Version Control

Kapil Hari Paranjape

The Institute of Mathematical Sciences

14th March 2008

Pre-Version Control

"Version Control" is about organising programs and documents. How does one organise one's documents and programs before one learns version control but *after* one has learnt to use some basic commands.

Pre-Version Control

"Version Control" is about organising programs and documents. How does one organise one's documents and programs before one learns version control but *after* one has learnt to use some basic commands.

1. The first step is to organise things into folders.

Pre-Version Control

"Version Control" is about organising programs and documents. How does one organise one's documents and programs before one learns version control but *after* one has learnt to use some basic commands.

1. The first step is to organise things into folders.
2. The second step is to create different files to do different things.

Pre-Version Control

"Version Control" is about organising programs and documents. How does one organise one's documents and programs before one learns version control but *after* one has learnt to use some basic commands.

1. The first step is to organise things into folders.
2. The second step is to create different files to do different things.
3. The third step is to test the result of the work done.

Pre-Version Control

"Version Control" is about organising programs and documents. How does one organise one's documents and programs before one learns version control but *after* one has learnt to use some basic commands.

1. The first step is to organise things into folders.
2. The second step is to create different files to do different things.
3. The third step is to test the result of the work done.
4. The fourth step is to properly save and export/send the result.

Pre-Version Control

"Version Control" is about organising programs and documents. How does one organise one's documents and programs before one learns version control but *after* one has learnt to use some basic commands.

1. The first step is to organise things into folders.
2. The second step is to create different files to do different things.
3. The third step is to test the result of the work done.
4. The fourth step is to properly save and export/send the result.
5. The final step is to clean-up the work so that anyone returning to look at it can figure out how the material is organised.

We often skip or truncate some of these steps! We will not discuss those cases (which deserve our *pity* rather than *annoyance*!) where even the steps on this slide are not followed.

What is missing?

The above approach is more than adequate for a large number of use cases. However, as one begins to collect a large body of work we discover some gaps:

What is missing?

The above approach is more than adequate for a large number of use cases. However, as one begins to collect a large body of work we discover some gaps:

1. There are a large number of different folders.

What is missing?

The above approach is more than adequate for a large number of use cases. However, as one begins to collect a large body of work we discover some gaps:

1. There are a large number of different folders.
2. We often have files which we are not sure whether to keep or not.

What is missing?

The above approach is more than adequate for a large number of use cases. However, as one begins to collect a large body of work we discover some gaps:

1. There are a large number of different folders.
2. We often have files which we are not sure whether to keep or not.
3. The testing worked at the time it was done but we can't verify those tests now.

What is missing?

The above approach is more than adequate for a large number of use cases. However, as one begins to collect a large body of work we discover some gaps:

1. There are a large number of different folders.
2. We often have files which we are not sure whether to keep or not.
3. The testing worked at the time it was done but we can't verify those tests now.
4. In cleaning up we delete (by mistake!) files which are important.

What is missing?

The above approach is more than adequate for a large number of use cases. However, as one begins to collect a large body of work we discover some gaps:

1. There are a large number of different folders.
2. We often have files which we are not sure whether to keep or not.
3. The testing worked at the time it was done but we can't verify those tests now.
4. In cleaning up we delete (by mistake!) files which are important.
5. We do not cleanup and after a while the folder is full of files with cryptic similar sounding names and we are not really sure which is the one that got the correct results!

What is missing?

The above approach is more than adequate for a large number of use cases. However, as one begins to collect a large body of work we discover some gaps:

1. There are a large number of different folders.
2. We often have files which we are not sure whether to keep or not.
3. The testing worked at the time it was done but we can't verify those tests now.
4. In cleaning up we delete (by mistake!) files which are important.
5. We do not cleanup and after a while the folder is full of files with cryptic similar sounding names and we are not really sure which is the one that got the correct results!
6. In an effort to keep all the relevant documentation in the file we keep all kinds of comments in the file which are not for the "public version".

What is missing?

The above approach is more than adequate for a large number of use cases. However, as one begins to collect a large body of work we discover some gaps:

1. There are a large number of different folders.
2. We often have files which we are not sure whether to keep or not.
3. The testing worked at the time it was done but we can't verify those tests now.
4. In cleaning up we delete (by mistake!) files which are important.
5. We do not cleanup and after a while the folder is full of files with cryptic similar sounding names and we are not really sure which is the one that got the correct results!
6. In an effort to keep all the relevant documentation in the file we keep all kinds of comments in the file which are not for the "public version".
7. It turns out the the work needs to be continued and developed further.

. . . and so on!

What is the ideal?

In an ideal setup, *after* we have finished the work we should have a folder containing files (and possibly sub-folders).

What is the ideal?

In an ideal setup, *after* we have finished the work we should have a folder containing files (and possibly sub-folders).

1. Each file/sub-folder has a recognisable name, or we have a `README` file which documents what each of these is about.

What is the ideal?

In an ideal setup, *after* we have finished the work we should have a folder containing files (and possibly sub-folders).

1. Each file/sub-folder has a recognisable name, or we have a `README` file which documents what each of these is about.
2. Each file/sub-folder is actually necessary for generating the program or document in final form. The files that can be automatically generated are handled by a `makefile` or following instructions given in `INSTALL`.

What is the ideal?

In an ideal setup, *after* we have finished the work we should have a folder containing files (and possibly sub-folders).

1. Each file/sub-folder has a recognisable name, or we have a `README` file which documents what each of these is about.
2. Each file/sub-folder is actually necessary for generating the program or document in final form. The files that can be automatically generated are handled by a `makefile` or following instructions given in `INSTALL`.
3. We have clear history of how the files have been changed and why.

What is the ideal?

In an ideal setup, *after* we have finished the work we should have a folder containing files (and possibly sub-folders).

1. Each file/sub-folder has a recognisable name, or we have a `README` file which documents what each of these is about.
2. Each file/sub-folder is actually necessary for generating the program or document in final form. The files that can be automatically generated are handled by a `makefile` or following instructions given in `INSTALL`.
3. We have clear history of how the files have been changed and why.
4. Older "tricks" hidden in earlier versions of files can be recovered.

. . . and so on!