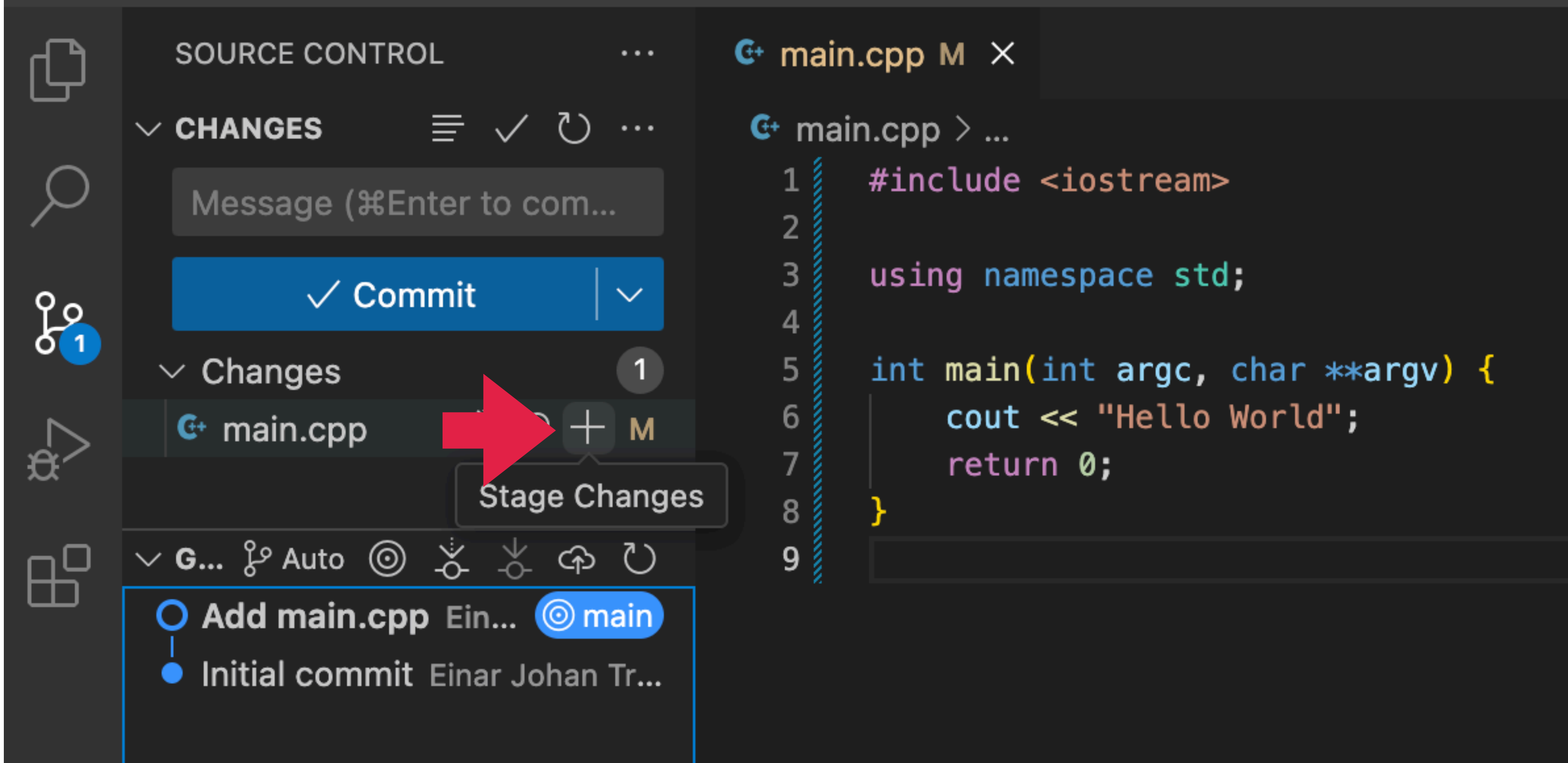
+ La oss legge til kode i denne fila



Git - commits (Demo)

+ La oss legge til kode i denne fila

+



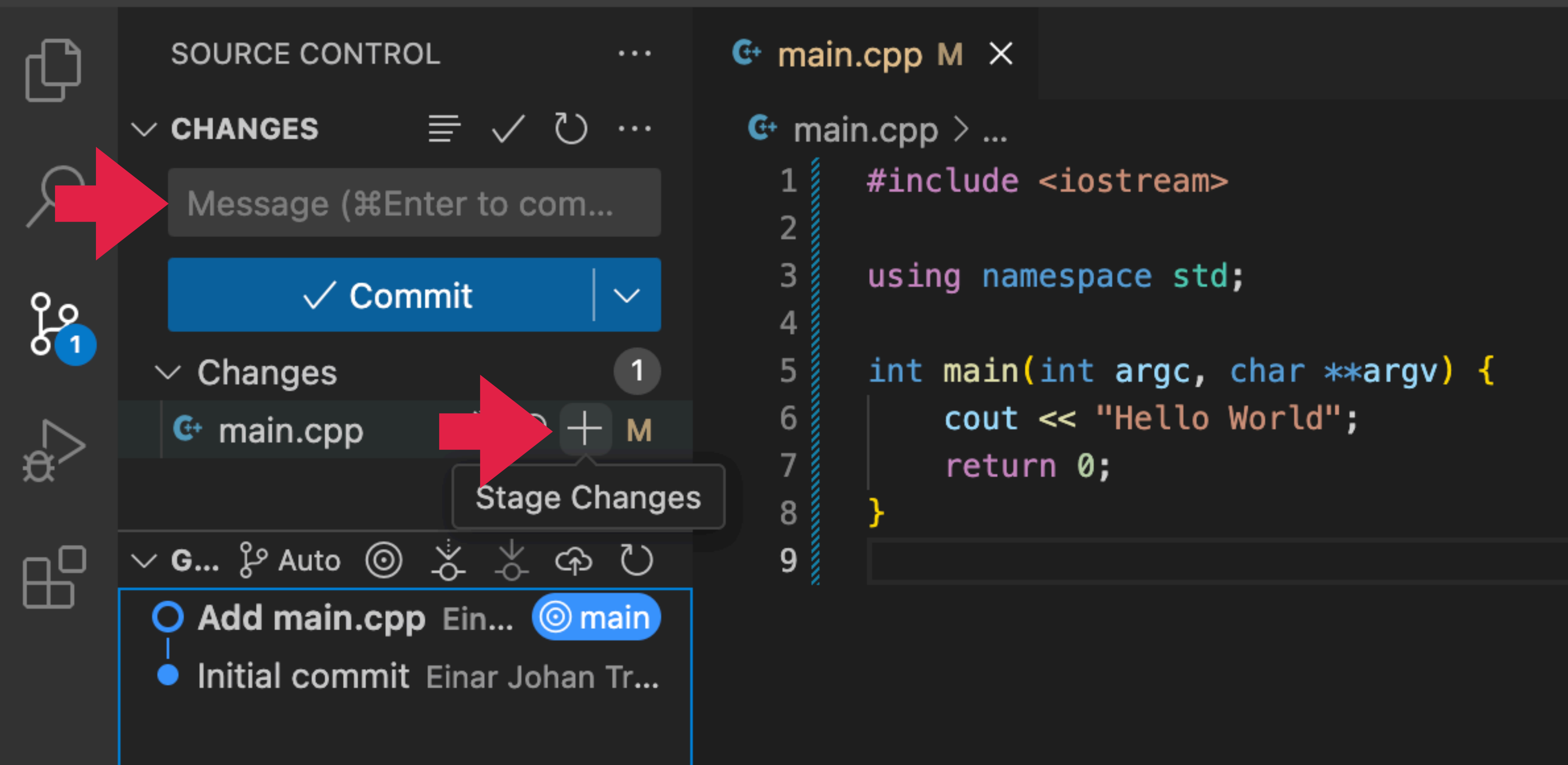
The screenshot shows the Visual Studio Source Control interface. On the left, the 'SOURCE CONTROL' pane is open, displaying the 'CHANGES' section. A message box prompts the user to 'Enter to commit'. Below this, a 'Commit' button is visible. The 'Changes' list shows 'main.cpp' with a red arrow pointing to the 'Stage Changes' button. The 'Commit' dialog is open, showing the commit message 'Initial commit' and the author 'Einar Johan Tr...'. The main editor on the right shows the code for 'main.cpp', which includes the standard C++ 'Hello World' program.

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main(int argc, char **argv) {
6      cout << "Hello World";
7      return 0;
8  }
9
```

Git - commits (Demo)

+ La oss legge til kode i denne fila

+



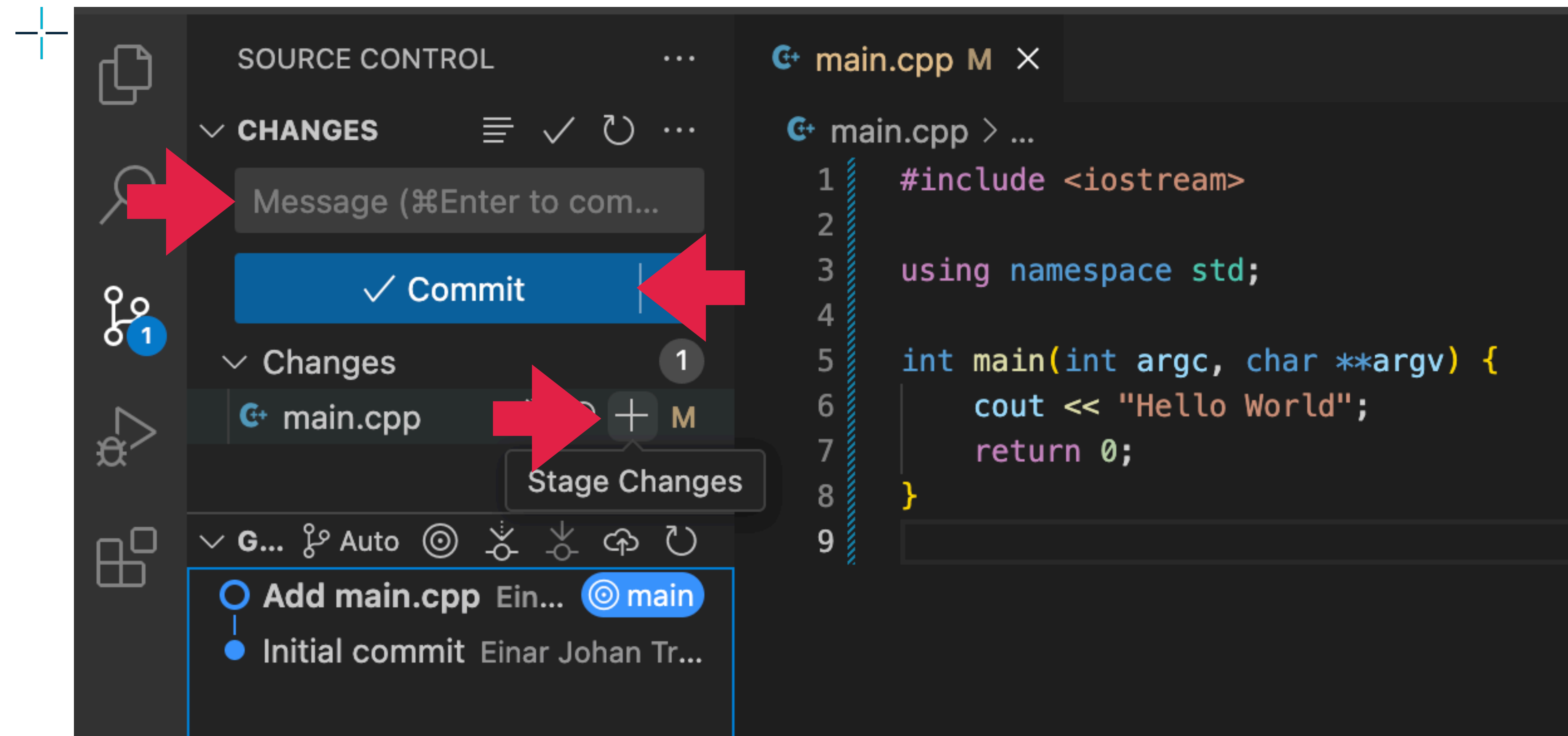
The screenshot shows the Visual Studio Source Control interface. On the left, the 'SOURCE CONTROL' pane is open, displaying the 'CHANGES' section. A red arrow points to the 'Commit' button. Below it, the 'Changes' section shows 'main.cpp' with a '+' icon and a tooltip that says 'Stage Changes'. Another red arrow points to this '+' icon. At the bottom, the 'Commit' dialog is open, showing the commit message 'Initial commit Einar Johan Tr...' and the 'main' branch selected.

main.cpp M X

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main(int argc, char **argv) {
6      cout << "Hello World";
7      return 0;
8  }
9
```

Git - commits (Demo)

- La oss legge til kode i denne fila



Git - commits (Demo)

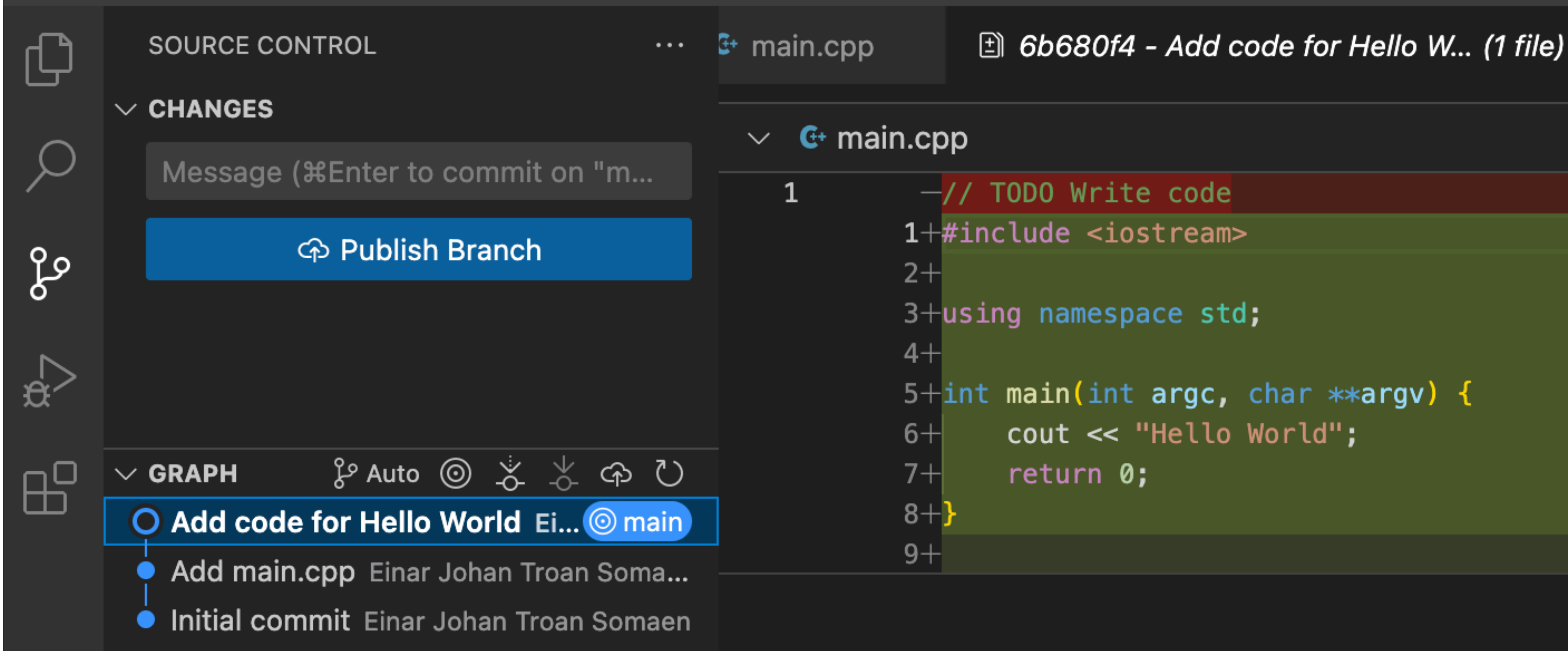
Git - commits (Demo)

— Da har vi en commit, la oss se på den

Git - commits (Demo)

+ Da har vi en commit, la oss se på den

+



The screenshot shows the Visual Studio Source Control interface. On the left, the 'SOURCE CONTROL' pane is open, displaying the 'CHANGES' section with a message input field and a 'Publish Branch' button. Below this, the 'GRAPH' section shows a commit history with two entries: 'Add code for Hello World Ei...' (selected) and 'Add main.cpp Einar Johan Troan Soma...'. The right pane shows the code editor for 'main.cpp', displaying the following C++ code:

```
1 // TODO Write code
1+ #include <iostream>
2+
3+ using namespace std;
4+
5+ int main(int argc, char **argv) {
6+     cout << "Hello World";
7+     return 0;
8+ }
9+
```

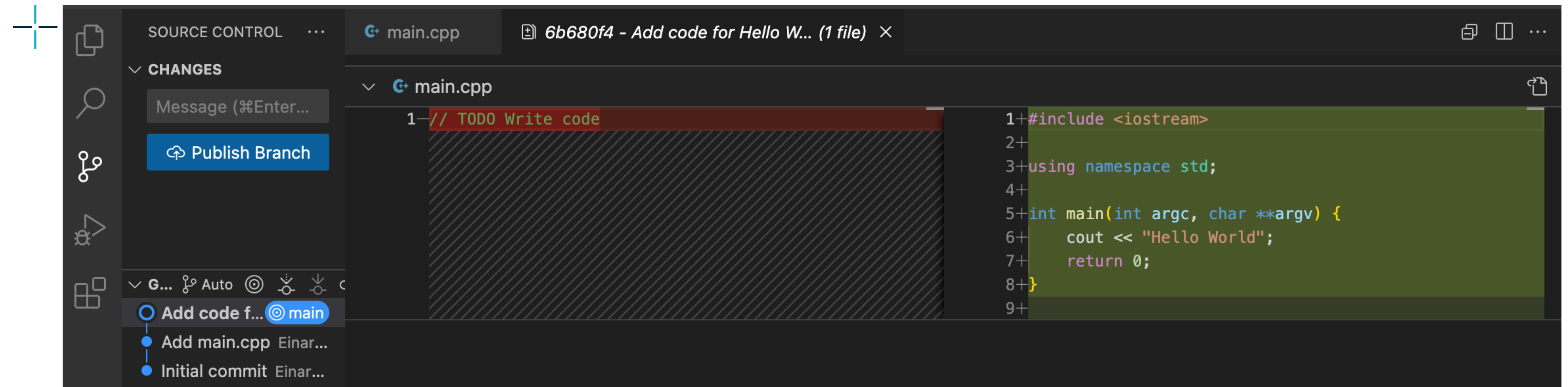
Git - commits (Demo)

Git - commits (Demo)

└─ Vindusbredden avgjør sammenligningsstil:

Git - commits (Demo)

+ Vindusbredden avgjør sammenligningsstil:



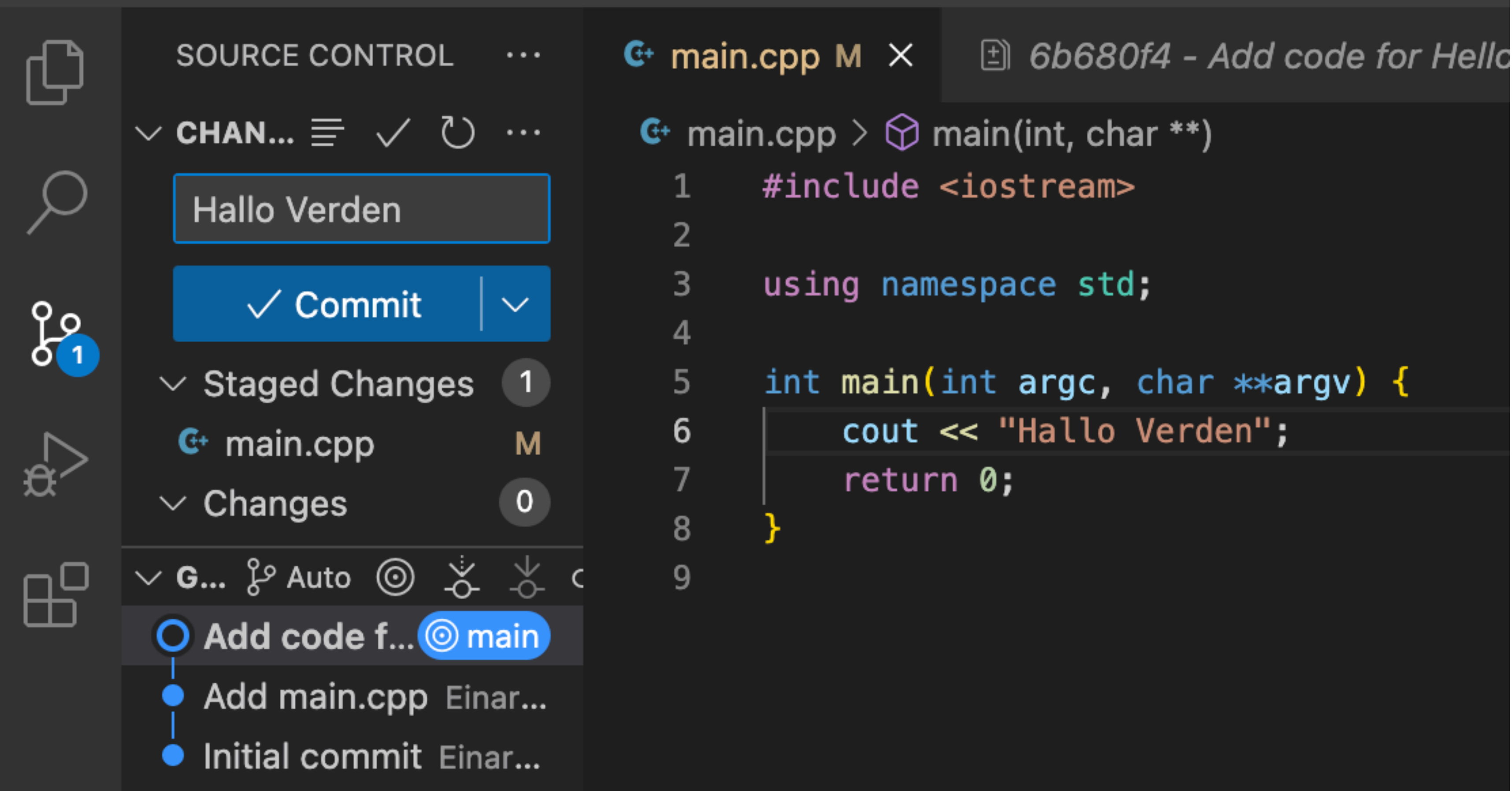
Git - commits (Demo)

Git - commits (Demo)

— La oss endre koden:

Git - commits (Demo)

+ La oss endre koden:

+ 

The screenshot shows the Visual Studio Source Control interface. On the left, the 'SOURCE CONTROL' pane is open, showing a commit message 'Hallo Verden' and a 'Commit' button. Below this, the 'Staged Changes' section shows 'main.cpp' with a modification (M). The 'Changes' section shows 'Add code f...' with a commit icon. The commit history shows 'Add main.cpp Einar...' and 'Initial commit Einar...'. On the right, the 'main.cpp' file is open, showing the following code:

```
main.cpp > main(int, char **)
1  #include <iostream>
2
3  using namespace std;
4
5  int main(int argc, char **argv) {
6      cout << "Hallo Verden";
7      return 0;
8  }
9
```

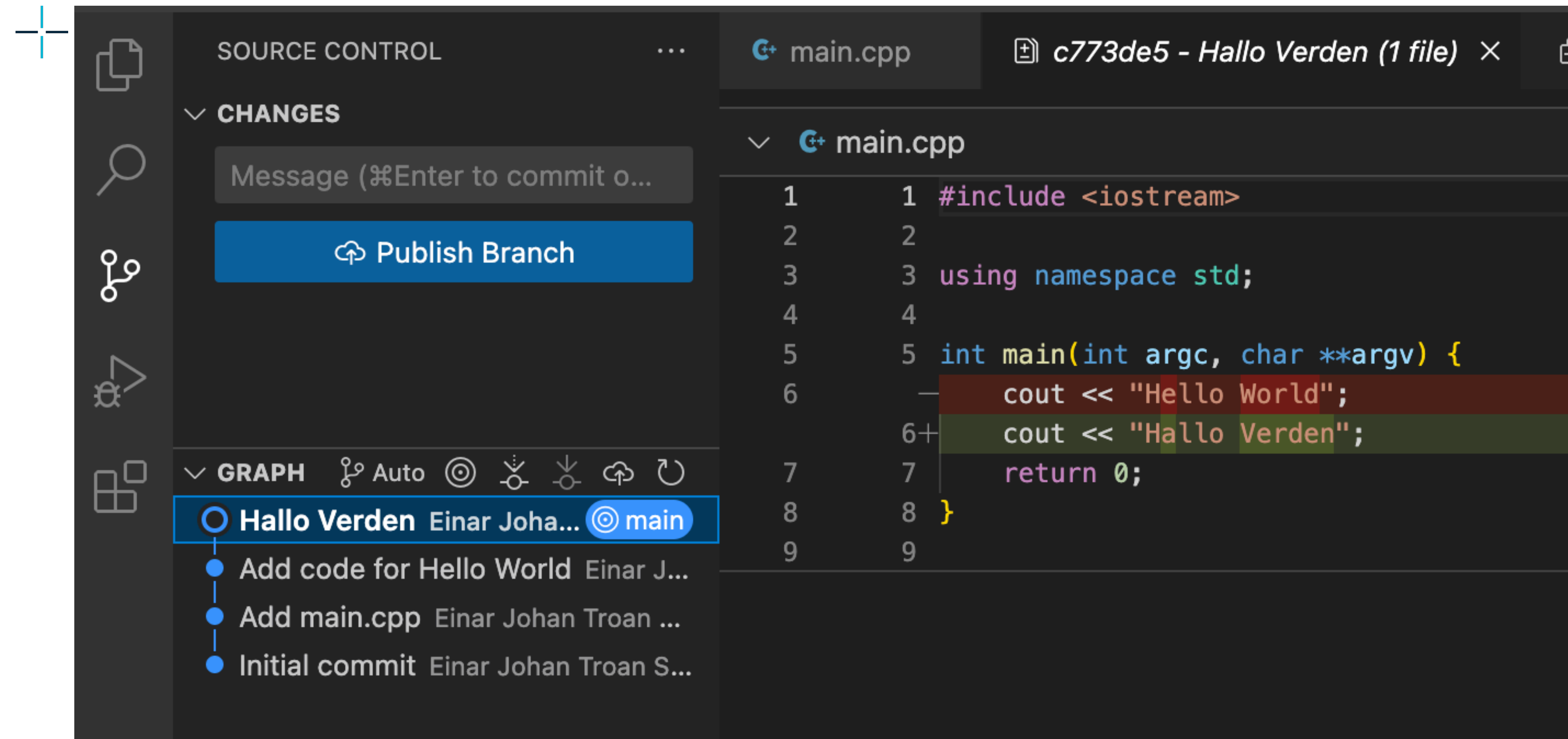
Git - commits (Demo)

Git - commits (Demo)

— La oss endre koden:

Git - commits (Demo)

+ La oss endre koden:



The screenshot displays the Visual Studio Code interface during a Git commit process. On the left, the 'SOURCE CONTROL' sidebar is active, showing a commit message input field with the placeholder 'Message (%Enter to commit o...)' and a blue 'Publish Branch' button. Below this, the 'GRAPH' section shows a commit history with three entries: 'Hallo Verden Einar Joha...', 'Add code for Hello World Einar J...', and 'Add main.cpp Einar Johan Troan ...'. The 'main' branch is selected. The main editor window shows the 'main.cpp' file with the following code:

```
1 1 #include <iostream>
2 2
3 3 using namespace std;
4 4
5 5 int main(int argc, char **argv) {
6 6     cout << "Hello World";
6+ cout << "Hallo Verden";
7 7     return 0;
8 8 }
9 9
```

Git - Nyttige tips

Git - Nyttige tips

—+— Commit early / commit often

Git - Nyttige tips

- + Commit early / commit often
- + I verste fall kan man rydde i historikken senere

Git - Nyttige tips

- Commit early / commit often
- I verste fall kan man rydde i historikken senere
- Skriv ALLTID gode commit meldinger

Git - Nyttige tips

- Commit early / commit often
 - I verste fall kan man rydde i historikken senere
- Skriv ALLTID gode commit meldinger
 - Overordnet beskrivelse av årsak.

Git - Nyttige tips

- Commit early / commit often
 - I verste fall kan man rydde i historikken senere
- Skriv ALLTID gode commit meldinger
 - Overordnet beskrivelse av årsak.
 - Nøyaktig hva som ble endret inneholder commiten allerede.

Git - Nyttige tips

- Commit early / commit often
 - I verste fall kan man rydde i historikken senere
- Skriv ALLTID gode commit meldinger
 - Overordnet beskrivelse av årsak.
 - Nøyaktig hva som ble endret inneholder commiten allerede.
 - Aktiv form, ikke fortid.

Git - Nyttige tips

- Commit early / commit often
 - I verste fall kan man rydde i historikken senere
- Skriv ALLTID gode commit meldinger
 - Overordnet beskrivelse av årsak.
 - Nøyaktig hva som ble endret inneholder commiten allerede.
 - Aktiv form, ikke fortid.
 - «Add comments to explain how the math works»

Git - Nyttige tips

- Commit early / commit often
 - I verste fall kan man rydde i historikken senere
- Skriv ALLTID gode commit meldinger
 - Overordnet beskrivelse av årsak.
 - Nøyaktig hva som ble endret inneholder commiten allerede.
 - Aktiv form, ikke fortid.
 - «Add comments to explain how the math works»
 - «Fix mistakes in calculations» etc

Git - Avanserte ting

Git - Avanserte ting

- Mest for å vise at det finnes en del kule ting man kan gjøre når man har historikk

Git - Avanserte ting

- Mest for å vise at det finnes en del kule ting man kan gjøre når man har historikk
- Bruke historikken til feilsøking: git bisect

Git - Avanserte ting

- Mest for å vise at det finnes en del kule ting man kan gjøre når man har historikk
- Bruke historikken til feilsøking: git bisect
 - Veileder deg gjennom et «binærsøk» for å finne når et problem oppstod

Git - Avanserte ting

- Mest for å vise at det finnes en del kule ting man kan gjøre når man har historikk
- Bruke historikken til feilsøking: git bisect
 - Veileder deg gjennom et «binærsøk» for å finne når et problem oppstod
 - Dette er lettest hvis man passer på at koden kompilerer før hver commit.

Git - Avanserte ting

- Mest for å vise at det finnes en del kule ting man kan gjøre når man har historikk
- Bruke historikken til feilsøking: git bisect
 - Veileder deg gjennom et «binærsøk» for å finne når et problem oppstod
 - Dette er lettest hvis man passer på at koden kompilerer før hver commit.
- Finne årsak til at en linje ble endret: git blame

Git - Avanserte ting

- Mest for å vise at det finnes en del kule ting man kan gjøre når man har historikk
- Bruke historikken til feilsøking: git bisect
 - Veileder deg gjennom et «binærsøk» for å finne når et problem oppstod
 - Dette er lettest hvis man passer på at koden kompilerer før hver commit.
- Finne årsak til at en linje ble endret: git blame
- Vi kan omskrive historikken: git rebase

Git - Avanserte ting

- Mest for å vise at det finnes en del kule ting man kan gjøre når man har historikk
- Bruke historikken til feilsøking: `git bisect`
 - Veileder deg gjennom et «binærsøk» for å finne når et problem oppstod
 - Dette er lettest hvis man passer på at koden kompilerer før hver commit.
- Finne årsak til at en linje ble endret: `git blame`
- Vi kan omskrive historikken: `git rebase`
- Vi kan stage endringer bit for bit: `git add -p`

Git - NB:

Git - NB:

— Git krever installasjon, avhengig av platform:

Git - NB:

- Git krever installasjon, avhengig av platform:
- <https://git-scm.com/downloads>

Git - NB:

- Git krever installasjon, avhengig av platform:
 - <https://git-scm.com/downloads>
- Man får feilmelding/advarsel hvis man ikke har satt brukernavn/epost:

Git - NB:

- + Git krever installasjon, avhengig av platform:
- + <https://git-scm.com/downloads>
- + Man får feilmelding/advarsel hvis man ikke har satt brukernavn/epost:

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

PORTS

```
● demo@MacBookPro example3 % git config --global user.name "Einar Johan Troan Somaen"
● demo@MacBookPro example3 % git config --global user.email "einarjohan@troansomaen.com"
○ demo@MacBookPro example3 %
```

Git - versjonskontroll

Git - versjonskontroll

— Et av de mest generelt nyttige verktøyene å ha «i verktøykassa»

Git - versjonskontroll

- Et av de mest generelt nyttige verktøyene å ha «i verktøykassa»
- Skriver man kode profesjonelt brukes versjonskontroll, og git er veldig vanlig.

Git - versjonskontroll

- Et av de mest generelt nyttige verktøyene å ha «i verktøykassa»
 - Skriver man kode profesjonelt brukes versjonskontroll, og git er veldig vanlig.
 - Kan også (med fordel) brukes til ting som masteroppgaver, prosjekter e.l.

arm

Thank You

Danke

Gracias

Grazie

谢谢

ありがとう

Asante

Merci

감사합니다

धन्यवाद

Kiitos

شكراً

ধন্যবাদ



The Arm trademarks featured in this presentation are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All rights reserved. All other marks featured may be trademarks of their respective owners.

www.arm.com/company/policies/trademarks