

Part 1

Summary

This report describes an automated build system implemented for the Eclipse ECF project. The system uses cruisecontrol and ant; it runs on an OSU machine. It checks out code from the Eclipse repository, performs a headless build, and uploads the files to the ECF downloads directory on `dev.eclipse.org`. In addition, extra builds are performed from a repository located on another OSU machine and made available from a web site hosted by that machine. The report also describes how the automated build system can be remotely administered and suggests future work.

Contents

Summary	1
The Goal	3
Defining the Goal	3
The Zeroth Stage	3
The First Stage	4
Download and install CruiseControl	5
Download and Install Ant	5
Testing that Ant Works	6
Testing that Cruisecontrol Works	6
Getting the Build under Cruisecontrol	6
Starting Up CruiseControl	7
The ECF Builds	7
Build Intervals and Quiet Periods	7
The ECF Daily Build	9
cc-build.xml	9
Sending Email	10
Uploading Files	14
Passphraseless Key Authentication	14
Making and Using the Key-Pair on Linux	14

ECF Autobuild System

<u>Using the Key-Pair on Windows.....</u>	<u>15</u>
<u>Uploading Daily Files.....</u>	<u>18</u>
<u>Making Files Available on eclipse.org.....</u>	<u>19</u>
<u>Keeping a Week's Worth of Dailies.....</u>	<u>21</u>
<u>More Detail on the Daily Builds.....</u>	<u>21</u>
<u>Second Stage.....</u>	<u>22</u>
<u>Setting Up the Cruisecontrol Web Reporting Tool.....</u>	<u>22</u>
<u>Using the Web Reporting Tool Remotely.....</u>	<u>22</u>
<u>Getting a Remote Desktop.....</u>	<u>22</u>
<u>Getting a Remote Browser.....</u>	<u>30</u>
<u>Third Stage.....</u>	<u>31</u>
<u>Fourth Stage.....</u>	<u>35</u>

The Goal

Our goal is to have an automated build system for the Eclipse ECF project.

Defining the Goal

What do we mean by an automated build system? I'm not going to try to describe our entire vision and then show it in operation. It has too much plumage for that.

Rather I'll describe our automated build system in stages. This reflects reality more closely because that's how we implemented it. In fact, some of the later stages have not been implemented.

The Zeroth Stage

In this stage we just wanted to understand the key terms.

PDE Builds. Just some background information about PDE builds. We're building ECF, which is an Eclipse pug-in. This is called a PDE build.

The most straightforward way is to build the plug-in from within the Eclipse IDE. But we want to be able to build from the command line. We don't want the Eclipse IDE to pop up. Once we can build from the command line, we can put the build in a script and then automate the script.

Headless Builds. Building from the command line is called a *headless build*. There is an important caveat. A headless build requires the file `startup.jar` which is no longer part of the Eclipse SDK distribution. So we grabbed an older one and put it in our own section of the repository.

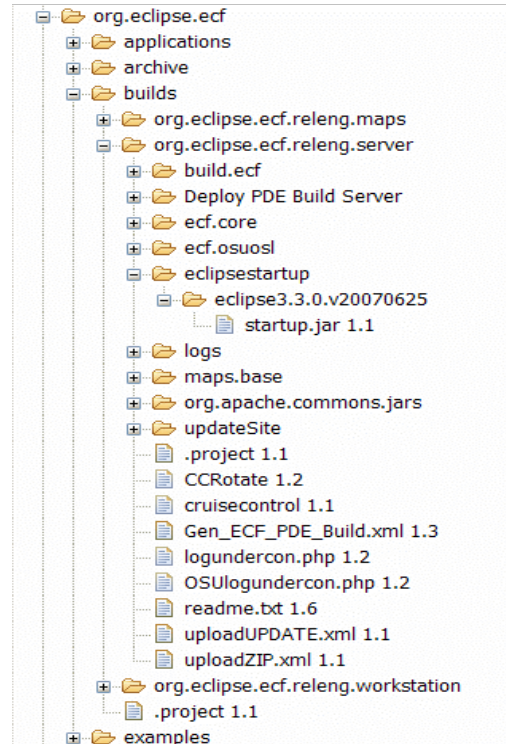
We are using the `startup.jar` from Eclipse 3.3.0.

Server vs. Workstation Builds. Notice that our repository makes a distinction between server and workstation builds. This is a distinction that has ceased to be important; we use the server build exclusively now.

The workstation build was one that we could start and run from the command line. You would provide arguments on the command line; but it turns out that you can do the very same thing with the workstation build.

Here's a figure that shows how our repository is structured. The repository is `/cvsroot/technology/org.eclipse.ecf`.

ECF Autobuild System



The First Stage

Here are the characteristics of our first stage.

- We do our builds on a machine in the opensource lab at OSU in Corvallis, Oregon called **ecf2**. It runs Suse 10.2 Linux.
- Our source code is in the Eclipse repository.
- We run cruisecontrol on **ecf2**. This cruisecontrol checks for changes in the repository every 30 minutes. If there is a change it performs a build. We call this the *auto build*. It sends out email to the list **ecf-build**. We do not save the auto builds.
- Cruisecontrol also performs a daily build if it detects a change in the repository. We upload the daily build to **dev.eclipse.org**, and it is available from our ECF web page. (Later we decided to perform a daily build whether or not there was a change in the repository.)

We chose cruisecontrol for no special reason. It's the warhorse of automated builds, but it is showing its age. We are certainly open to trying out other newer systems. We are dependent on cruisecontrol at this time, and so if we try out other methods, we'd have to do it as a prototype and make the switch after we determine it is working the way we want.

Here is how we implemented the first stage.

ECF Autobuild System

Download and install CruiseControl.

2.7 was the latest release when we started. There is now a 2.7.1, but we have not upgraded. Our first installation used 2.6.2.

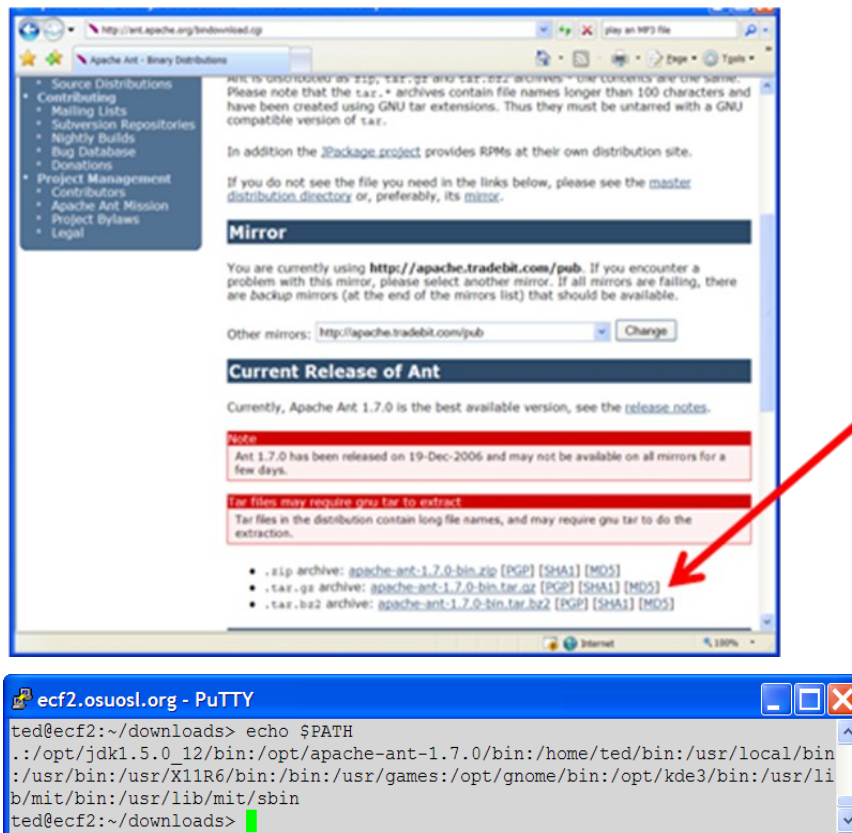
Go to <http://cruisecontrol.sourceforge.net/download.html> and download the file `cruisecontrol-src-2.7.zip`. Unzip it to `/opt`. Then, go into the directory `/opt/cruisecontrol-2.t/main` and run the script `build.sh`.

You get a lot of screen output that lasts for a couple of minutes. A `dist` directory is created under `main`. To run `cruisecontrol`, run the script `cruisecontrol.sh` under `main/bin`.

Download and Install Ant

The ant that came with Suse 10.2 (downloaded with YaST) did not work for me. Most of it did, but the `scp` authentication part did not. However, the ant I downloaded from the ant site did work. It was a newer version too.

Go to <http://ant.apache.org>. Click on **Download->Binary Distributions..** Then, untar the file into `/opt`, and put `/opt/apache-ant-1.7/bin` in the path.



You can test by creating an antfile and running it. I liked to test ant by trying out a sample project from the book *Pragmatic Project Automation* by Mike Clark. I'm going to skip the details of that test here, but it's worth saying something. So here goes.

ECF Autobuild System

Testing that Ant Works

You make an antfile; I called it `cc-build.xml` like Mike Clark. You start it up as

```
ant -buildfile cc-build.xml
```

Note that `-f` and `-file` perform the same operation as `-buildfile`. The file `cc-build.xml` can be very simple. Note that the example below has one project that runs the target called `build`. It deletes a directory called `dms` (this stands for Document Management System and comes from Mike Clark's book). This checks out (from CVS) the project `dms`. It uses the repository defined in the `CVSROOT` environment variable.

```
<project name="cc-build" default="build" basedir="/home/ted/builds/checkout">
  <target name="build">
    <delete dir="dms" />
    <cvs command="co dms" />
    <ant antfile="build.xml" dir="dms" target="test" />
  </target>
</project>
```

The file `cc-build.xml` calls the target `test` in another antfile called `build.xml`. I'm not going to reproduce the details of `build.xml` here, but the target `test` depends on the target `compile-tests`, which depends on the target `compile`. And the target `compile` calls the `ant` task `<javac/>` which does the compilation.

Testing that Cruisecontrol Works

That last build was run under ant, not cruisecontrol. Let's just see what's necessary to get it under cruisecontrol.

Getting the Build under Cruisecontrol

You need a file called `config.xml`. Well, that's the default name, and we might as well use it. Here's a minimal `config.xml`.

```
<cruisecontrol>
  <project name="dms" buildafterfailed="false">

    <listeners>
      <currentbuildstatuslistener
        file="logs/dms/DMScurrentbuildstatus.txt" />
    </listeners>

    <modificationset quietperiod="30">
      <cvs localworkingcopy="checkout/dms" />
    </modificationset>

    <schedule interval="300">
      <ant buildfile="cc-build.xml" target="build" />
    </schedule>

  </project>
</cruisecontrol>
```

ECF Autobuild System

Note the `<listeners/>` element. This replaces the obsolete

```
<bootstrapper>
  <currentbuildstatusbootstrapper/>
</bootstrapper>
```

and

```
<publisher>
  <currentbuildstatuspublisher/>
</ publisher >.
```

Starting Up CruiseControl

Start up cruisecontrol by just running the `cruisecontrol.sh` script. Give it options to identify the config file, specify the port for the JMX console, and specify the port for remote method invocation (rmi). The values below are the default values, but I like to list them anyway. More on these ports later.

```
/opt/cruisecontrol-2.7/main/bin/cruisecontrol.sh -configfile config.xml -port
8000 -rmiport 1099'
```

The ECF Builds

We do an auto build and a daily build. Each is a separate cruisecontrol project.

```
<!-- PROJECT ECF This is the Auto build -->
  <project name="ecf" buildafterfailed="false">
    .
    .
  </project>
<!-- PROJECT ECFDAILY This is the Daily build -->
  <project name="ecfDaily" buildafterfailed="false">
    .
    .
  </project>
```

Each of these builds consists of three zip files. More about what these zip files are and how to make them later. Right now, think about how to configure the scheduling.

Build Intervals and Quiet Periods

We want to set cruisecontrol to consider an auto build every 30 minutes. Note the sample `config.xml` listed above and the elements `<modificationset/>` and `<schedule/>`.

The `<modificationset/>` is just what you think it is. It's the location of the checked-out files that you want to build. Cruisecontrol monitors these files to see if they change in the CVS repository. The `<schedule/>` element specifies how often cruisecontrol considers doing a build. The units are seconds ... 30 minutes are 1800 seconds, so for auto builds, set the interval to 1800.

```
<modificationset quietperiod="300">
  <cvsw localworkingcopy="{localcopy}" />
</modificationset>

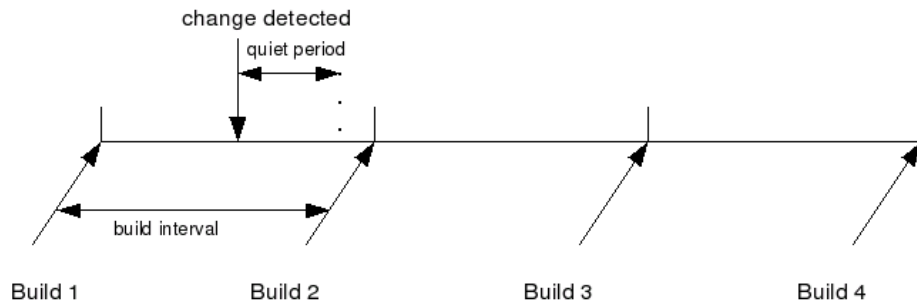
<schedule interval="1800">
```

ECF Autobuild System

```
<ant buildfile="cc-build.xml" target="ecf.build" >
  <property name="mapVTag" value="HEAD" />
  <property name="feature" value="ecf.core" />
  <property name="buildIdentifier" value="false" />
  <property name="buildType" value="A" />
  <property name="genFVSuffix" value="true" />
</ant>
</schedule>
```

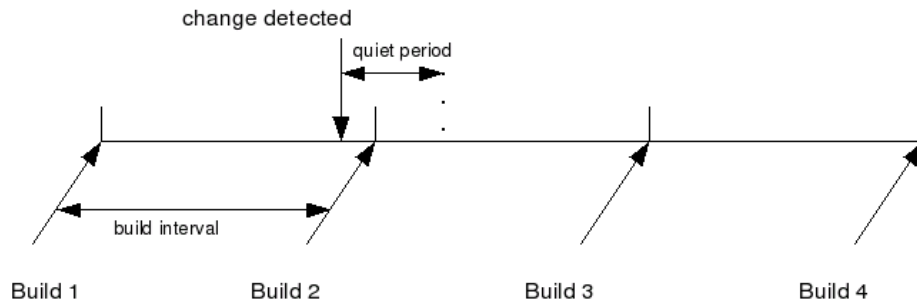
What about the quietperiod? The CVS repository must be quiet for the quietperiod before a build is attempted. If it is not, then the next scheduled build is skipped. The next two figures illustrate what I mean.

Consider a series of builds ... Build 1, Build 2, Build 3, Build 4, each separated by the build interval. In the first figure, the quiet period starts after a CVS change is detected and before the build interval between Build 1 and Build 2 ends. Hence, Build 2 occurs.



The CVS repository must be quiet for the quiet period before a build occurs. In the above case, Build 2 occurs.

In the next figure, the build interval between Build 1 and Build 2 ends before the quiet period terminates. Hence, Build 2 does not occur.



The CVS repository must be quiet for the quiet period before a build occurs. In the above case, Build 2 does not occur. It wants to occur during the quiet period, and that is not allowed. Build 2 is skipped and Build 3 occurs.

ECF Autobuild System

The ECF Daily Build

The last section showed a snippet from `config.xml` for the ECF auto build. Here are the equivalent lines for the Daily Build.

```
<modificationset quietperiod="300">
  <filesystem folder="/opt/ECFBUILDS/daily_kick" />
  <cvswork localworkingcopy="${localcopy}" />
</modificationset>

<schedule>
  <ant buildfile="cc-build.xml"
      time="1500"
      target="ecf.copy" >
    <property name="mapVTag" value="HEAD" />
    <property name="feature" value="ecf.core" />
    <property name="buildIdentifier" value="DAILY" />
    <property name="buildType" value="D" />
    <property name="genFVSSuffix" value="false" />
  </ant>
</schedule>
```

Notice that there isn't an interval attribute for the `<ant/>` element. Rather there is a time attribute. It specifies the time based on a 24-hour clock; 1500 is 3PM.

Note the properties. These pass values to the antfile `cc-build.xml`. If you wanted to run the build from the command line (not through cruisecontrol), use the `-D` option on the ant command line to pass these values.

```
ant -DmapVTag=HEAD -Dfeature=ecf.core -DbuildIdentifier=DAILY -DbuildType=D
    -DgenFVSSuffix=false -buildfile=cc-build.xml ecf.copy
```

Why is the target `ecf.copy` and not `ecf.core`? I'll explain that later. Let's look at `cc-build.xml` first.

cc-build.xml

The file `cc-build.xml` executes the `<java/>` task that performs the headless build. You know, you don't have to perform the headless build through ant. You could invoke `java` right from the command line and supply the appropriate options.

Here are the key lines in `cc-build.xml`.

```
<java classname="org.eclipse.core.launcher.Main"
      classpath="${eclipse.home}/startup.jar"
      fork="true" resultproperty="buildResult">
  <arg line ="-application org.eclipse.ant.core.antRunner
    -buildfile build.ecf.xml -logfile logs/${logfile}"/>
  <sysproperty key='feature' value='${feature}' />
  <sysproperty key='mapVersionTag' value='${mapVersionTag}' />
  <sysproperty key='buildType' value='${buildType}' />
  <sysproperty key='buildId' value='${buildId}' />
  <sysproperty key='buildLabel' value='${buildType}-${timestamp}' />
  <!-- Zip build folder -->
  <sysproperty key='baseLocation' value='${eclipse.home}' />
```

ECF Autobuild System

```
<sysproperty key='timestamp' value='${timestamp}' />
<sysproperty key='generateFeatureVersionSuffix' value='${genFVSuffix}' />
<sysproperty key='forceContextQualifier'
  value='${forceContextQualifier}' />
</java>
```

What's happening here? Well, you're running the eclipse launcher from the command line. You are launching the application `antRunner` (this is the `ant` that's part of Eclipse) and specifying the antfile `build.ecf.xml`.

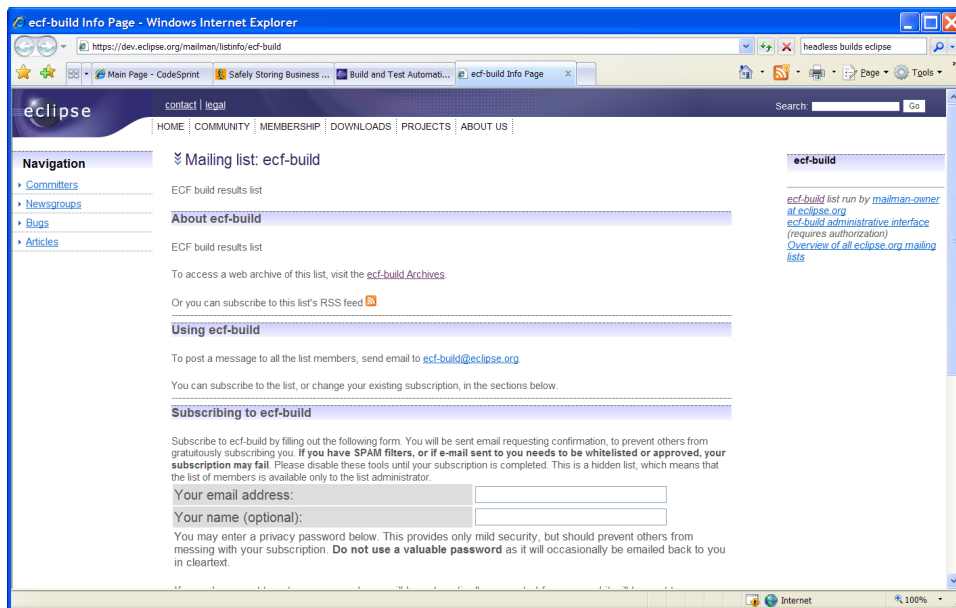
Remember, when you did a headless build, you invoked `java` from the command line or more likely put this java invocation in a script. What we're doing is instead of that script we use the antfile `cc-build.xml` and then arrange that cruisecontrol call this antfile. So the flow of control looks like the following.

```
cruisecontrol →
config.xml →
cc-build.xml →
java, eclipse, anrunner, build.ecf.xml →
build.xml
```

Sending Email

Cruisecontrol takes care of sending email in case of success or failure. This is controlled through `config.xml`.

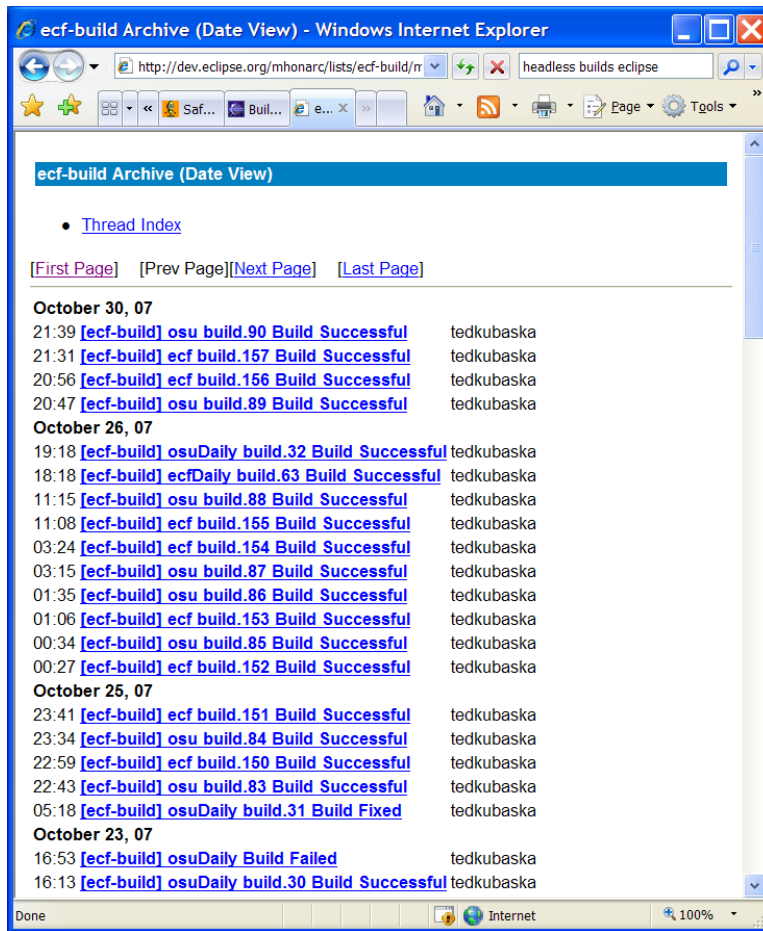
But first we set up a mailing list called `ecf-build`. To subscribe to this mailing list, go to <https://dev.eclipse.org/mailman/listinfo/ecf-build>. Click on **ECF Build Archives** to see an archive of mail messages.



Here is a sample mail message. If you click on the link in the message, right now you just get a link to the ECF downloads page (where the Daily Builds are available). Our intent is to provide

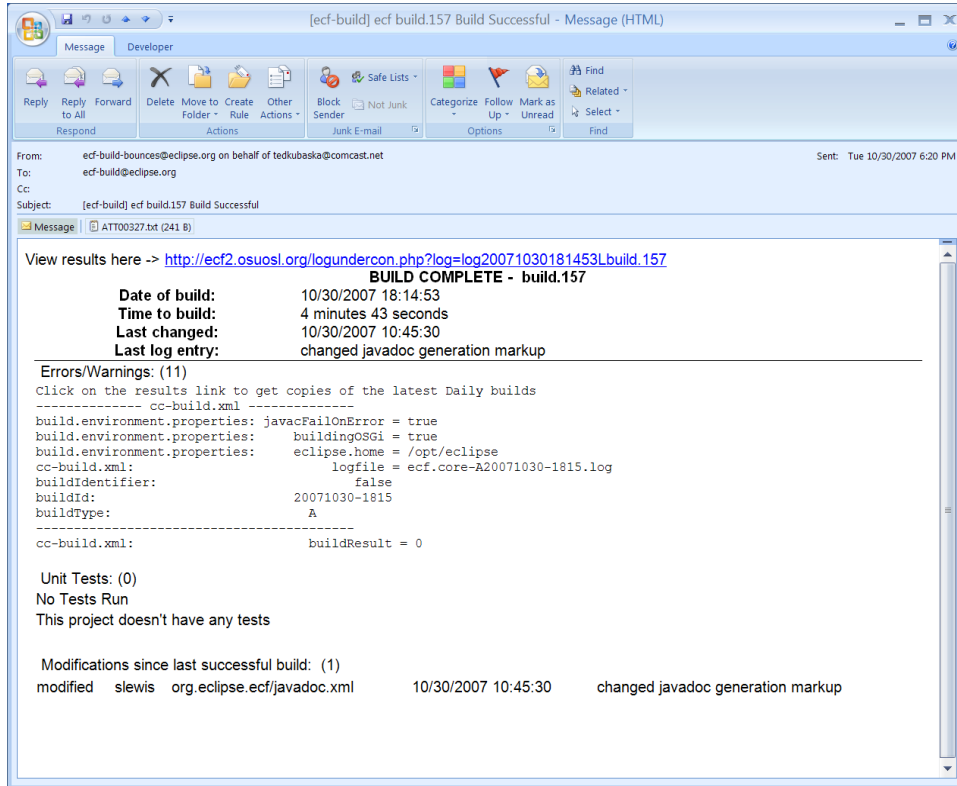
ECF Autobuild System

more build statistics on this page in the future.

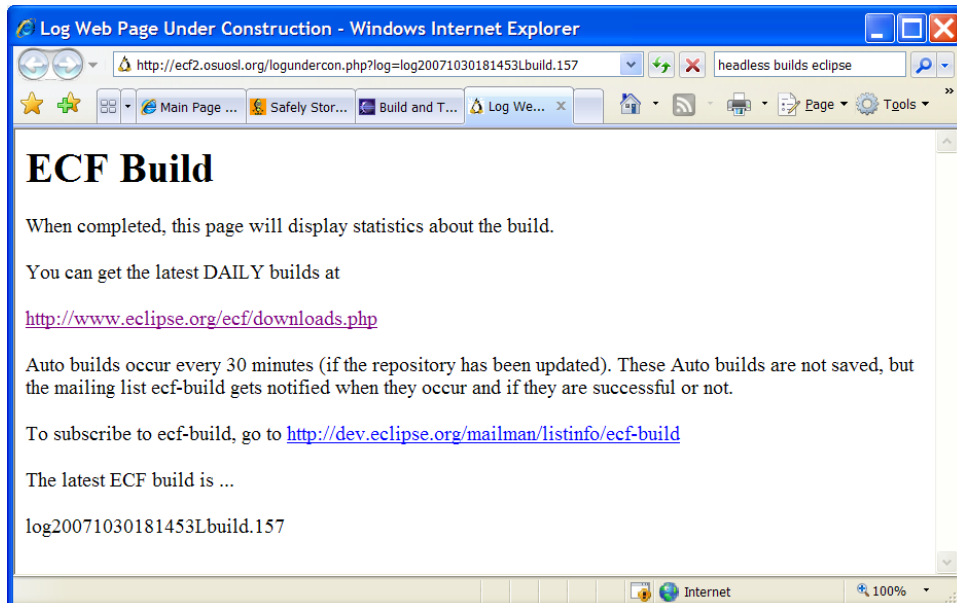


A typical mail message looks like the following:

ECF Autobuild System



If you click on the link next to “View results here,” you get the following:



To set up cruisecontrol to send mail to our mailing list, add the following to `config.xml`.

```
<!-- PROJECT ECF This is the Auto build -->  
<project name="ecf" buildafterfailed="false">
```

ECF Autobuild System

```
<listeners>
.
.
</listeners>
<modificationset ...>
.
.
</modificationset>
<schedule ...>
.
.
</schedule>
<publishers>
  <htmlmail mailhost="${smtpserver}"
    returnaddress="${mailsender}"
    skipusers="true"
    buildresultsurl="${resultswebpage}"
    css="/opt/cruisecontrol-2.7/docs/cruisecontrol.css"
    xsldir="/opt/cruisecontrol-2.7/reporting/jsp/webcontent/xsl">
    <map alias="ecf-build" address="ecf-build@eclipse.org"/>
    <always address="ecf-build" />
  </htmlmail>
</publishers>
</project>
```

A couple of comments about `${smtpserver}` and `${mailsender}` are in order. On `ecf2`, we define `${smtpserver}` to be `smtp.osuos1.org`, which is in fact OSU's smtp server.

On `ecf2`, what we put in for `${mailsender}` is irrelevant; it doesn't even have to be a real mail address. However, when I run `cruisecontrol` on my own workstation and talk to my own ISP (comcast), I must provide my comcast email address for comcast to accept my mail for transfer. Both `${smtpserver}` and `${mailsender}` are defined as properties up at the top of `config.xml`.

```
<property name="smtpserver" value="smtp.osuos1.org" />
<property name="mailsender" value="tedkubaska@comcast.net" />
```

The `css` and `xsl` files are defaults that come with `cruisecontrol`. Change them if you like.

`skipusers` is an important attribute. `Cruisecontrol` has an interesting feature in that it tries to send email to the user that made the cvs change. It constructs the address as

```
<the cvs user's name>@<the value of defaultsuffix>
```

`defaultsuffix` is an attribute of `<htmlmail />` that I haven't shown and don't use. What `skipusers` does is cause `cruisecontrol` to ignore this feature. With `skipusers` set to true, mail is sent only to `ecf-build` and not to any of the cvs users.

Note that the mail is listed as `<always />`. This means that both success and failure messages are sent. Sometimes you want only failure messages, and you can arrange that as follows.

```
<failure address="ecf-build" reportWhenFixed="true"/>
```

When you do that, an email is sent only on a failure and then again when that failure is fixed.

ECF Autobuild System

Uploading Files

We want to make our Daily Builds available for download from <http://www.eclipse.org/ecf/downloads.php> . We can make it available by uploading the files to a directory on `dev.eclipse.org`, namely `downloads/technology/ecf`. Downloads is a link under my home on `dev.eclipse.org`. Put a file there and in a few minutes, it starts appearing on the mirrors.

PassphraselessKey Authentication

When we first considered uploading files to `dev.eclipse.org`, we looked at the `<ftp />` task. However, for security reasons we preferred to use `scp` with passphraseless key authentication. We wanted passphraseless because we did not want to embed a password in our scripts.

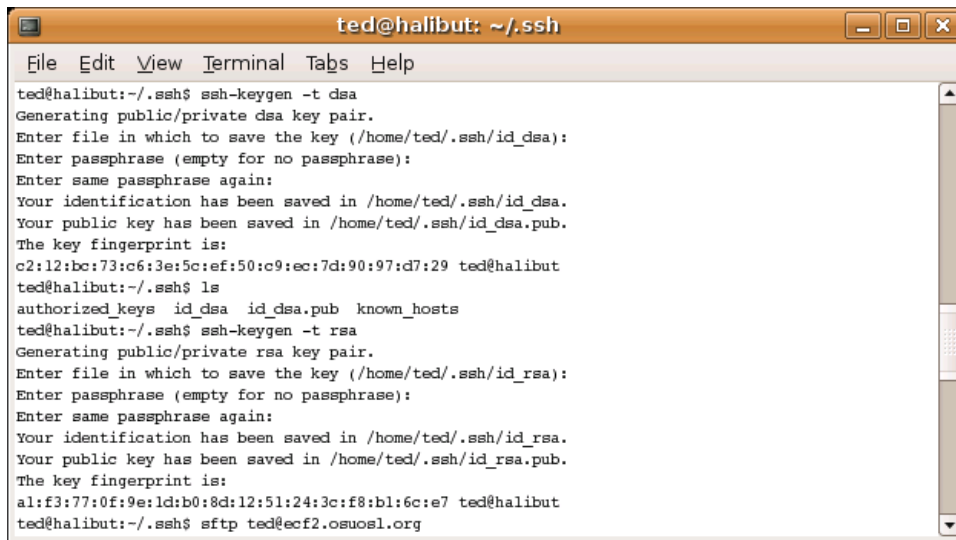
Making and Using the Key-Pair on Linux

Here's how I made both an rsa and a dsa key. In your home, create and enter the directory `.ssh`. Issue the command

```
ssh-keygen -t rsa or ssh-keygen -t dsa
```

and then enter a return for a password.

In the screenshots that follow I used my own Linux desktop called `halibut`. The keys I made are not unique to `halibut`. The keys are ASCII files, and you can look in the public key file and see `ted@halibut` in it, but that's OK. It still works with `tkubaska@dev.eclipse.org`.



```
ted@halibut: ~/ssh
File Edit View Terminal Tabs Help
ted@halibut:~/ssh$ ssh-keygen -t dsa
Generating public/private dsa key pair.
Enter file in which to save the key (/home/ted/.ssh/id_dsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/ted/.ssh/id_dsa.
Your public key has been saved in /home/ted/.ssh/id_dsa.pub.
The key fingerprint is:
c2:12:bc:73:c6:3e:5c:ef:50:c9:ec:7d:90:97:d7:29 ted@halibut
ted@halibut:~/ssh$ ls
authorized_keys id_dsa id_dsa.pub known_hosts
ted@halibut:~/ssh$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/ted/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/ted/.ssh/id_rsa.
Your public key has been saved in /home/ted/.ssh/id_rsa.pub.
The key fingerprint is:
al:f3:77:0f:9e:1d:b0:8d:12:51:24:3c:f8:b1:6c:e7 ted@halibut
ted@halibut:~/ssh$ sftp ted@ecf2.osuosl.org
```

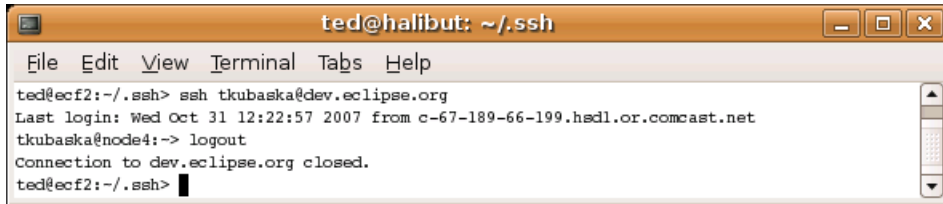
You get two keys: a private key (`id_dsa`) and a public key (`id_dsa.pub`).

Keep the private key in your `.ssh` on your source machine, in this case `ecf2`. Make the private key `-rw-----` by you; you here is the user that cruisecontrol runs as.

Put the private key in a file called `authorized_keys` in `.ssh` in your home on the destination machine. Make the file `authorized_keys -rw-----` by you.

ECF Autobuild System

With passphraseless key authentication, you can then log onto the destination machine without using a password as follows.

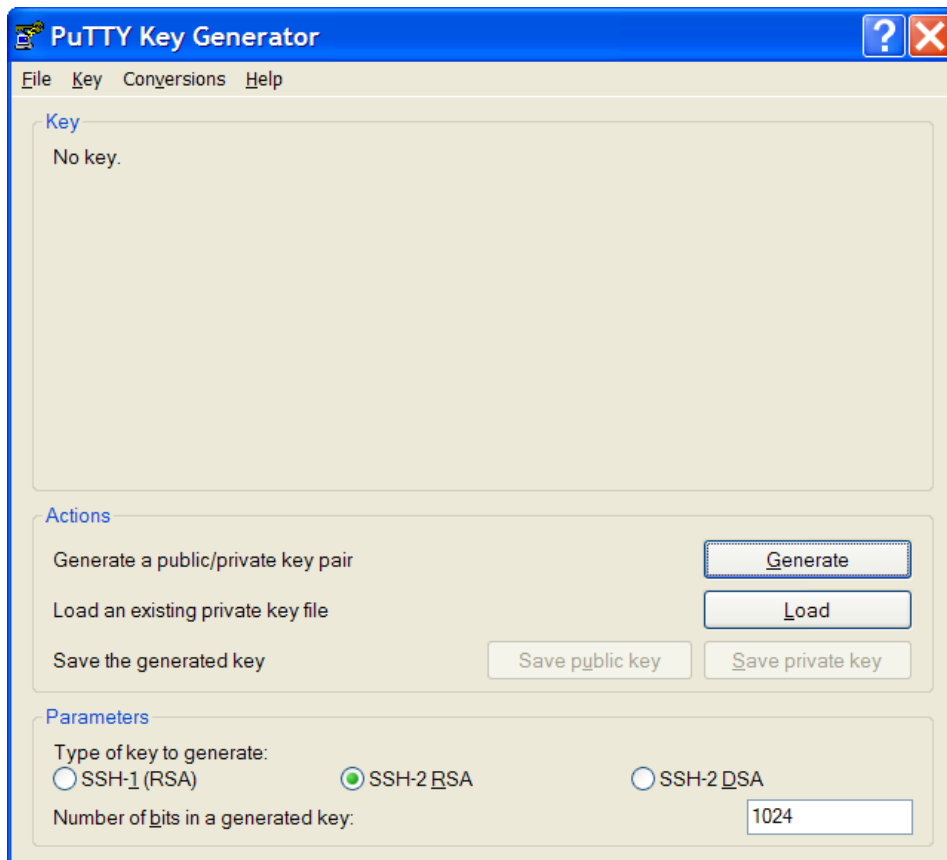


```
ted@halibut: ~/.ssh
File Edit View Terminal Tabs Help
ted@ecf2:~/.ssh> ssh tkubaska@dev.eclipse.org
Last login: Wed Oct 31 12:22:57 2007 from c-67-189-66-199.hsd1.or.comcast.net
tkubaska@node4:~> logout
Connection to dev.eclipse.org closed.
ted@ecf2:~/.ssh>
```

You can also use `scp` to transfer files from `ecf2` to `dev.eclipse.org` without having to supply a password.

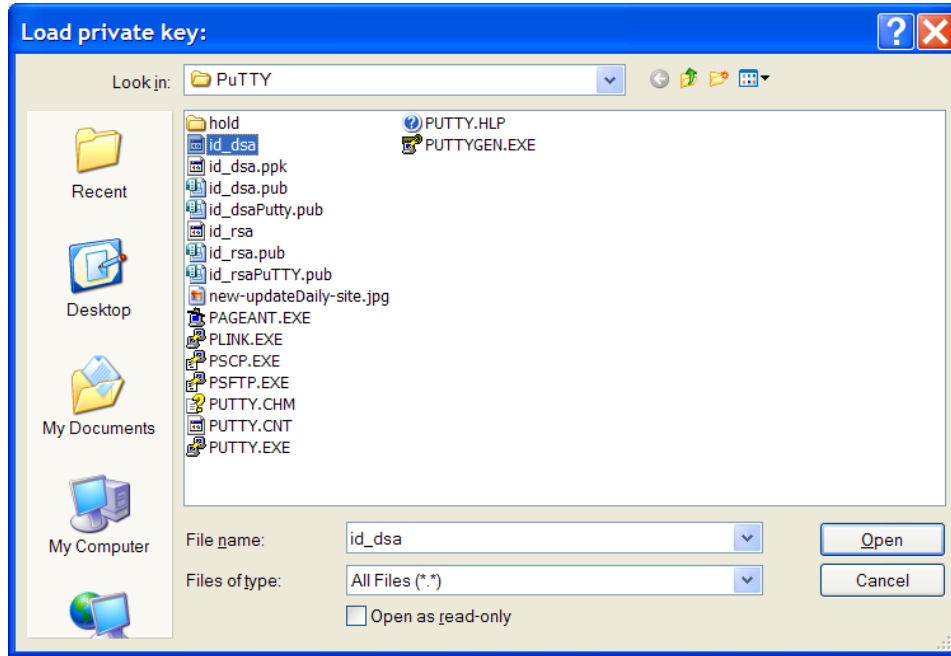
Using the Key-Pair on Windows

If you are using Windows with PuTTY, you have to load this key (which on Linux was made as an OpenSSH key) and convert it into PuTTY format. Run `puttygen.exe` and click on **Load**.



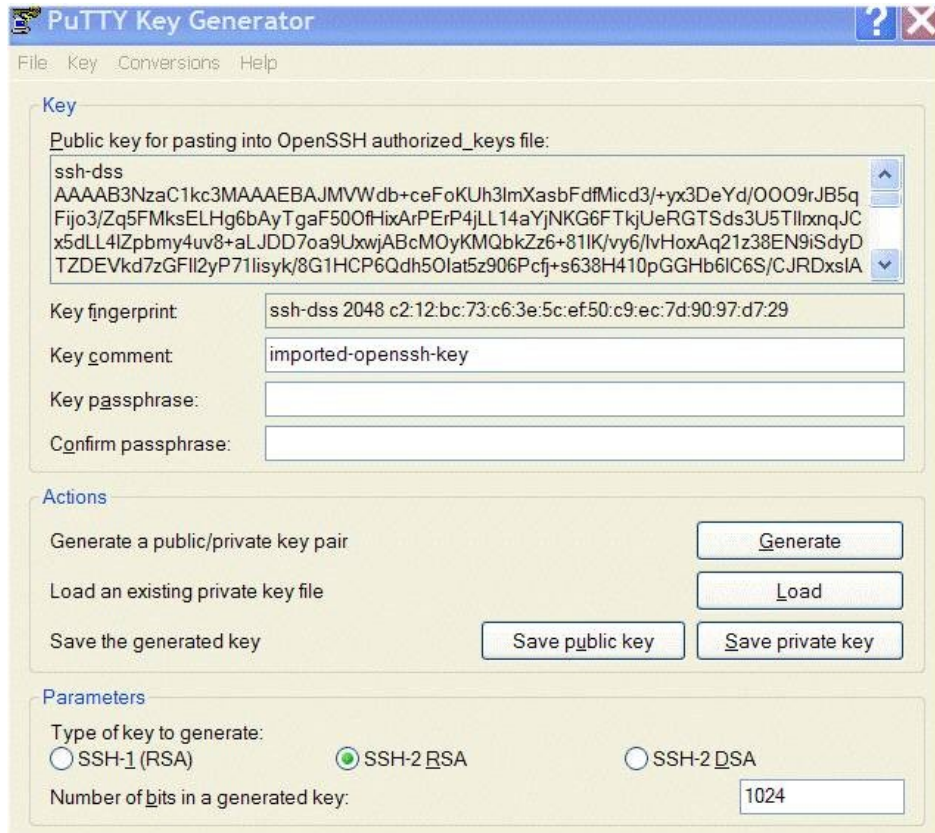
Load the private key `id_dsa`.

ECF Autobuild System



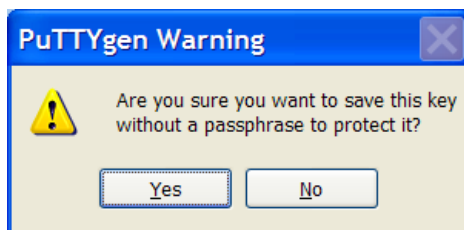
Click **OK**. Then, click **Save Private Key**.

ECF Autobuild System



The screenshot shows the PuTTY Key Generator window. The 'Key' section contains a text area with a public key, a 'Key fingerprint' field, a 'Key comment' field, and two empty fields for 'Key passphrase' and 'Confirm passphrase'. The 'Actions' section has buttons for 'Generate', 'Load', 'Save public key', and 'Save private key'. The 'Parameters' section has radio buttons for 'SSH-1 (RSA)', 'SSH-2 RSA' (selected), and 'SSH-2 DSA', and a 'Number of bits in a generated key' field set to 1024.

Click **yes**



Save the file ensuring that you save it with a **ppk** extension. Then, when you use PuTTY to connect, you must first run **pageant.exe**. When you execute pageant, it appears in your toolbar. Click on it icon and Add the private key you made. You must add this private key every time you bring up Windows, but it will stay active during your Windows session.

Uploading Daily Files

When we first started making Daily Builds, we didn't save more than the latest one. So every Daily Build that we uploaded just overwrote the previous one. What we did was add an `<onsuccess />` task underneath `<publishers />`, and in that `<onsuccess />` task call the antfile `antscp.xml`.

```
<!-- PROJECT ECF This is the Daily build -->
```

ECF Autobuild System

```
<project name="ecfDaily" buildafterfailed="false">
  <listeners>
    .
    .
  </listeners>
  <modificationset ...>
    .
    .
  </modificationset>
  <schedule ...>
    .
    .
  </schedule>
  <publishers>
    <htmlmail ...>
      .
      .
    </htmlmail>

    <onsuccess>
      <antpublisher
        antscript="/opt/apache-ant-1.7.0/bin/ant"
        antWorkingDir="/opt/build.ecf"
        buildfile="antscp.xml"
        target="deploy" />
    </onsuccess>
  </publishers>
</project>
```

The file `antscp.xml` looks as follows. We define source and destination directories. The source directory is on `ecf2`; the destination directory is on `dev.eclipse.org`. We also specify the location of the private key.

```
<project name="upload files" default="deploy">

<property name="deploysrc" location="/opt/build.ecf/ecf.output" />
<property name="deploydest"
  value="tkubaska@dev.eclipse.org:/home/data/users/tkubaska/downloads/technology/ecf/dailies" />
<property name="keyloc" location="/home/ted/.ssh/id_dsa" />

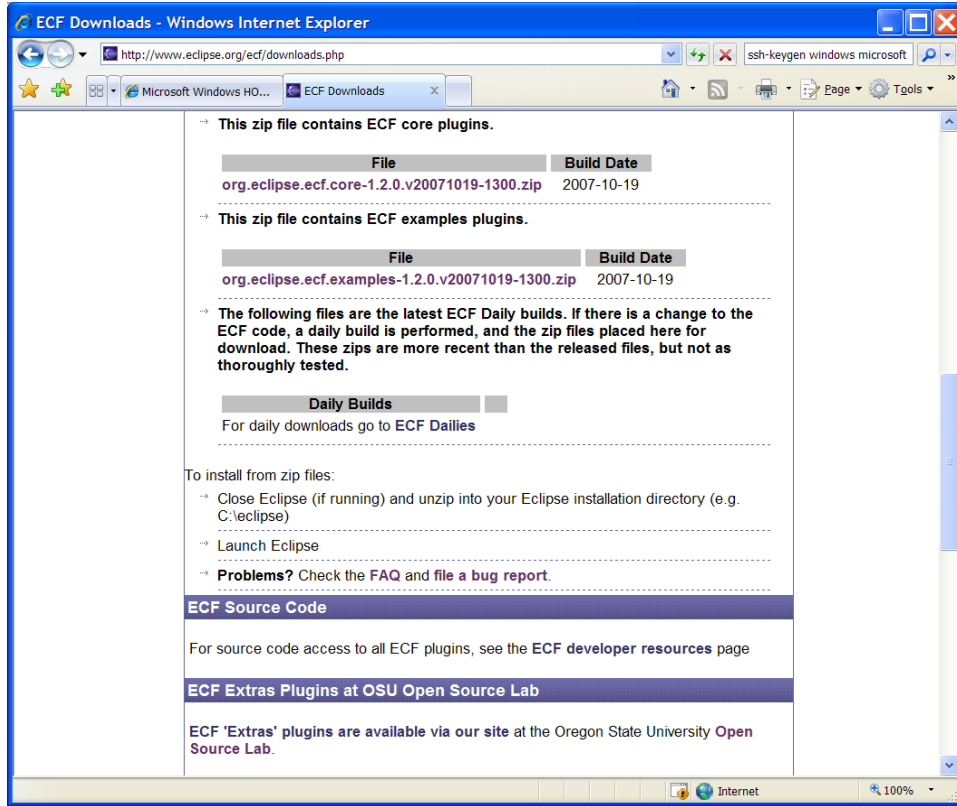
<target name="deploy">
  <scp todir="${deploydest}"
    keyfile="${keyloc}"
    passphrase="">
    <fileset dir="${deploysrc}">
      <include name="**/*.zip"/>
    </fileset>
  </scp>
</target>
```

Making Files Available on eclipse.org.

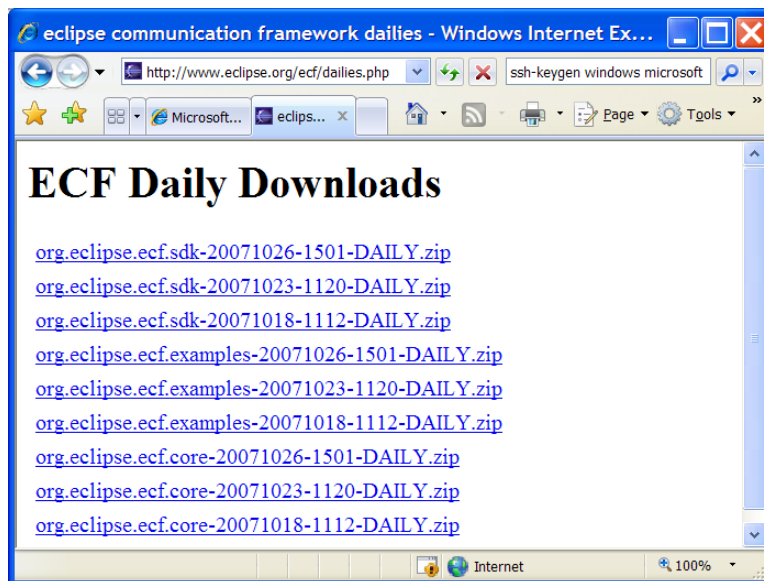
These files are available from <http://www.eclipse.org/ecf/downloads.php> . I edited

ECF Autobuild System

`download.php` to have an ECF Dailies link.



Click on ECF Dailies and see the following.



ECF Autobuild System

Note that on `dailies.php`, I show several daily builds. When we first started, we had just three Daily zips. So the three zips we had were the `sdk`, `examples`, and `core` zips. The filenames did not include the date and time (those numbers just before the DAILY). And the filenames now have the version number, which is not shown in the old figure above. A typical zip name is `org.eclipse.ecf.sdk-2.0.0.v20080620-1600-DAILY.zip`.

Keeping a Week's Worth of Dailies

Here's a problem. Our ECF web page needs to show the names of the daily builds and these names depend on what is currently being offered for download and what is being offered changes depending on whether a Daily Build is performed or not. It depends on what is in the downloads directory on `dev.eclipse.org`.

So when a user accesses the downloads page and clicks on ECF Dailies, we don't want our PHP to have to go and do an `ls` on `dev.eclipse.org`. This might result in an inordinate number of hits on `dev.eclipse.org` and, well, that's just inappropriate.

So what to do? The solution we have is not likely the best one, but in the interest of true collaboration, I'm going to describe what I did, knowing full well that it's a kludge.

More Detail on the Daily Builds

To modify how our web site presents daily builds for download, you have to first look at how the resulting zip files are moved around.

Remember in `config.xml` the target for the daily build was `ecf.copy`. If you look in `cc-build.xml` at the target `ecf.copy`, you see that it depends on `ecf.build`. So `ecf.build` gets done first and then `ecf.copy` happens.

The first thing `ecf.copy` does is copy the output to a directory called `ecf.output` on `ecf2`. It copies the updatesite as well as the zips. Then, it puts the daily builds in `ecf.dailies`. Then it does an `ls -1` on `ecf.dailies` and puts the output in `filelist.txt`. Note that this is an `ls -1` (one not el). The file `filelist.txt` is in the workspace area that eclipse uses for its checkouts. Then, `cc-build.xml` calls the antfile `antfilelist.xml`.

The antfile `antfilelist.xml` checks `filelist.txt` into `www/ecf` under the repository `dev.eclipse.org:/cvsroot/org.eclipse`, which is where our web site is stored.

Then, remember that the link `ECF Dailies` actually brings up `dailies.php`, which has in it the following lines.

```
<?php
  $files=file("filelist.txt");
  rsort($files);
  foreach ($files as $file) {
    echo '<tr> <td><p>
      <a href="http://www.eclipse.org/downloads/download.php? ' .
        'file=/technology/ecf/dailies/' . $file .'">' . $file .
        '</a></p></td></tr>';
  }
?>
```

ECF Autobuild System

Now as time marches on, the list of available daily downloads gets larger and larger. There's lots of room on `dev.eclipse.org`, but it's not good practice to let something be so open-ended. The Eclipse Foundation does not allow its users to set up `cron` jobs (rightly so, it's a security leak). But you can request that Eclipse System Administration set one up for you, and if you got a good reason, they do it. We have a `cron` that looks as follows. It gets run once a week.

```
find /home/data/users/tkubaska/downloads/technology/ecf/dailies -mtime +7
    -name '*.zip' -type f -exec rm -rf {} \;
```

Second Stage

We want to be able to force builds remotely. And we want to do that from the cruisecontrol web reporting tool. The cruisecontrol site makes a distinction between the *web reporting tool* and the *cruisecontrol GUI*. The web reporting tool is a `war` file that runs in tomcat's `webapps` folder (or under your favourite servlet engine), and the cruisecontrol GUI is a `jnlp` file.

Setting Up the Cruisecontrol Web Reporting Tool

How do you make the `war` file?

As `root`, go into the directory `/opt/cruisecontrol-2.7/reporting/jsp`. Issue the command
`sh build.sh war`

You get lots of output on the screen. You get asked for a `logs` directory; I chose `/opt/build.ecf/logs/`. Now I don't know if this is absolutely reproducible, but I think I needed that trailing slash (/). You also get asked for a `status` file; I chose `ECFcurrentbuildstatus.txt`. You get asked for an `artifacts` directory; I chose `/opt/build.ecf/logs/`. I'm not really using artifacts (an artifact is something like a `junit` report).

The result is a file called `cruisecontrol.war` in the directory `/opt/cruisescontrol-2.7/reporting/jsp/dist`. As `root`, copy this file to tomcat's `webapps` folder which on `ecf2` is `/srv/www/tomcat5/base/webapps`.

Then, as `root`, start up tomcat.

```
/usr/share/tomcat5/bin/startup.sh
```

Using the Web Reporting Tool Remotely

From Portland, we want to be able to log onto `ecf2` in Corvallis and force either an auto or a daily build. We do this with *ssh passphraseless key authentication* and *ssh port forwarding*.

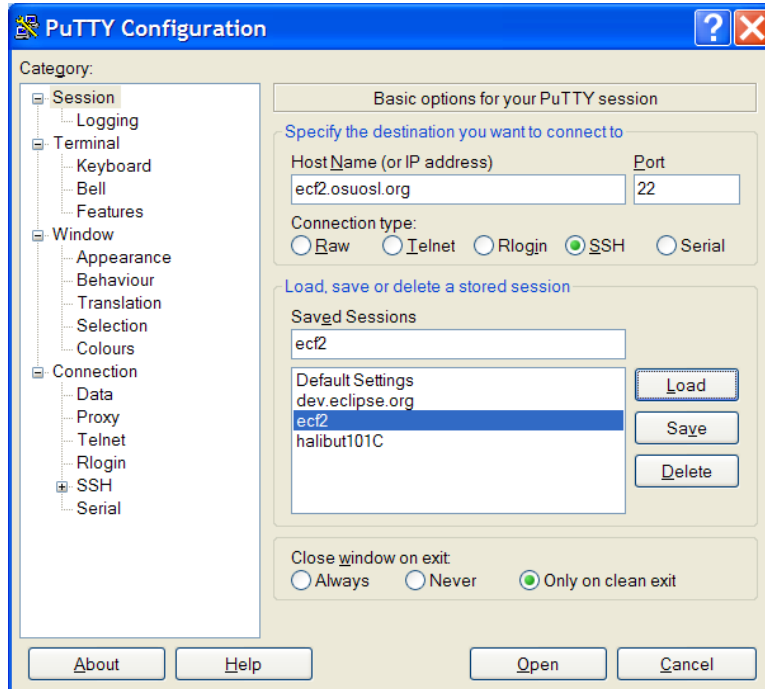
We do our remote administration from the command line, from a port-forwarded browser, and from a port-forwarded desktop. From the command line, we can edit the configuration files, start/stop cruisecontrol, and look at log files. We use the port-forwarded browser to force builds. We use the port-forwarded browser to configure Eclipse and sometimes force builds.

Getting a Remote Desktop

I'll show this from Windows, but it works also from Linux.

ECF Autobuild System

From Windows, I use PuTTY. Set `ecf2.osuosl.org` as the host name and port 22. Click on Load, then Open.



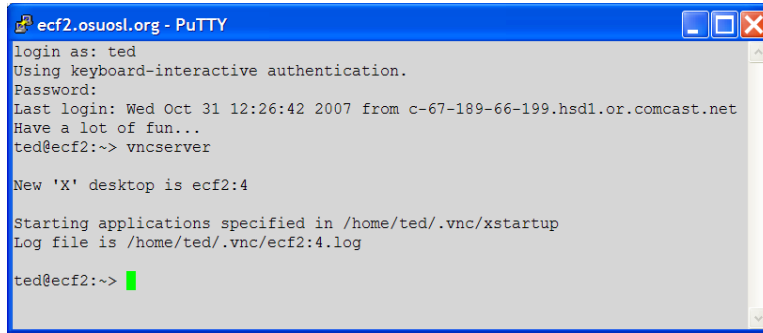
Get a login screen. Login and start the `vncserver`. If this is the first time you are running `vncserver`, you get to set a password. Remember it.

Now as a sidenote, realize that the window you get by default is going to be `twm`. If you like `twm`, that's fine. I prefer `gnome`. You can choose your window manager by putting the appropriate `xstartup` file in `.vnc` in your home. Note the `passwd` file. It stores your `vnc` password as a data file.

```
ecf2.osuosl.org - PuTTY
ted@ecf2:~/.vnc> ls -l
total 212
-rw-rw-r-- 1 ted cimasters 118931 2007-10-31 13:00 ecf2:4.log
-rw-r--r-- 1 ted users 6 2007-10-31 12:32 ecf2:4.pid
-rw-rw-r-- 1 ted cimasters 27017 2007-10-18 21:49 ecf2:5.log
-rw-r--r-- 1 ted users 6 2007-10-18 21:49 ecf2:5.pid
-rw-rw-r-- 1 ted cimasters 32655 2007-10-18 21:48 ecf2:6.log
-rw-rw-r-- 1 ted cimasters 6 2007-10-18 21:47 ecf2:6.pid
-rw-r--r-- 1 ted users 2909 2007-08-01 14:51 ecf2:9.log
-rw-r--r-- 1 ted users 4 2007-08-01 14:44 ecf2:9.pid
-rw----- 1 ted users 8 2007-08-01 14:08 passwd
-rwxr-xr-x 1 ted users 1365 2007-08-01 14:08 xstartup
ted@ecf2:~/.vnc>
ted@ecf2:~/.vnc>
```

Assuming you got the files in `.vnc` that you want, here's how the start for `vncserver` looks.

ECF Autobuild System



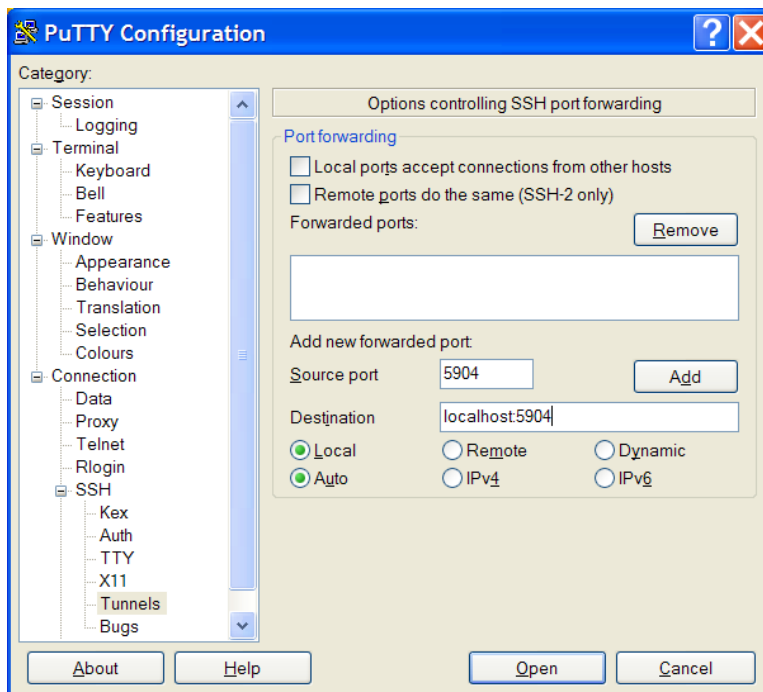
```
ecf2.osuosl.org - PuTTY
login as: ted
Using keyboard-interactive authentication.
Password:
Last login: Wed Oct 31 12:26:42 2007 from c-67-189-66-199.hsd1.or.comcast.net
Have a lot of fun...
ted@ecf2:~$ vncserver

New 'X' desktop is ecf2:4

Starting applications specified in /home/ted/.vnc/xstartup
Log file is /home/ted/.vnc/ecf2:4.log

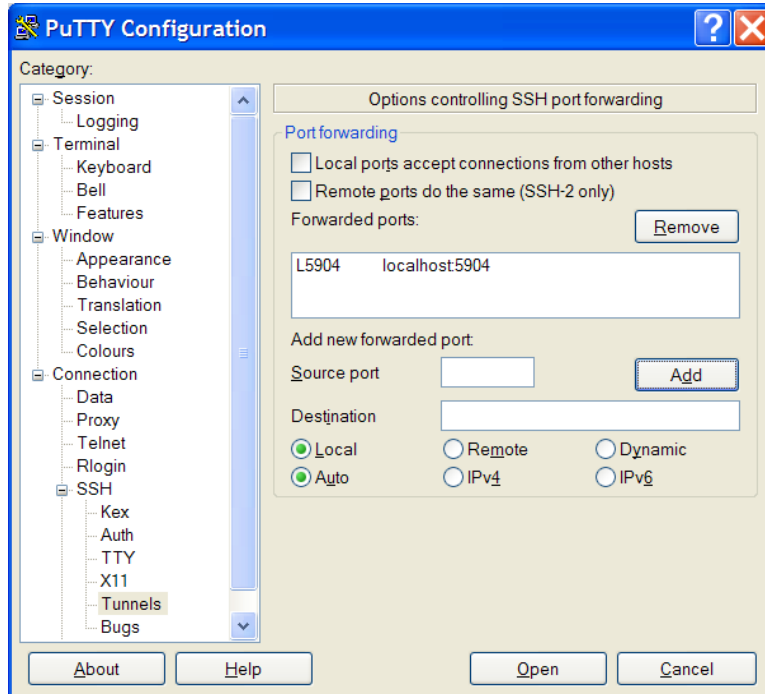
ted@ecf2:~$
```

Then, open up another instance of PuTTY. Load ecf2. Then, choose SSH/Tunnels. Type in 5904 for the Source port. The 4 in 5904 comes from the 4 in the login window shown above. Also type in localhost:5904 for Destination.

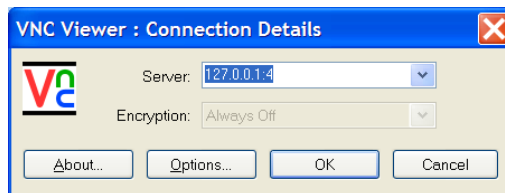


Click on Add.

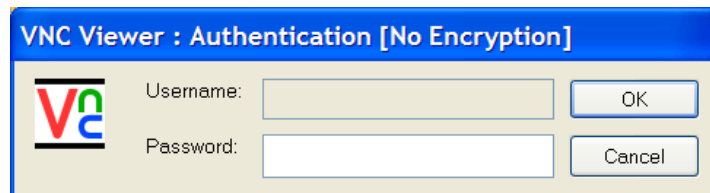
ECF Autobuild System



Then, click on Open and login. On your Windows box, open up RealVNC (you've already installed this, right?)

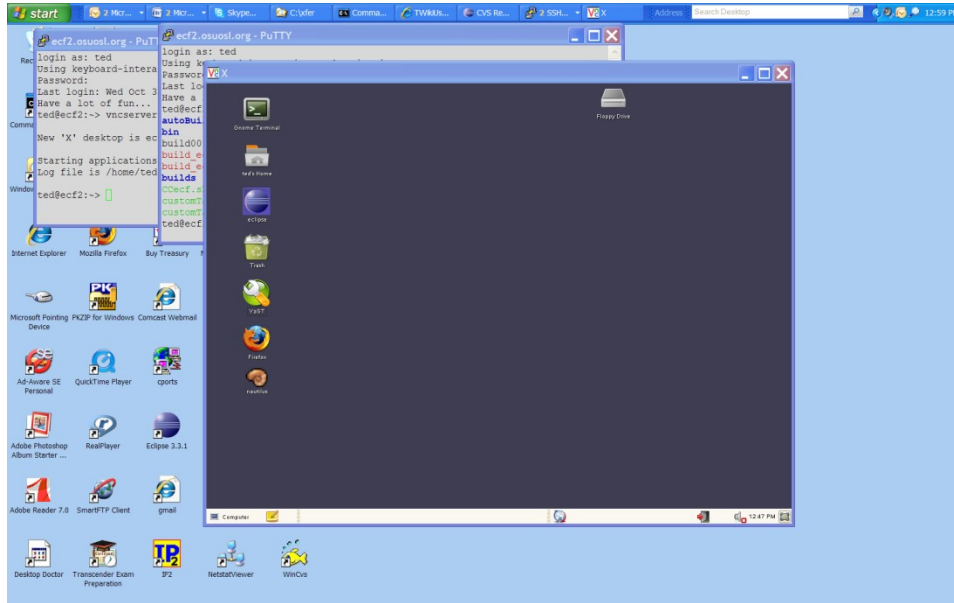


Click on OK.

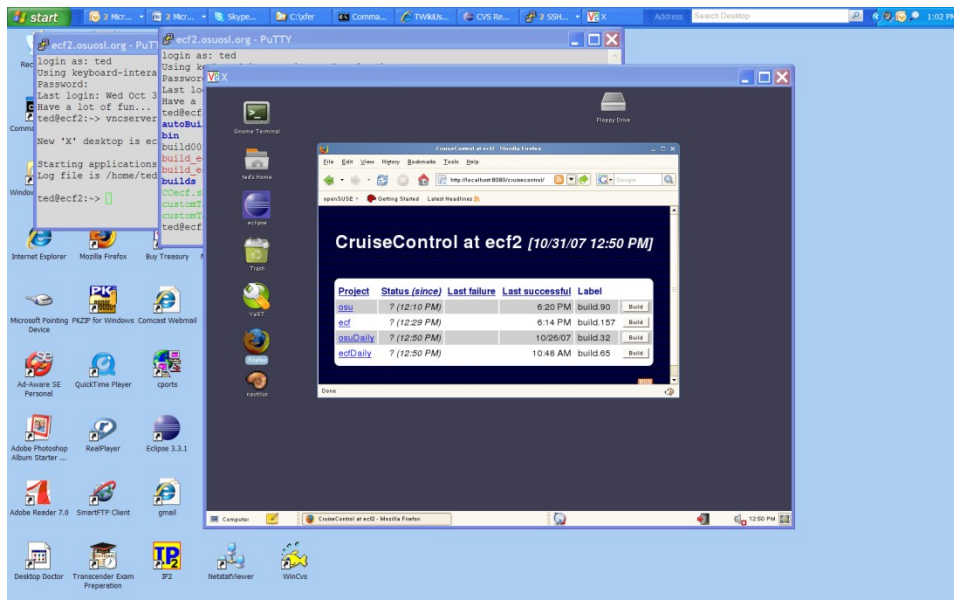


Type in your password. This is not your user password on **ecf2** but rather the password you chose when you set up **vncserver** for the first time. Then you see the VNC window into **ecf2**.

ECF Autobuild System



Open up Firefox in the **ecf2** window. Navigate to <http://localhost:8080/cruisecontrol>. The cruisecontrol GUI comes up. Notice that there are actually four projects listed. (Currently, we have nine projects.) **osu** and **ecf** are the Auto Builds for the **osu** and **ecf** projects (we haven't talked about **osu** builds yet). **osuDaily** and **ecfDaily** are the Daily Builds.



OK, let's start a build. First, if you want to monitor the build, open up an **xterm**. You're not going to get the **gnome** terminal launcher by default. I set one up previously. In the **xterm**, **cd** to

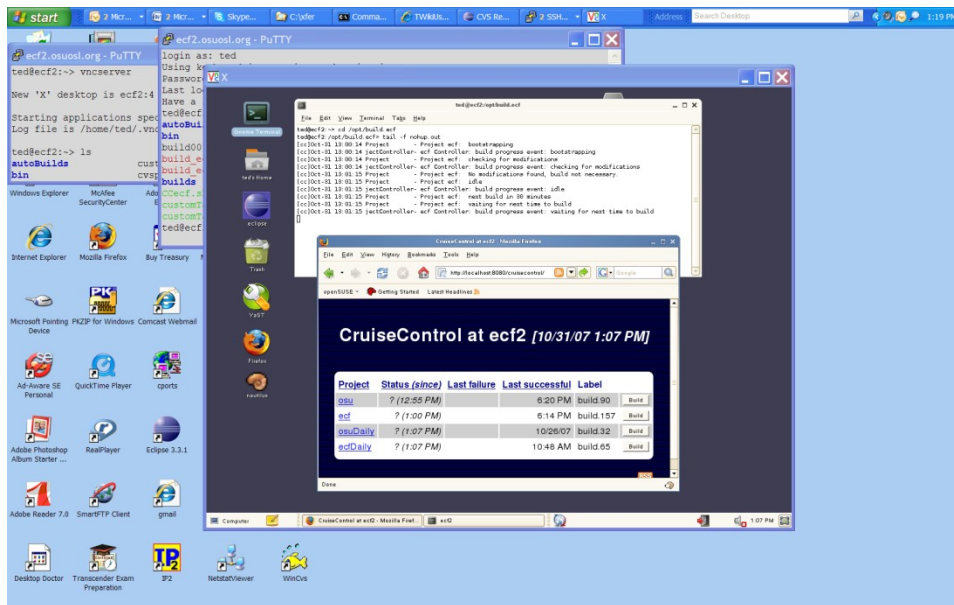
ECF Autobuild System

`/opt/build.ecf` and issue a `tail -f nohup.out`.

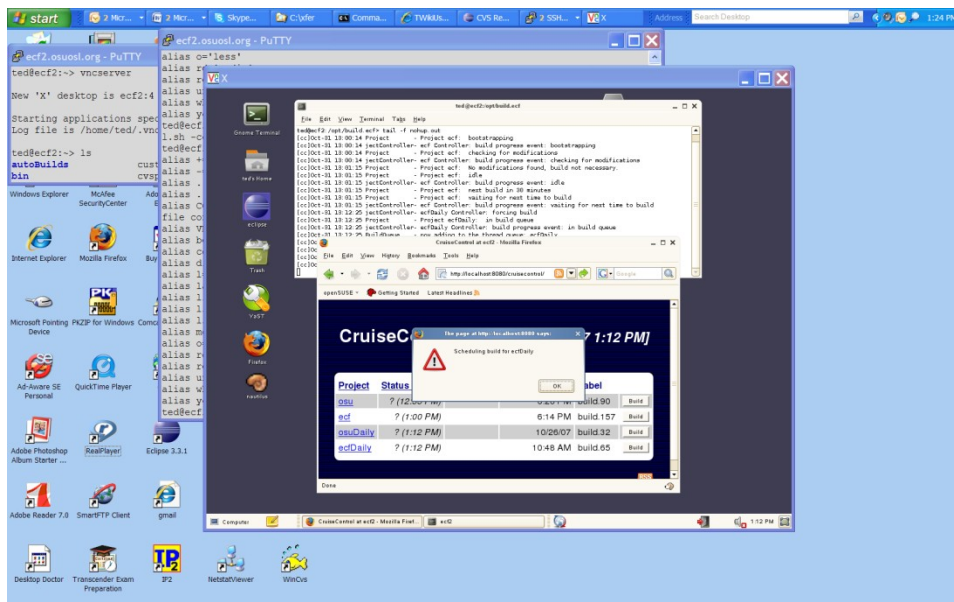
Cruisecontrol runs all the time. It does sometimes die or hang, but it's pretty reliable. But I started it with `nohup` so that I could log out of `ecf2`. I started it as follows.

```
nohup /opt/cruisecontrol-2.7/main/bin/cruisecontrol.sh -configfile config.xml  
-port 8000 -rmiport 1099'
```

You can see its output by looking at `nohup.out`.

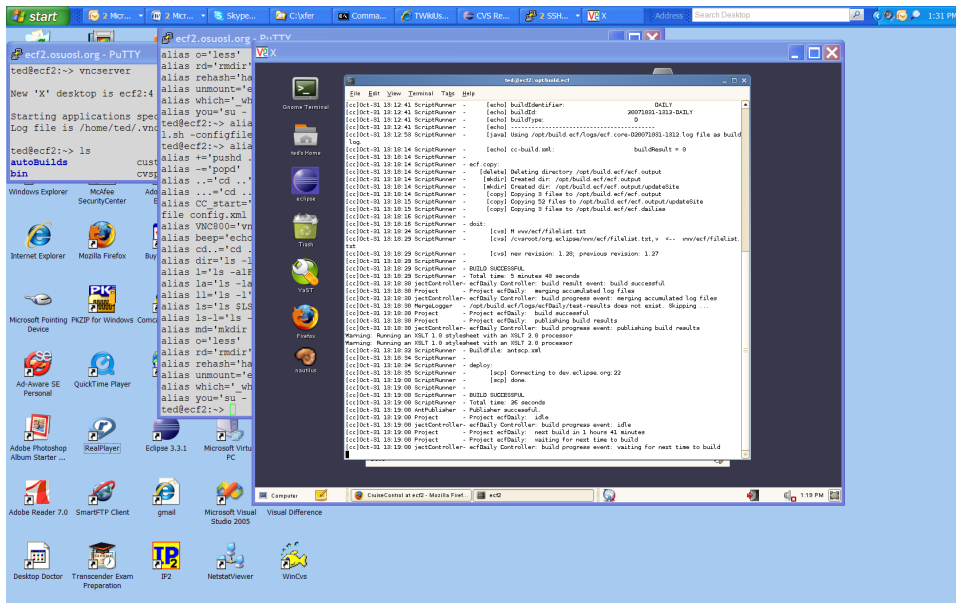


Click on the **Build** button for `ecfDaily`. Then, click on OK.



ECF Autobuild System

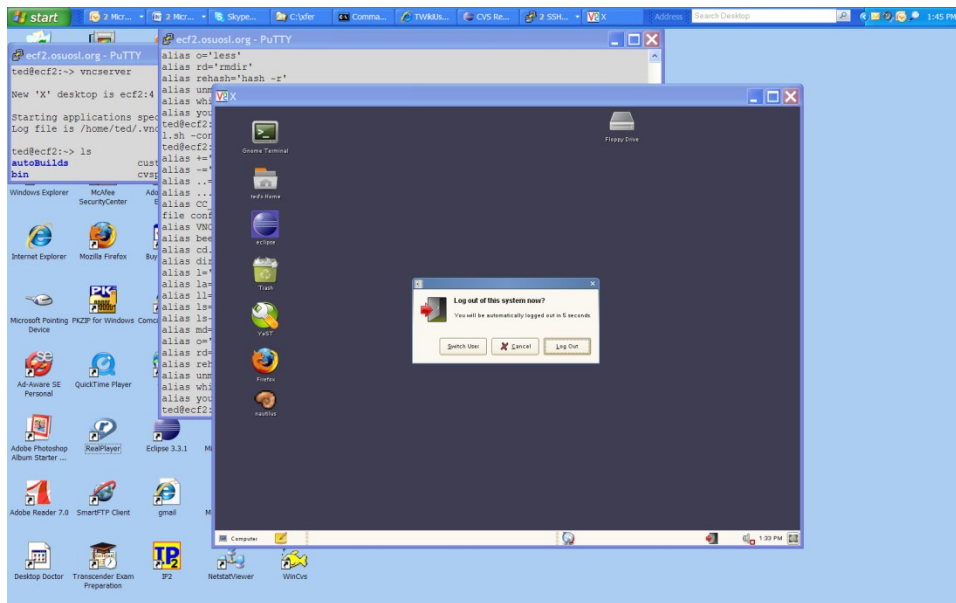
Here's the build output.



Notice from the build output that the build is successful (buildResult=0 ... first red arrow), that the checkin of filelist.txt occurs (the second red arrow), and that the scp transfer occurs (the third red arrow).

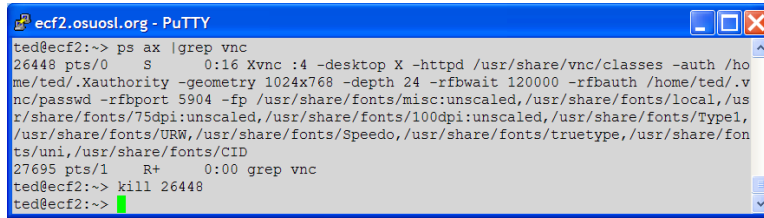
ECF Autobuild System

```
ted@ecf2:/opt/build.ecf
File Edit View Terminal Tabs Help
[cc]Oct-31 13:12:41 ScriptRunner - [echo] buildIdentifier: DAILY
[cc]Oct-31 13:12:41 ScriptRunner - [echo] buildId: 20071031-1312-DAILY
[cc]Oct-31 13:12:41 ScriptRunner - [echo] buildType: D
[cc]Oct-31 13:12:41 ScriptRunner - [echo] -----
[cc]Oct-31 13:12:53 ScriptRunner - [java] Using /opt/build.ecf/logs/ecf.core-D20071031-1312.log file as build
log.
[cc]Oct-31 13:18:14 ScriptRunner - [echo] cc-build.xml: buildResult = 0
[cc]Oct-31 13:18:14 ScriptRunner - [delete] Deleting directory /opt/build.ecf/ecf.output
[cc]Oct-31 13:18:14 ScriptRunner - [mkdir] Created dir: /opt/build.ecf/ecf.output
[cc]Oct-31 13:18:14 ScriptRunner - [mkdir] Created dir: /opt/build.ecf/ecf.output/updateSite
[cc]Oct-31 13:18:14 ScriptRunner - [copy] Copying 3 files to /opt/build.ecf/ecf.output
[cc]Oct-31 13:18:15 ScriptRunner - [copy] Copying 52 files to /opt/build.ecf/ecf.output/updateSite
[cc]Oct-31 13:18:15 ScriptRunner - [copy] Copying 3 files to /opt/build.ecf/ecf.dailies
[cc]Oct-31 13:18:16 ScriptRunner - [doit]:
[cc]Oct-31 13:18:24 ScriptRunner - [cvs] M www/ecf/filelist.txt
[cc]Oct-31 13:18:29 ScriptRunner - [cvs] /cvsroot/org.eclipse/www/ecf/filelist.txt,v <- www/ecf/filelist.
txt
[cc]Oct-31 13:18:29 ScriptRunner - [cvs] new revision: 1.28, previous revision: 1.27
[cc]Oct-31 13:18:29 ScriptRunner - BUILD SUCCESSFUL
[cc]Oct-31 13:18:29 ScriptRunner - Total time: 5 minutes 48 seconds
[cc]Oct-31 13:18:30 jectController- ecfDaily Controller: build result event: build successful
[cc]Oct-31 13:18:30 Project - Project ecfDaily: merging accumulated log files
[cc]Oct-31 13:18:30 jectController- ecfDaily Controller: build progress event: merging accumulated log files
[cc]Oct-31 13:18:30 MergeLogger - /opt/build.ecf/logs/ecfDaily/test-results does not exist. Skipping ...
[cc]Oct-31 13:18:30 Project - Project ecfDaily: build successful
[cc]Oct-31 13:18:30 Project - Project ecfDaily: publishing build results
[cc]Oct-31 13:18:30 jectController- ecfDaily Controller: build progress event: publishing build results
Warning: Running an XSLT 1.0 stylesheet with an XSLT 2.0 processor
Warning: Running an XSLT 1.0 stylesheet with an XSLT 2.0 processor
[cc]Oct-31 13:18:32 ScriptRunner - Buildfile: antscp.xml
[cc]Oct-31 13:18:34 ScriptRunner -
[cc]Oct-31 13:18:34 ScriptRunner - deploy:
[cc]Oct-31 13:18:35 ScriptRunner - [scp] Connecting to dev.eclipse.org:22
[cc]Oct-31 13:19:00 ScriptRunner - [scp] done.
[cc]Oct-31 13:19:00 ScriptRunner - BUILD SUCCESSFUL
[cc]Oct-31 13:19:00 ScriptRunner - Total time: 26 seconds
[cc]Oct-31 13:19:00 AntPublisher - Publisher successful.
[cc]Oct-31 13:19:00 Project - Project ecfDaily: idle
[cc]Oct-31 13:19:00 jectController- ecfDaily Controller: build progress event: idle
[cc]Oct-31 13:19:00 Project - Project ecfDaily: next build in 1 hours 41 minutes
[cc]Oct-31 13:19:00 Project - Project ecfDaily: waiting for next time to build
[cc]Oct-31 13:19:00 jectController- ecfDaily Controller: build progress event: waiting for next time to build
```



Click on Log Out. Then, in one of the PuTTY windows, find the pid of the `vnc` process and kill it.

ECF Autobuild System



```
ecf2.osuosl.org - PuTTY
ted@ecf2:~$ ps ax |grep vnc
26448 pts/0    S      0:16 Xvnc :4 -desktop X -httpd /usr/share/vnc/classes -auth /home/ted/.Xauthority -geometry 1024x768 -depth 24 -rfbwait 120000 -rfbauth /home/ted/.vnc/passwd -rfbport 5904 -fp /usr/share/fonts/misc:unscald,/usr/share/fonts/local,/usr/share/fonts/75dpi:unscald,/usr/share/fonts/100dpi:unscald,/usr/share/fonts/Type1,/usr/share/fonts/URW,/usr/share/fonts/Speedo,/usr/share/fonts/truetype,/usr/share/fonts/uni,/usr/share/fonts/CID
27695 pts/1  R+    0:00 grep vnc
ted@ecf2:~$ kill 26448
ted@ecf2:~$
```

Logout of any remaining PuTTY windows.

Getting a Remote Browser

We want to force builds without having to get a remote desktop. This involves forwarding ports 8000 and 8080. But just forwarding the ports is not enough.

At first that's what I did, and the JMX console on port 8000 is functional; the cruisecontrol GUI (that tomcat war file) was not. The cruisecontrol GUI came up, but the **Build** buttons didn't do anything.

You can force builds through the JMX console, but it's clunky. The JMX console is very functional, and what we want to do is very simple. It's easier to use the cruisecontrol GUI.

Here's what I had to do.

Ec2 runs SuSE Linux, which wants to do a bunch of stuff by default that gets in the way. Before making the `cruisecontrol.war` file, I took out the default SuSE stuff by adding the following lines to `.bashrc` for both `root` and `ted`.

```
unset JDK_HOME
unset JAVA_BINDIR
unset JAVA_HOME
unset JRE_HOME
unset SDK_HOME
unset JAVA_ROOT
```

Then defined the `JAVA_HOME` I wanted.

```
JAVA_HOME=/opt/jdk1.6.0_04
export JAVA_HOME
```

In `root`'s `.bashrc` I defined

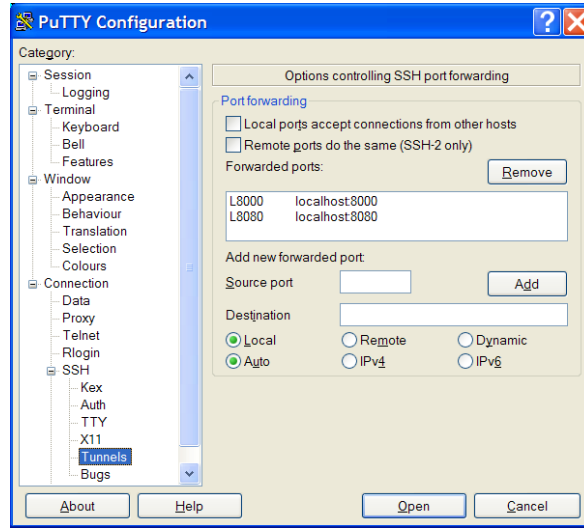
```
CATALINA_HOME=/opt/apache-tomcat-6.0.13
export CATALINA_HOME
```

Then, I edited the `web.xml` in `/opt/apache-tomcat-6.0.13/webapps/cruisecontrol/WEB-INF`. For some reason I do not understand, the IP address did not work. But I used `localhost` and that worked.

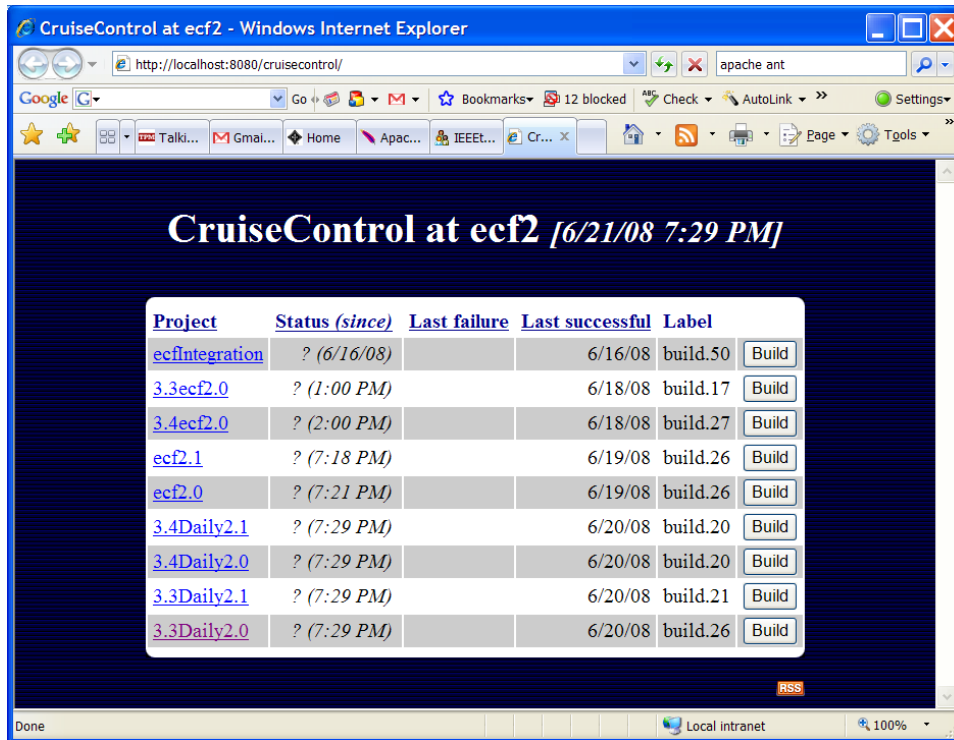
```
<context-param>
  <param-name>cruisecontrol.jmxhost</param-name>
  <param-value>localhost</param-value>
  <description>If your server doesn't know it's own proper IP address or it's name isn't
  resolvable by other machines on the network, set the IP address or resolvable name here. This
  will enable things like the "Force Builds" button to work.
</description>
</context-param>
```

To use the CC GUI on `ecf2`, make a PuTTY connection as follows.

ECF Autobuild System



Then, point your browser to <http://localhost:8080/cruisecontrol>. You see the following.



Third Stage

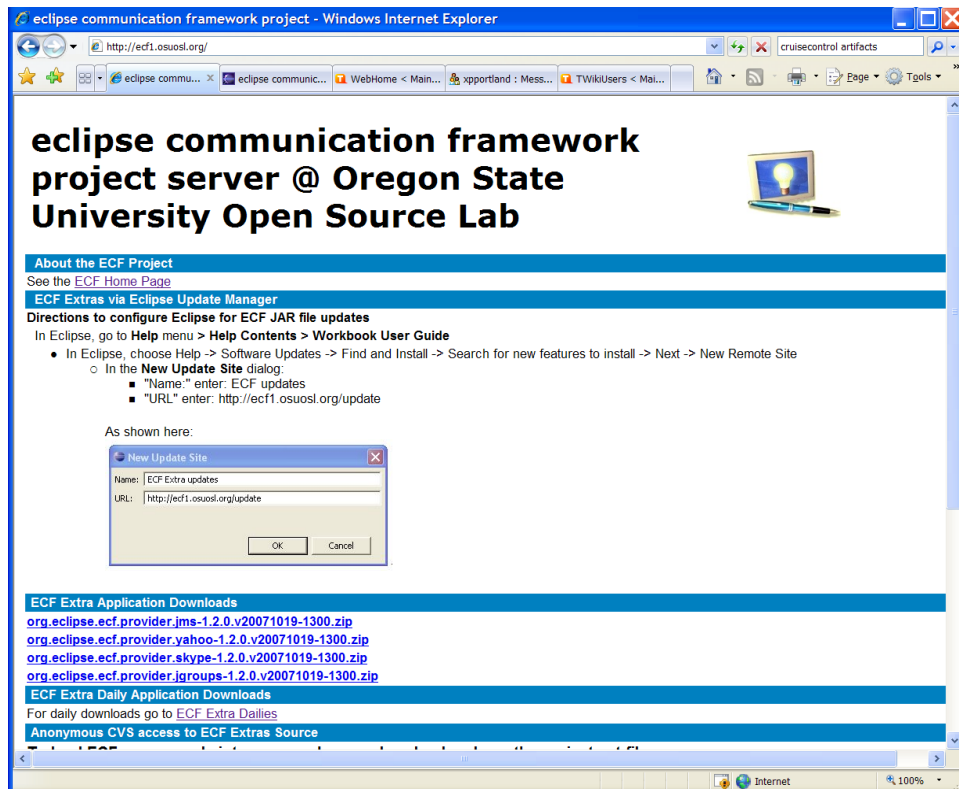
We want to incorporate the `osu` build into our automated build. The `osu` build consists of plug-ins that are in an early stage of development, so early that they have not yet passed the stringent IP requirements that Eclipse puts on its plug-ins. We have every expectation that they will, but

ECF Autobuild System

they haven't yet.

So the plug-ins belonging to the **osu** build are not kept in our eclipse repository. Rather they are kept in a repository on another machine at Oregon State called **ecf1**, and they are accessed through a web site hosted on **ecf1**.

I modified the **ecf1** site to provide a link to ECF Extra Downloads.



Click on **ECF Extra Dailies**, and see the available **osu** downloads. Currently the **osu** plug-ins consist of the **yahoo**, **skype**, **jms**, and **jgroups** plug-ins.

ECF Autobuild System



Originally, we ran all the projects on **ecf2**. We found, when we did that, that the Skype plugins were non-functional. The problem was that **ecf2** is a 64-bit machine and that the Skype builds use some 32-bit plugins and fragments from the Eclipse plug-in directory. **ecf2** requires a 64-bit version of Eclipse and a 64-bit version of Java.; the 32-bit versions do not work on **ecf2**. There should, I think, be some way of forcing the builds on **ecf2** to use the 32-bit plugins and fragments, but I was not successful in doing this. Finally, I just set up cruisecontrol/ant/tomcat on **ecf1** and run the osu builds on **ecf1**, and that works.

To add the **osu** build to cruisecontrol, edit **config.xml**. The new projects (called **osu** and **osuDaily**) are very similar to the **ecf** projects.

ECF Autobuild System

```
<project name="osu" buildafterfailed="false">

  <listeners>
    .
  </listeners>

  <modificationset quietperiod="300">
    <cvsw localworkingcopy="${localcopyEXAMPLE}" />
    <cvsw localworkingcopy="${OSUlocalcopy}" />
  </modificationset>

  <schedule interval="2700">
    <ant buildfile="OSUcc-build.xml" target="ecf.osuos1" >
      <property name="mapVTag" value="HEAD" />
      <property name="feature" value="ecf.osuos1" />
      <property name="buildIdentifier" value="false" />
      <property name="buildType" value="A" />
      <property name="genFVSuffix" value="true" />
    </ant>
  </schedule>

  <publishers>
    <htmlmail ...>
      .
    </htmlmail>
  </publishers>
</project>
```

Notice that the `<modificationset />` is different because the cruisecontrol monitors a different set of files. Also, the antfile that is called in the `<schedule />` task is different, and this antfile calls a different task called `ecf.osuos1`.

Inside `OSUcc-build.xml`, specific build information comes from the directory `ecf.osuos1` instead of `ecf.core`. This specific information consists of stuff like what map files to use, the location of the CVS repository, and the location of the Eclipse workspace.

Also, when the `osuDaily` build is performed the resulting zip files are not transferred to `dev.eclipse.org`. Rather they are transferred to `ecf1`.

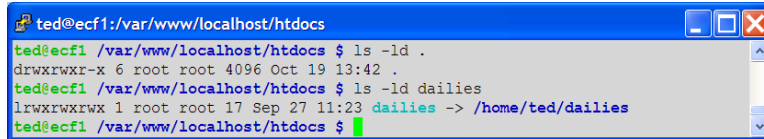
The `osu` repository is on `ecf1`; the build is performed on `ecf1` (used to be `ecf2`), and the resulting zip files stay on `ecf1`. Actually, they are transferred to `ecf1` from `ecf1`, so that the code stays essentially the same.

How does that file transfer occur? Remember the file `antscp.xml` used in the `ecf` file transfer? There's a corresponding `OSUantscp.xml`. `ecf1` has a copy of the `ecf2` public key, just like `dev.eclipse.org`. The files are transferred to the directory `/home/ted/dailies` on `ecf1`.

Now to have the files available on the `ecf1` web site, they must reside in `/var/www/localhost/htdocs`. We actually want them in a directory `dailies` in `htdocs`. The key authentication only works for an ordinary user (giving key authentication to `root` is a silly idea even if it is possible).

ECF Autobuild System

Nor do we want to open up `htdocs` to a group to which a user has write access. So what I did was make a link in `htdocs` to a directory in my home that contains the `osu` zips.



```
ted@ecf1:/var/www/localhost/htdocs
ted@ecf1 /var/www/localhost/htdocs $ ls -ld .
drwxrwxr-x 6 root root 4096 Oct 19 13:42 .
ted@ecf1 /var/www/localhost/htdocs $ ls -ld dailies
lrwxrwxrwx 1 root root 17 Sep 27 11:23 dailies -> /home/ted/dailies
ted@ecf1 /var/www/localhost/htdocs $
```

Then if you have the filenames in an array (called `$file`), you present them for download as follows.

```
foreach ($files as $file) {
    echo '<tr> <td> <p><a href="http://ecf1.osuosl.org/dailies/' . $file . '">'
    . $file . '</a></p></td></tr>';
}
}
```

How to get them into a `$file` array. The following PHP function does that.

```
function directoryToArray($directory, $recursive) {
    $array_items = array();
    if ($handle = opendir($directory)) {
        while (false !== ($file = readdir($handle))) {
            if ($file != "." && $file != "..") {
                $array_items[] = preg_replace("/\\/\\/\\/si", "/", $file);
            }
        }
    }
    closedir($handle);
    rsort($array_items, SORT_STRING);
    return $array_items;
}
```

Then,

```
$files=directoryToArray("/var/www/localhost/htdocs/dailies",false);
```

Oh, and the same thing with `ecf1` ... OSU won't let us write our own cron job; we must make a request.

Fourth Stage

I'm going to describe the fourth stage in another file because this one is getting too long. Here's what happened in our fourth stage.

- We added a number of new projects on `ecf2` for a total of nine. The extra projects have to do with using CVS branches.
- We have a 2.0 and a 2.1 branch. We want to continue work on 2.1 and provide 2.0 for the Eclipse Ganymede release.

ECF Autobuild System

- We have a branch used for integration with the Eclipse platform. The Eclipse platform itself uses six ECF plugins/fragments for 3.4. So we need a project that provides these plugins to the platform on a weekly basis.
- We want projects for the Eclipse 3.3 and 3.4 releases. These release builds must have their jars signed with Eclipse's Verisign certificate. The difference between ECF provided for 3.3 and ECF provided for 3.4 is that the 3.3 ECF includes the plugins/fragments that in 3.4 ECF are provided by the platform. So the 3.4 ECF must not include these plugins.