

Titan Documentation > Introduction > Getting Started

Chapter 3. Getting Started

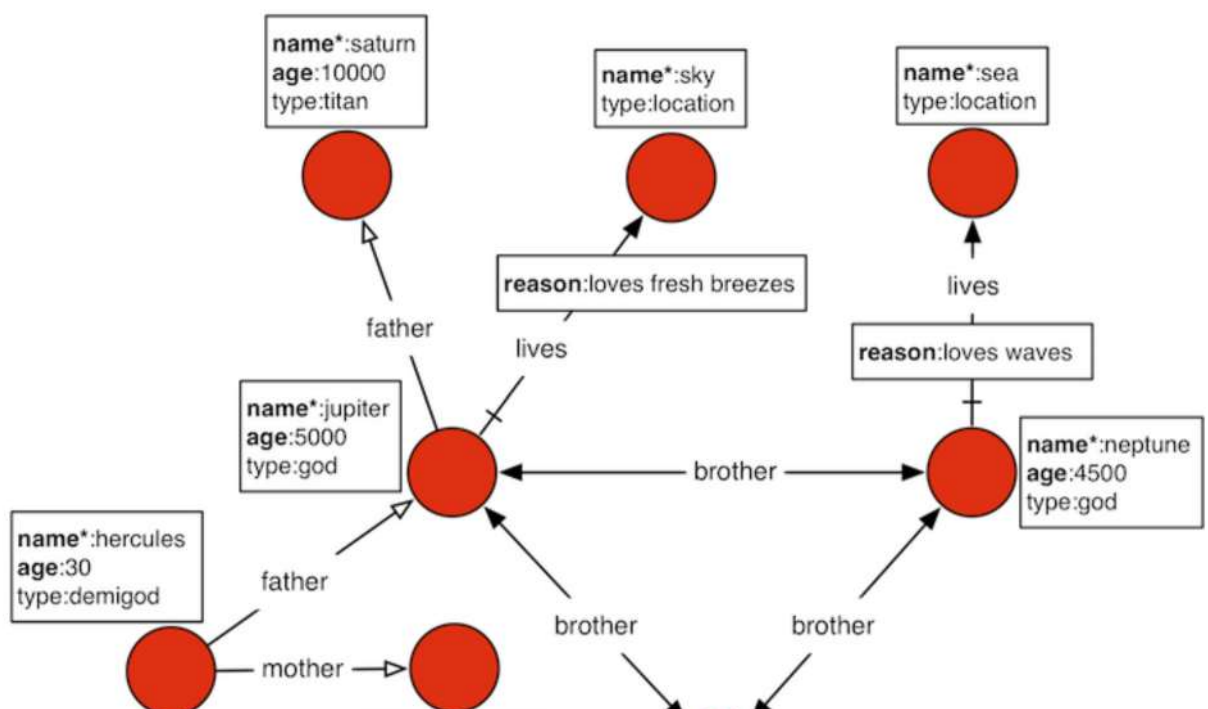
In the beginning, there existed two deities known as **Uranus** and **Gaia**. They gave birth to the **Titans** (a race of powerful beings). **Saturn**, Titan of time, set reality in motion. Ultimately, time yielded the existence of the sky, the sea, and the end of life—death. To rule these notions, Saturn had three sons: **Jupiter** (sky), **Neptune** (sea), and **Pluto** (underworld). The son's of Saturn were not Titans, but a race of seemingly less powerful deities known the world over as the Gods. Fearful that his sons would overthrow him, Saturn devoured them and imprisoned them in his stomach. This caused a **great war** between the Titans and Gods. Ultimately, the Gods won and Jupiter took the throne as leader of the Gods.

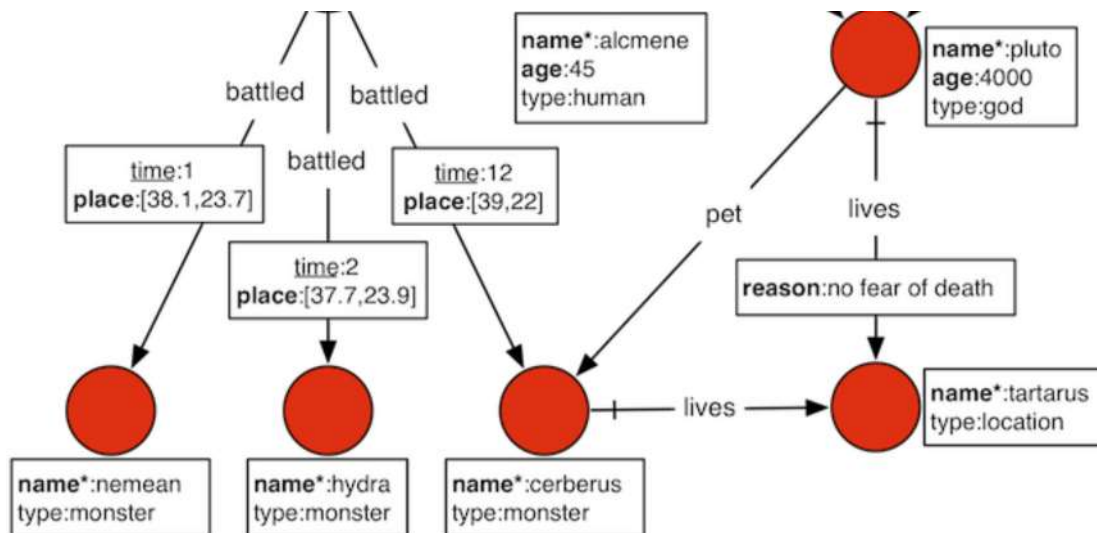
Table of Contents

- 3.1. Downloading Titan and Running the Gremlin Shell
- 3.2. Loading the Graph of the Gods Into Titan
- 3.3. Global Graph Indices
 - 3.3.1. Graph Traversal Examples
 - 3.3.2. More Complex Graph Traversal Examples

The examples in this section make extensive use of a toy graph distributed with Titan called *The Graph of the Gods*. This graph is diagrammed below. The abstract data model is known as a **Property Graph Model** and this particular instance describes the relationships between the beings and places of the Roman pantheon. Moreover, special text and symbol modifiers in the diagram (e.g. bold, underline, etc.) denote different schematics/typings in the graph.

Figure 3.1. Graph of the Gods





visual symbol	meaning
bold key	a graph indexed key
bold key with star	a graph indexed key that must have a unique value
underlined key	a vertex-centric indexed key
hollow-head edge	a functional/unique edge (no duplicates)
tail-crossed edge	a unidirectional edge (can only traverse in one direction)

3.1. Downloading Titan and Running the Gremlin Shell

Unbeknownst to the Gods, there still lived one Titan. This Titan can not be seen, has no name, and is only apparent in the fact that reality exists. Upon the shoulders of this lost Titan, all of reality hangs together in an undulating web of relations.

Titan can be downloaded from the [Downloads](#) section of the project repository. Once retrieved and unpacked, a [Gremlin](#) terminal can be started. The Gremlin [REPL](#) (i.e. interactive shell) is distributed with Titan and differs slightly from the main [TinkerPop](#) Gremlin distribution in that it comes preloaded with Titan-specific `imports` and helper methods. In the example below, `titan.zip` is used, however, be sure to unzip the zip-file that was downloaded.

```

$ unzip titan.zip
Archive:  titan.zip
  creating: titan/
...
$ cd titan
$ bin/gremlin.sh

```

RootFile

```

\,,,/
(o o)
-----oOOo-(_)oOOo-----
gremlin>

```

The Gremlin terminal is a [Groovy](#) shell. Groovy is a superset of Java that has various shorthand notations that make interactive programming easier. Likewise Gremlin is a superset of Groovy with various shorthand notations that make graph traversals easy. The basic examples below demonstrate handling numbers, strings, and maps. The remainder of the tutorial will discuss graph-specific

constructs.

RootFile pushing code
to the stdin of the
program

```
gremlin> 100-10
==>90
gremlin> "Titan:" + " The Rise of Big Graph Data"
==>Titan: The Rise of Big Graph Data
gremlin> [name:'aurelius',vocation:['philosopher','emperor']]
==>name=aurelius
==>vocation=[philosopher, emperor]
```



Tip

Refer to [GremlinDocs](#), [SQL2Gremlin](#), and [the Gremlin Wiki](#) for more information about using Gremlin.

3.2. Loading the Graph of the Gods Into Titan

The example below will open a Titan graph instance and load *The Graph of the Gods* dataset diagrammed above. `TitanFactory` provides a set of static `open` methods, each of which takes a configuration as its argument and returns a graph instance. This tutorial calls one of these `open` methods on a configuration that uses the [BerkeleyDB](#) storage backend and the [Elasticsearch](#) index backend, then loads *The Graph of the Gods* using the helper class `GraphOfTheGodsFactory`. This section skips over the configuration details, but additional information about storage backends, index backends, and their configuration are available in [Part III, "Storage Backends"](#), [Part IV, "Index Backends"](#), and [Chapter 12, Configuration Reference](#).

```
gremlin> g = TitanFactory.open('conf/titan-berkeleydb-es.properties')
==>titangraph[berkeleyje:../db/berkeley]
gremlin> GraphOfTheGodsFactory.load(g)
==>null
```

same

The `TitanFactory.open()` and `GraphOfTheGodsFactory.load()` methods do the following to the newly constructed graph prior to returning it:

1. Creates a collection of global and vertex-centric indices on the graph.
2. Adds all the vertices to the graph along with their properties.
3. Adds all the edges to the graph along with their properties.

Please see the [GraphOfTheGodsFactory source code](#) for details.

For those using Titan/Cassandra (or Titan/HBase), be sure to make use of `conf/titan-cassandra-es.properties` (or `conf/titan-hbase-es.properties`) and `GraphOfTheGodsFactory.load()`.

```
gremlin> g = TitanFactory.open('conf/titan-cassandra-es.properties')
==>titangraph[cassandrathrift:127.0.0.1]
gremlin> GraphOfTheGodsFactory.load(g)
==>null
```

same

3.3. Global Graph Indices

The typical pattern for accessing data in a graph database is to first locate the entry point into the graph using a graph index. That entry point is an element (or set of elements) — i.e. a vertex or edge. From the entry elements, a Gremlin path description describes how to traverse to other elements in the graph via the explicit graph structure.

Given that there is a unique index on `name` property, the Saturn vertex can be retrieved. The property map (i.e. the key/value pairs of Saturn) can then be examined. As demonstrated, the Saturn vertex has a `name` of "saturn," an `age` of 10000, and a `type` of "titan." The grandchild of Saturn can be retrieved with a traversal that expresses: "Who is Saturn's grandchild?" (the inverse of "father" is "child"). The result is Hercules.

```
gremlin> saturn = g.V.has('name','saturn').next()
==>v[256]
gremlin> saturn.map()
==>name=saturn
==>age=10000
gremlin> saturn.in('father').in('father').name
==>hercules
```

same

The property `place` is also in a graph index. The property `place` is an edge property. Therefore, Titan can index edges in a graph index. It is possible to query *The Graph of the Gods* for all events that have happened within 50 kilometers of **Athens** (latitude:37.97 and long:23.72). Then, given that information, which vertices were involved in those events.

```
gremlin> g.query().has('place',WITHIN,Geoshape.circle(37.97,23.72,50)).edge
==>e[75s-16o-iz9-1z4][1536-battled->2560]
==>e[74w-16o-iz9-1s0][1536-battled->2304]
gremlin> g.query().has('place',WITHIN,Geoshape.circle(37.97,23.72,50)).edge
    it.bothV.name.next(2)
}
==>[hercules, hydra]
==>[hercules, nemean]
```

same


Graph indices are one type of index structure in Titan. Graph indices are accessible via the `Graph.query()` method. The second aspect of indexing in Titan is known as vertex-centric indices. Vertex-centric indices are accessible via the `Vertex.query()` method. Vertex-centric indices are described later.

3.3.1. Graph Traversal Examples

Hercules, son of Jupiter and **Alcmene**, bore super human strength. Hercules was a **Demigod** because his father was a god and his mother was a human. **Juno**, wife of Jupiter, was furious with Jupiter's infidelity. In revenge, she blinded Hercules with temporary insanity and caused him to kill his wife and children. To atone for the slaying, Hercules was ordered by the **Oracle of Delphi** to serve **Eurystheus**. Eurystheus appointed Hercules to 12 labors.


In the previous section, it was demonstrated that Saturn's grandchild was Hercules. This can be expressed using a `loop`. In essence, Hercules is the vertex that is 2-steps away from Saturn along the `in('father')` path.

```
gremlin> hercules = saturn.as('x').in('father').loop('x'){it.loops < 3}.next()
==>v[1536]
```




Hercules is a demigod. To prove that Hercules is half human and half god, his parent's origins must be examined. It is possible to traverse from the Hercules vertex to his mother and father. Finally, it is possible to determine the type of each of them — yielding "god" and "human."

```
gremlin> hercules.out('father','mother')
==>v[1024]
==>v[1792]
gremlin> hercules.out('father','mother').name
==>jupiter
==>alcmene
gremlin> hercules.out('father','mother').*.getVertexLabel()
==>god
==>human
gremlin> hercules.getVertexLabel()
==>demigod
```



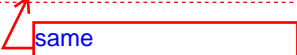
The examples thus far have been with respect to the genetic lines of the various actors in the Roman pantheon. The **Property Graph Model** is expressive enough to represent multiple types of things and relationships. In this way, *The Graph of the Gods* also identifies Hercules' various heroic exploits --- his famous 12 labors. In the previous section, it was discovered that Hercules was involved in two battles near Athens. It is possible to explore these events by traversing `battled` edges out of the Hercules vertex.

```
gremlin> hercules.out('battled')
==>v[2304]
==>v[2560]
==>v[2816]
gremlin> hercules.out('battled').map
==>{name=nemean}
==>{name=hydra}
==>{name=cerberus}
gremlin> hercules.outE('battled').has('time',T.gt,1).inV.name
==>cerberus
==>hydra
```



The edge property `time` on `battled` edges is indexed by the vertex-centric indices of a vertex. Retrieving `battled` edges incident to Hercules according to a constraint/filter on `time` is faster than doing a linear scan of all edges and filtering (typically $O(\log n)$, where n is the number incident edges). Gremlin is intelligent enough to use vertex-centric indices when available. A `toString()` of a Gremlin expression shows the underlying **query pipeline**.

```
gremlin> hercules.outE('battled').has('time',T.gt,1).inV.name.toString()
==>[StartPipe, VertexQueryPipe(out,[battled],has,edge), IdentityPipe, InVer]
```




3.3.2. More Complex Graph Traversal Examples

In the depths of Tartarus lives Pluto. His relationship with Hercules was strained by the

fact that Hercules battled his pet, Cerberus. However, Hercules is his nephew — how should he make Hercules pay for his insolence?


The Gremlin traversals below provide more examples over *The Graph of the Gods*. The explanation of each traversal is provided in the prior line as a `//` comment.

3.3.2.1. Cohabiters of Tartarus




```
gremlin> pluto = g.V('name','pluto').next()
==>v[2048]
gremlin> // who are pluto's cohabitants?
gremlin> pluto.out('lives').in('lives').name
==>pluto
==>cerberus
gremlin> // pluto can't be his own cohabitant
gremlin> pluto.out('lives').in('lives').except([pluto]).name
==>cerberus
gremlin> pluto.as('x').out('lives').in('lives').except('x').name
==>cerberus
```

3.3.2.2. Pluto's Brothers



```
gremlin> // where do pluto's brothers live?
gremlin> pluto.out('brother').out('lives').name
==>sky
==>sea
gremlin> // which brother lives in which place?
gremlin> pluto.out('brother').as('god').out('lives').as('place').select
==>[god:v[1024], place:v[512]]
==>[god:v[1280], place:v[768]]
gremlin> // what is the name of the brother and the name of the place?
gremlin> pluto.out('brother').as('god').out('lives').as('place').select{it}
==>[god:jupiter, place:sky]
==>[god:neptune, place:sea]
```

Finally, Pluto lives in Tartarus because he shows no concern for death. His brothers, on the other hand, chose their locations based upon their love for certain qualities of those locations.



```
gremlin> pluto.outE('lives').reason
==>no fear of death
gremlin> g.query().has('reason',CONTAINS,'loves').edges()
==>e[6xs-sg-m51-e8][1024-lives->512]
==>e[70g-zk-m51-lc][1280-lives->768]
gremlin> g.query().has('reason',CONTAINS,'loves').edges().collect{
  [it.outV.name.next(),it.reason,it.inV.name.next()]
}
==>[jupiter, loves fresh breezes, sky]
==>[neptune, loves waves, sea]
```

[Prev](#)
Chapter 2. Architectural Overview

[Up](#)
[Home](#)

[Next](#)
Part II. Titan Basics

Copyright 2014 All Rights Reserved - Titan is a trademark of Aurelius LLC.

Titan support provided by Aurelius.

Cassandra, HBase, and Hadoop are trademarks of the Apache Software Foundation.

Berkeley DB and Berkeley DB Java Edition are trademarks of Oracle.

Documentation generated with AsciiDoc, AsciiDoctor, DocBook, and Saxon.