## GIT support

The *sshd-git* artifact contains both client and server-side command factories for issuing and handling some *git* commands. The code is based on [JGit](https://github.com/eclipse/jgit) and iteracts with it smoothly.

### Client-side

This module provides SSHD-based replacements for the SSH and SFTP transports used by the JGIT client - see GitSshdSessionFactory - it can be used as a drop-in replacement for the [JSCH](http://www.jcraft.com/jsch/) based built-in session factory provided by *jgit*. In this context, it is worth noting that the GitSshdSessionFactory has been tailored so as to provide flexible control over which SshClient instance to use, and even which ClientSession. The default instance allocates a **new** client every time a new GitSshdSession is created - which is started and stopped as necessary. However, this can be pretty wasteful, so if one intends to issue several commands that access GIT repositories via SSH, one should maintain a **single** client instance and re-use it:

SshClient client = ...create and setup the client...  
 try {  
 client.start();  
  
 GitSshdSessionFactory sshdFactory = new GitSshdSessionFactory(client); // re-use the same client for all SSH sessions  
 org.eclipse.jgit.transport.SshSessionFactory.setInstance(sshdFactory); // replace the JSCH-based factory  
  
 ... issue GIT commands that access remote repositories via SSH ....  
  
 } finally {  
 client.stop();  
 }

### Server-side

See GitPackCommandFactory and GitPgmCommandFactory - in order for the various commands to function correctly, they require a GitLocationResolver that is invoked in order to allow the user to decide which is the correct GIT repository root location for a given command. The resolver is provided with all the relevant details - including the command and server session through which the command was received:

GitLocationResolver resolver = (cmd, session, fs) -> ...consult some code - perhaps based on the authenticated username...  
 sshd.setCommandFactory(new GitPackCommandFactory().withGitLocationResolver(resolver));

These command factories also accept a delegate to which non-*git* commands are routed:

sshd.setCommandFactory(new GitPackCommandFactory()  
 .withDelegate(new MyCommandFactory())  
 .withGitLocationResolver(resolver));  
  
 // Here is how it looks if SCP is also requested  
 sshd.setCommandFactory(new GitPackCommandFactory()  
 .withDelegate(new ScpCommandFactory()  
 .withDelegate(new MyCommandFactory()))  
 .withGitLocationResolver(resolver));  
  
 // or  
 sshd.setCommandFactory(new ScpCommandFactory()  
 .withDelegate(new GitPackCommandFactory()  
 .withDelegate(new MyCommandFactory())  
 .withGitLocationResolver(resolver)));  
  
 // or any other combination ...

as with all other built-in commands, the factories allow the user to provide an ExecutorService in order to control the spawned threads for servicing the commands. If none provided, an internal single-threaded "pool" is created ad-hoc and destroyed once the command execution is completed (regardless of whether successful or not):

sshd.setCommandFactory(new GitPackCommandFactory(resolver)  
 .withDelegate(new MyCommandFactory())  
 .withExecutorService(myService)  
 .withShutdownOnExit(false));