Crash Intro to SVN

"I will forge the commit logs, so help me God."

This is a short document on how to use the SVN version control system.[1] It will teach you the bare minimum you need to know to use it with game development.

What is SVN? Well, the internet says it's a "version control system". Basically, it tries to solve all of the things that can go wrong when multiple people are editing, adding and deleting files from a big directory all at once (two people trying to edit the same file, nobody knowing who has the newest version of a given file, etc.). This is what you will be doing while writing a game. Subversion works through the following mechanisms:

- It maintains *one* central copy of the project you're working on ("the repository").
- It lets each user make changes by modifying their own copy of the project and "committing" those changes back to the repository.
- When a user asks for the most up-to-date version of the project, SVN will give it to them, being careful not to override changes that they've made on their copy.

We will talk about how to set it up and how to use it. First, to make sure svn works for you on Athena, add the following line to your `.environment` file.

```
add svn
```

Creating a repository *You will only have to do this step once per game.*

Before you do this step, you should have a GameTeX tree already set up in one of your lockers (see the "Crash Intro to GameTeX" greensheet; life will be easier if you've already run Extras/changeclass.pl). The central SVN repository (often called "the repo") needs someplace to live. One of you should agree to host it in your locker, say in a directory called `svngamename`. From the command line, you should do several things.

You should also have an MIT mailing list for your team. It should be a moira list (*not* mailman) and an AFS group.

First, create the directory and give your GM team access to it (let's say that your GMs are all on a list called `gamename-gms`. Then, remove everybody else's "list" permissions from it. We'll call the user creating the repo `joeuser`.

```
% mkdir svngamename
% fs sa svngamename system:gamename-gms write
% fs sa svngamename system:anyuser none
```

You can double check the permissions. They should look something like this:

```
% fs la svngamename
Access list for svngamename is
Normal rights:
system:gamename-gms rlidwk
system:expunge ld
joeuser rlidwka
```

Then, let's create a fresh, blank repository inside:

```
% svnadmin create svngamename/
```

[1] last modified 21 July 2012

Good. If you look inside the repo, it will have all sorts of mysterious directories.

```
% ls svngamename/
conf dav db format hooks locks README.txt
```

Do not manually change anything inside the repository. It will look like garbage anyway. Only use the SVN commands we'll talk about in a second.

This blank repository needs a copy of your GameTeX tree in it so that you can start using the repo. To do this, you need to *import* the tree so that the repo can start keeping track of it. Go to the root of a GameTeX tree you've previously set up, and run the following command. A text editor will pop up. Just enter "Initial commit" and exit the text editor.

```
GameName% svn import .  file:///mit/joeuser/svngamename
Adding README
Adding Bluesheets
Adding Bluesheets/README.tex
...  lots of other output ...
Committed revision 1.
```

Now your repo contains a copy of your GameTeX tree and is ready to be used.

Checking out a copy *You will only have to do this step once per GM.*

Good. Now that the repository is up and running in `joeuser`'s locker, each GM needs to check out a local copy of it. Since the repo is running out of `joeuser`'s locker, let's make sure it is always attached by adding the following line to your `.environment`:

```
add joeuser
```

Now, create the folder to wherever you decided your local copy will sit (you probably want it somewhere like `/mit/yourusername/GameName/`, but consult the "Intro to GameTeX" greensheet). The SVN command to checkout a copy from the repo has this form:

```
% svn checkout file:///mit/joeuser/svngamename GameName/
A GameName/README
A GameName/Bluesheets
A GameName/Bluesheets/README.tex
...  lots of other output ...
Checked out revision 2.
```

Above, make sure to replace `/mit/joeuser/svngamename` with the actual location of the repository up above and `GameName` with the directory you want to keep your game tree in. After all is said and done, you will now have a working local copy of the repository. Each GM should do this, including the one who created the repository. Your checked out copy will have hidden `.svn/` folders all over the place that you may notice. Don't mess with those - that's just how SVN keeps track of some information.

Updating and committing This section will tell you about the most common operations. Once you have checked out a copy of the repository, you can start to use SVN in full. To update your local copy with the newest version of the repository, use the `svn update` command.

```
% svn update
U Greensheets/magic.tex
G Bluesheets/anarchists.tex
```

A "U" means that a file was updated with the newest version. A "G" means that the file was updated with the newest version, but you had some local changes in the file, and SVN merged the two just fine. A "C" means there was a conflict, but we'll talk about that in the next section. An "A" means that the file was recently added to the repo and that this is your first time you are receiving a copy. A "D" means the file was deleted. **You should update early and often.**

After you have made some changes to a file, you should *commit* them to the central repository, so that when everybody else updates, they get a copy of your changes. When you commit, you will be asked to supply a *log*, which is any written comments you want to make for your other users. You can also just specify the log message with `-m`.

```
% svn commit Lists/char-LIST.tex -m "Gave James Bond a pistol."
Sending Lists/char-LIST.tex
Transmitting file data.
Committed revision 43.
```

Now, whenever the other users update, their copy of `char-LIST.tex` will include your changes. You can commit individual files, or entire directories. **You should commit early and often.**

There's one more issue. If a version of the file already exists in the repository, you can update and commit normally. However, if you want to add a new file for the first time, you have to use the `add` command. This will schedule the file for inclusion next time you commit:

```
% svn add Charsheets/nickfury.tex
A Charsheets/nickfury.tex
% svn commit Charsheets/ -m "Added Nick's charsheet."
Sending Charsheets/nickfury.tex
Transmitting file data.
Committed revision 46.
```

Similarly, you can `add` individual files or entire directories all at once. When other users update, they will receive a copy of `nickfury.tex` for the first time. Then you can all update and commit changes to it normally.

Handling conflicts When Alice and Bob are make changes to the same file, Alice will commit her changes and Bob will have to update his local copy before commiting his changes. Most of the time, SVN will merge Alice's changes intelligently into Bob's local copy and integrate their changes (and Bob will see a "G" when he updates). Sometimes, for whatever reason[2], SVN doesn't trust itself that it knows how Alice and Bob want their changes merged. When this happens, Bob will see a "C" when updating, which stands for *conflict*.

```
% svn update
C Charsheets/warlock.tex
```

SVN will markup any file that is in *conflict* with special symbols to show you where the conflict is, showing Bob both Alice's version that is now in the repo and Bob's version, and where they disagree. Bob will have to manually merge the changes. When Bob looks at the file, he might see a lot of blocks like this:

```
<<<<<<< .mine
and so you came to Moscow to kill your enemy James Bond.
=======
and so you came to Moscow to help your friend Nick Fury.
>>>>>>> .r54
```

Every block between the less than and greater than signs is a part of the file that is in conflict. What Bob's local file has is

[2]Most likely, that Alice and Bob and were making changes to the exact same place.

shown after the `<<<<<<< .mine`. What the repository has (in this case, Alice's changes) is shown above the `>>>>>>> .r54`.[3] The `=======` deliniates between the two versions. Bob needs to get rid of this whole block and merge the two changes so that they make sense:

```
and so you came to Moscow to kill your enemy James Bond and help your friend Nick Fury.
```

When Bob is done editing the file so that it merges Alice's and his changes the way they want, he needs to tell SVN that he fixed the problem with the *resolved* command.

```
% svn resolved Charsheets/warlock.tex
Resolved conflicted status of Charsheets/warlock.tex !
```

Now Bob can commit normally. When Alice updates, SVN will give her the correct copy of `warlock.tex` with both sets of changes.

Afterword SVN is one of those things that you won't be able to imagine you lived without. It will help get your whole house in order. Commit and update early and often, and you will save yourself and your GM team a lot of pain. Type `svn help` for more documentation, or look it up online.

[3] The "54" is just indicating tha the current revision of the project in the repo is revision #54.