SD Erlang Tutorial – Unix/Mac

SD Erlang building and installation

1. You should install SD Erlang/OTP from https://github.com/release-project/otp/tree/sd-erlang-tutorial . Instructions for building and installing Erlang OTP is provided here https://github.com/release-project/otp/blob/sd-erlang-tutorial/HOWTO/INSTALL.md
2. Check that command `erl` corresponds to SD Erlang (in case you have multiple Erlang versions) by running

   ```
   $ which erl
   ```

   If you get "`erl: Command not found`" or the path does not correspond to the SD Erlang then you need to provide the path explicitly, for example,

   ```
   $ ~/otp_src_R15B03/bin/erl
   ```

3. Check that SD Erlang works by starting a node, i.e. either

   ```
   $ erl
   ```

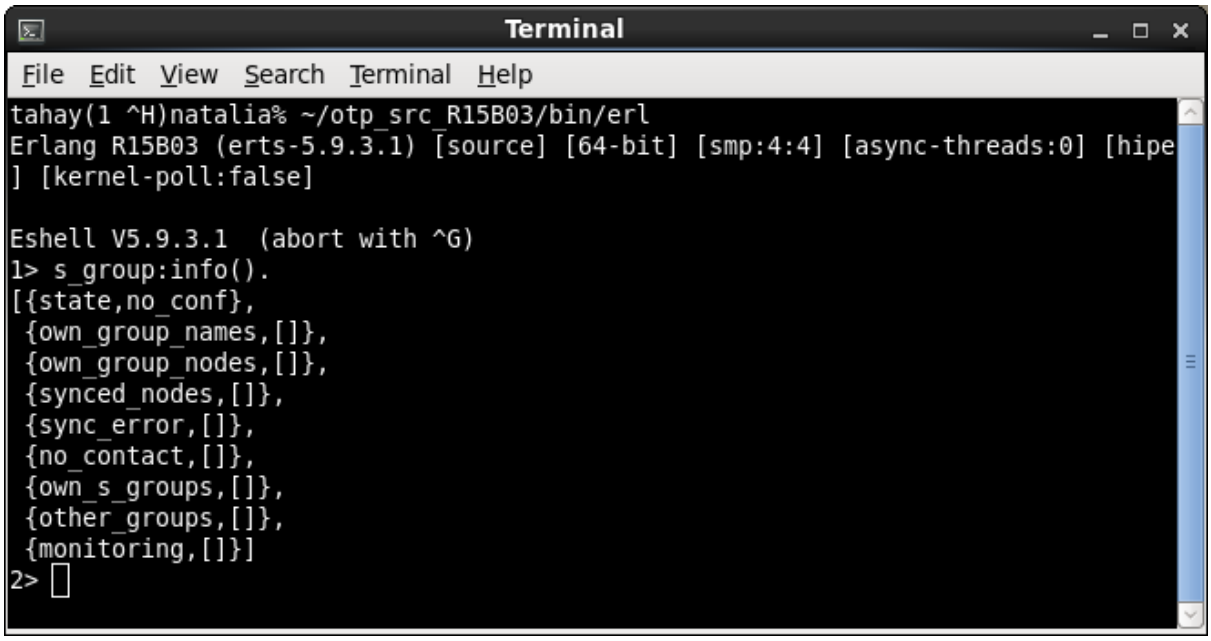
   or

   ```
   $ ~/otp_src_R15B03/bin/erl
   ```

   whichever works for you. Now execute the function

   ```
   > s_group:info().
   ```

   You should get something like this the state having no s_group configuration and the node belongs to no s_groups



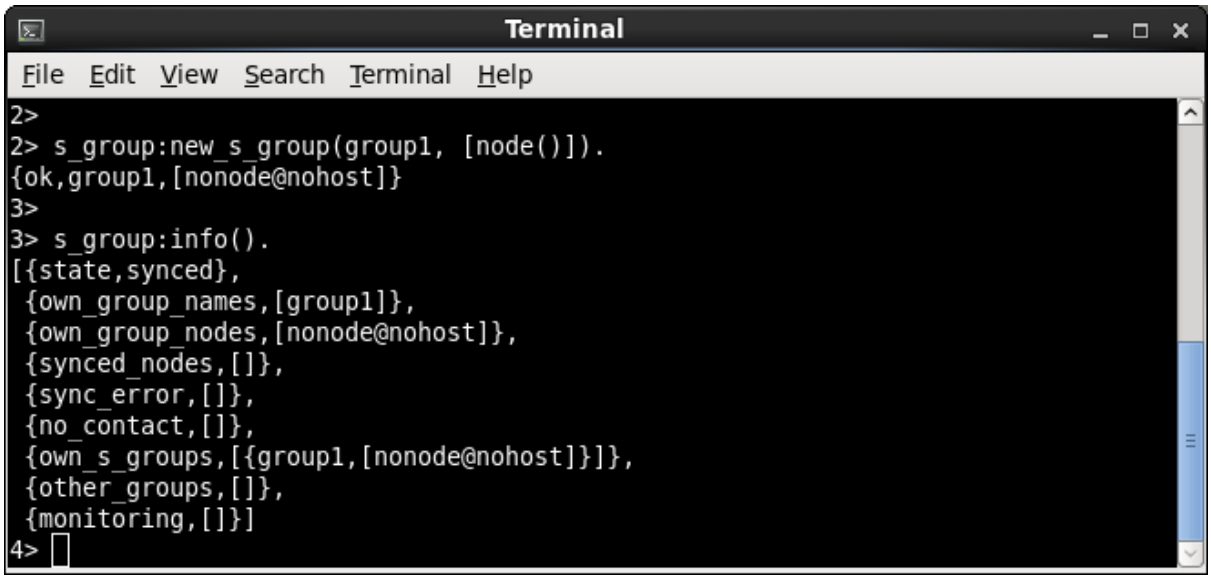
Now we can create an s_group of one node

```
> s_group:new_s_group(group1, [node()]).
```

And check how s_group information changed:

```
> s_group:info().
```

You should get something like this. The state is synchronised and the node belongs to s_group `group1`



4. Now we know that SD Erlang works we can create some more interesting s_groups.

Exercise 1: Creating s_groups

1. Create a folder, for example, `sd-erlang-tutorial` in your home directory where you'll keep your Erlang modules, and go to that directory

```
$ cd sd-erlang-tutorial
```

2. Create file `grouping.erl` and insert the following code

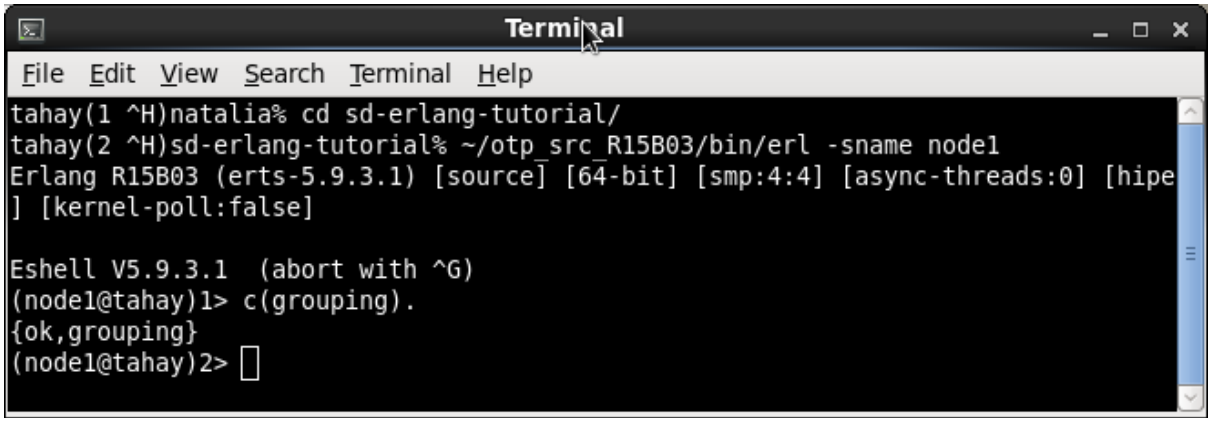
```
-module(grouping).

-compile([export_all]).

create_new_sgroup(MasterPid, SGroupName, Nodes) ->
    Result = s_group:new_s_group(SGroupName, Nodes),
    MasterPid ! Result.
```

`create_new_sgroup/3` function creates an s_group that consists of given nodes `Nodes` and has name `SGroupName`, and then returns a result by sending a message to a given pid `MasterPid`.

Hint: You can paste the text above from the electronic version of this tutorial at https://github.com/release-project/benchmarks into your favourite editor.

Start Erlang node `node1` in `sd-erlang-tutorial` directory and compile the code, e.g.

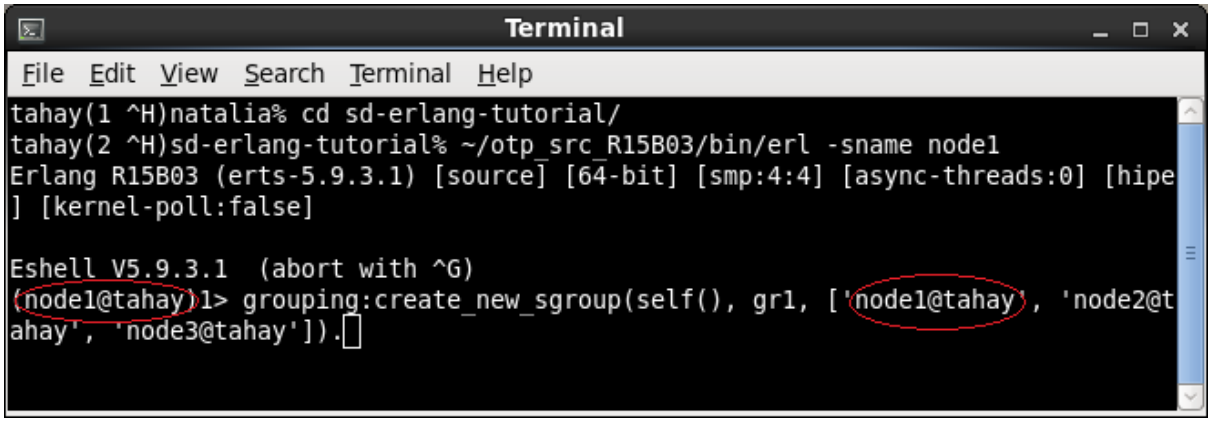
```
$ ~/otp_src_R15B03/bin/erl -sname node1
```



3. Now let us start two more nodes in `sd-erlang-tutorial` directory, i.e.
   ```
   $ ~/otp_src_R15B03/bin/erl -sname node2
   $ ~/otp_src_R15B03/bin/erl -sname node3
   ```

4. On `node1` run the following function:
   ```
   > grouping:create_new_sgroup(self(), gr1, ['node1@tahay',
   'node2@tahay', 'node3@tahay']).
   ```

   Where `'node1@tahay'`, `'node2@tahay'`, and `'node3@tahay'` are full names of your nodes (in our case the host name is `@tahay` but you may have a different host name), e.g.



5. Then if you run function to ensure that all messages are processed
   ```
   > flush().
   ```
   and get
   ```
   Shell got {ok,gr1,[node1@tahay,node2@tahay,node3@tahay]}
   ```
   the function works correctly and you have successfully created s_group `gr1`.

6. As before to check node s_group information we use `s_group:info()` function

7. Now let us add `create_sgroup/2` function in `grouping.erl` that creates an s_group on a remote node. So `grouping.erl` will look as follows (here, code from [bullet 2] is highlighted in blue).

```
-module(grouping).

-compile([export_all]).

create_sgroup(SGroupName, [Node|Nodes]) ->
    spawn(Node, ?MODULE, create_new_sgroup, [self(), SGroupName,
[Node|Nodes]]),
    receive
        {ok, SGroupName, Nodes1} ->
       {SGroupName, Nodes1};
   Response ->
       Response
    end.

create_new_sgroup(MasterPid, SGroupName, Nodes) ->
    Result = s_group:new_s_group(SGroupName, Nodes),
    MasterPid ! Result.
```

8. Again we compile the code as in [bullet 3], then start two more new nodes `node4` and `node5` in `sd-erlang-tutorial` directory, i.e.

```
$ ~/otp_src_R15B03/bin/erl -sname node4
$ ~/otp_src_R15B03/bin/erl -sname node5
```

   and then run function `create_sgroup/2`, i.e.

```
> grouping:create_sgroup(gr2, ['node3@tahay', 'node4@tahay',
'node5@tahay']).
```

9. To register a name in an s_group `s_group:register_name/3` function is used, for example, on nodes `node2`, `node3`, and `node4`

```
node2> s_group:register_name(gr1,name2,self()).
node3> s_group:register_name(gr1,name3,self()).
node4> s_group:register_name(gr2,name4,self()).
```

   We can also unregister names, for example,

```
node1> s_group:unregister_name(gr1,name3).
```

   And re-register names, for example,

```
node5> s_group:re_register_name(gr2,name4,self()).
```

10. Run the following functions on nodes `node2, node3` and `node4`, and compare results. Name registration functions:

```
%% returns a list of registered names from all s_groups the node
is a member of
> global:registered_names().
```

```
%% returns a list of names registered in the given s_group
> s_group:registered_names({s_group, gr1}).
```

```
%% returns the pid of a process registered in s_group gr2 under
name 'name4' (returns 'undefind' if the name is not registered)
> global:whereis_name(gr2, name4).
```

S_group information functions:

```
%% returns a list of {SGroupName, Nodes} tuples of s_groups the
node belongs to
> s_group:own_s_groups().
```

```
%% returns a list of nodes from all s_groups the node belongs to
> s_group:own_nodes().
```

Connections:

```
> nodes(connected).
```

11. Next let us add `create_two_s_groups/2` function in `grouping.erl` that creates two s_groups using a provided list of nodes `ListOfNodes` . The first s_group contains `NumberOfNodesInGroupOne` nodes, and the second s_group contains the rest of the nodes from the `ListOfNodes`. So `grouping.erl` will look as follows (again, code from [bullet 8] is highlighted in blue).

```
-module(grouping_3).

-compile([export_all]).

create_two_s_groups(NumberOfNodesInGroupOne, ListOfNodes) ->
    case NumberOfNodesInGroupOne >= length(ListOfNodes) of
   true ->
       not_enough_nodes;
   _Else ->
       create_two_s_groups_do(NumberOfNodesInGroupOne, ListOfNodes)
    end.

create_two_s_groups_do(NumberOfNodesInGroupOne, ListsOfNodes) ->
    {Nodes1, Nodes2} = lists:split(NumberOfNodesInGroupOne,
ListsOfNodes),
    SGroup1 = create_sgroup(gr3, Nodes1),
    SGroup2 = create_sgroup(gr4, Nodes2),
    [SGroup1, SGroup2].

create_sgroup(SGroupName, [Node|Nodes]) ->
    spawn(Node, ?MODULE, create_new_sgroup, [self(), SGroupName,
[Node|Nodes]]),
    receive
   {ok, SGroupName, Nodes1} ->
       {SGroupName, Nodes1};
   Response ->
       Response
    end.

create_new_sgroup(MasterPid, SGroupName, Nodes) ->
    Result = s_group:new_s_group(SGroupName, Nodes),
```

```
    MasterPid ! Result.
```

12. We compile the code as in [bullet 3], then start two more new nodes `node6` and `node7` in `sd-erlang-tutorial` directory, i.e.

```
$ ~/otp_src_R15B03/bin/erl -sname node6
$ ~/otp_src_R15B03/bin/erl -sname node7
```

    and then run function `create_two_s_groups/2`, i.e.

```
> create_two_s_groups(3, ['node1@tahay', 'node2@tahay',
'node3@tahay', 'node4@tahay', 'node5@tahay', 'node6@tahay',
'node7@tahay']).
```

13. To see the effect of s_groups on node connections and name registration run functions from [bullets 10 and 11].

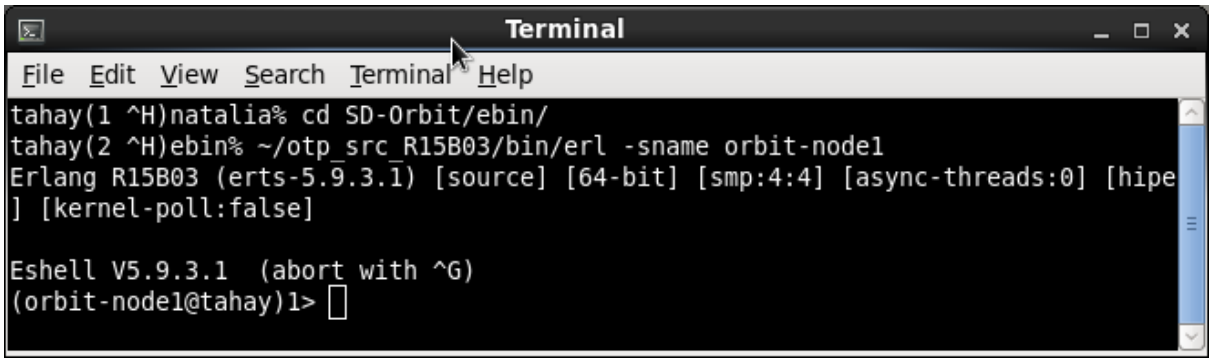
Exercise 2: Creating Orbit-like S_groups (optional)

Using the `grouping.erl` module from Exercise 1 [bullet 12] example provided in Exercise 1, write a function that creates an arbitrary number of s_groups. The input data is a number of s_groups and a list of nodes.

Exercise 3: Compiling SD Erlang Orbit

1. Copy the SD Erlang Orbit from https://github.com/release-project/benchmarks (SD-Orbit)
2. Assume that your `SD-Orbit` directory is in your home directory
3. Open a terminal in SD-Orbit directory and run `./compile` command
4. Go to SD-Orbit/ebin directory and start four worker nodes:

```
$ ~/otp_src_R15B03/bin/erl -sname orbit-node1
$ ~/otp_src_R15B03/bin/erl -sname orbit-node2
$ ~/otp_src_R15B03/bin/erl -sname orbit-node3
$ ~/otp_src_R15B03/bin/erl -sname orbit-node4
```



5. Now using your favourite editor open `init_bench.erl` in `SD-Orbit/src`
   In function `main()` modify `G, N, P, NumGateways`, and `G_size` parameters, for example, as follows:

```
G = fun bench:g12345/1,
N = 10000,            %% calculates Orbit for 0..N
P = 8,                %% Number of worker processes on each node
NumGateways = 40,     %% Number of gateway processes on each
                      %% sub-master node
G_size = 2,           %% Number of nodes in each s_group
```

6. Modify the list of nodes in SD-Orbit/bench.config file to include your worker nodes, e.g.

```
['nonode@nohost', 'orbit-node1@tahay', ' orbit-node2@tahay', '
orbit-node3@tahay', ' orbit-node4@tahay']
```

   Here '`nonode@nohost`' is the master node, and the rest are worker nodes and sub-master nodes.

7. Go back to the terminal you opened in [bullet 3] and run the benchmark using command

```
$ ./run
```