

# Submitting Assignments

Marius Nita and Travis Spencer

CS Tutors – Portland State University

July 26, 2005

## 1 Introduction

As of yet, the Portland State University CS department lacks a centralized system for assignment submission and the process is manual, involving a combination of creating a package and emailing it to the professor or grader, on the user's end. This process can become tedious and time consuming for the beginning user who is unfamiliar with the myriad of UNIX tools available. I provide a quick introduction to creating and submitting a homework package, describing all known methods for doing so.

## 2 Packaging your work

Typically, a student is asked to submit homework in one of two ways:

- in the body of a message, as plain-text
- in an attachment, as either plain-text or binary format

The former involves using `shar`, a UNIX tool which creates a self-extracting plain-text archive. The latter involves using `tar`, a UNIX archiving tool which produces a binary file, extractable with `tar` on the other end.

### 2.1 `shar`

To package several files (say `foo.c`, `bar.c`, and `baz.c`) into a `shar` archive, run the following command:

```
shar foo.c bar.c baz.c > ARCHIVE
```

where `ARCHIVE` is the name of the archive you are creating, typically something descriptive, e.g. `cs163_proj2.shar`. The resulting file will contain a plain-text archive of the files you specified in the command above. It is a good idea to verify that the archive has been created successfully by unpacking it in a scratch directory, by running the following in the same directory that you created the archive in:

```
mkdir tmp
cd tmp
sh ../ARCHIVE
```

Your project files should show up in the current directory if the archive is not corrupted. Finally, clean up the directory:

```
cd ..
rm -rf tmp
```

Note: be careful with the `rm` command!

## 2.2 tar and gzip

If you have several files with a non-trivial directory hierarchy, and/or are packaging binary files such as Word documents, you must use `tar` to produce the archive. Several classes will require that `tar` is used in the submission process.

If your project is in directory `proj`, the command

```
tar cvf proj.tar proj
```

will create a `tar` archive named `proj.tar` which contains the complete contents of the `proj` directory. The contents of the directory will be displayed as they are being packed into the archive. To extract the `tar` archive named `proj.tar`, you run

```
tar xvf proj.tar
```

which creates a directory `proj` containing all the files that you previously archived. Again, the files are being displayed as they are unpacked.

The `cvf` and `xvf` parts in the above commands are *flags* that specify `tar`'s behavior. They mean the following:

<code>x</code>	Extract (unpack) an archive.
<code>c</code>	Create an archive.
<code>v</code>	Be verbose. Display the files being packed.
<code>f</code>	The filename immediately following is the name of the archive.

If the project contains extremely large text files, such as verbose program output, one might benefit from **compressing** the files before they are archived. This is typically done with a program such as `gzip`, that takes a file and produces a highly compressed version of it. To compress a `tar` archive, run the following command:

```
gzip -c proj.tar > proj.tar.gz
```

Note that an extra `“.gz”` should be appended to the new filename, to indicate that it has been compressed. This is mainly for reasons of politeness toward the user of your archive. To decompress a compressed archive, run:

```
gunzip --decompress proj.tar.gz
```

which will leave a decompressed archive named `proj.tar` in the current directory.

### 2.2.1 GNU tar

GNU `tar` is a slightly more feature-rich version of `tar`. To make GNU `tar` available in your environment, run the command `addpkg gnu`, log out of your shell and log back in. GNU `tar` will afterward be available from the command line as `gtar`.

GNU `tar` has built-in support for `gzip`, activated by simply adding a `z` flag to the parameter list. Examples:

```
gtar zcvf proj.tar.gz proj      # create a compressed archive
gtar zxvf proj.tar.gz          # unpack a compressed archive
```

## 3 Mailing the package

As mentioned earlier, once a package is created, it can be mailed in two ways:

- in the body of a message, as plain-text
- in an attachment, as either plain-text or binary format

Obviously the former applies to `shar` archives (or single plain-text files) and the latter to `tar` or `tar.gz` archives.

### 3.1 In the body of a message

To mail a `shar` archive or any other plain-text file, one can use the `mailx` program, available universally on UNIX and Linux systems:

```
mailx -s SUBJECT EMAIL < ARCHIVE
```

where `SUBJECT` is the subject line in the email, `EMAIL` is the email address that the archive is sent to, and `ARCHIVE` is the file name for the plain-text file that you are sending. For example,

```
mailx -s "John Doe - CS161 Program 5" teacher@school.edu < prog5.shar
```

Note the double quotes, which are used when an argument has spaces in it.

It is a good idea to test-send the assignment to yourself before sending it to the teacher, by using your own email address for the `EMAIL` argument. This way, you have a chance to see exactly what the grader sees when they receive your email.

### 3.2 As an attachment

To email a `tar` file, one must *attach* it to an email. Furthermore, some classes require students to email their homework as attachments, regardless of whether they are plain-text or binary. Two UNIX mail programs exist which facilitate attaching files to emails: `pine` and `mutt`.

### 3.2.1 With pine

To attach a file to an email with `pine`, run the command

```
pine -attach FILENAME
```

A screen will come up in which you will need to fill out the `To` address, the subject line, and an optional body. Use the arrow keys to navigate up and down. Finally, hit **Control-x** to send the message. Hit **Control-c** to cancel at any time.

### 3.2.2 With mutt

To attach a file to an email with `mutt`, run

```
mutt -a FILENAME
```

A series of prompts will ask you for the `To` address, the subject line, and possibly other information. Press `Enter` to skip a prompt. You will be dropped into an editor (`vi` by default) in which you can enter an optional body. When you are done, save and quit the editor, and hit `y` at the final screen.

## 3.3 Recording Your Program's Execution

Often, instructors want a sample of your program's execution. The easiest way to do this is to use the `script` command. An example session follows (where “\$” stands for your UNIX prompt):

```
$ script
Script started, file is typescript
$ RUN YOUR PROGRAM AS USUAL HERE
$ exit
Script done, file is typescript
```

Invoking the `script` command causes a new shell to be started, activity in which will be recorded to a file named `typescript` in the current directory. When you are done running your program, type `exit` to end the session. You can then print the `typescript` file and turn it in as example output from your program.

Note that the `typescript` program records absolutely *everything*, including the fact that you hit Backspace, `Ctrl-a`, or any other key which you do not want reflected in the output. The tutors have written a program which cleans up `typescript` files. Although it does not do a complete job, it cleans up escape sequences which are likely to occur very often, e.g. backspaces.

The program can be downloaded from <http://www.cat.pdx.edu/tutors/files/fixts.cpp>. Once compiled, it can be run as follows:

```
fixts typescript > proj1.script
```

## 4 Further help

Email your questions to CS tutors at [tutors@cs.pdx.edu](mailto:tutors@cs.pdx.edu), or join our IRC channel, `#cschat` on [irc.cat.pdx.edu](http://irc.cat.pdx.edu).