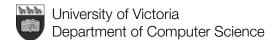
svn commands

- Note:
 - We've used "commands" as a synonym for "UNIX command"
 - Yet svn itself is a UNIX command
- An svn command is how we specify an action from the svn client
- Syntax
 - svn command [option] [arguments]



Previously seen

\$	svn	checkout	http://svn.example.com/repos/calc	
\$	svn	commit	button.c	
\$	svn	update		
Ψ	3 411	араасс		

Subversion command

argument



A few more examples

\$ s	vn checkout	-r 4	http://svn.example.com/repos/calc
\$ 5	svn commit	-m "Fixed typo in label"	button.c
		-	
\$ 5	ovn update	quiet	
D 3	ovii upuate	quiet	

Subversion option argument command



Some useful commands

add

blame

commit

delete

export

import

log

move

status

diff

update

add files and directories to the project

show detailed author and revision information for files

send changes from working copy to repository

delete item from working copy or repository

create a clean copy of the repository

recursively commit a copy of a local directory to a repository

display commit messages (the "log")

move a file or directory to a different place in the project

display file state of working copy file (or files or directories)

display differences between working copy and repository

send changes from repository to working copy



Basic cycle of use: revisited

- checkout a local copy of a project from a repository
- 2. in your local copy of the project (which is a directory)
 - update your local copy
 - build / run / test / view / render / read / <fill-in-verb> your work
 - if you have added files or directories to your working copy, make sure they are added to svn control
- 3. if not yet ready to **commit** changes, go to step 2
- **4. commit** your changes to the repository
- 5. go to step 2

- obtain copy of project
 - svn checkout
- update your working copy
 - svn update
- make changes
 - svn add
 - svn delete
 - svn copy
 - svn move
- examine your changes
 - svn status
 - svn diff
 - svn revert
- merge other's changes
 - svn merge
 - svn resolve
- commit your changes
 - svn commit



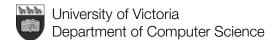
If you need help...

- For a specific command:
 - svn help <command>
 - Provides list of arguments and options
- For info on repository access
 - speak to the provider, or
 - read the provided documentation
 - example: http://labs.seng.uvic.ca/svn



Usage notes

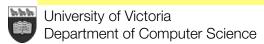
- Common practice:
 - You have an idea
 - You write a bit of code from scratch
 - Code base gets a bit bigger, you create some directories
 - Codebase gets large enough to need source-code control...
 - or codebase could now be expanded in collaboration with others
 - or both
- Note all this is done outside of Subversion
- How do we get the code base into a repository?
- Use svn import



Importing

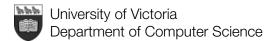
- Must specify the local directory that will be copied over to repository
- Must specify repository
 - Repository string also indicates where in the repository to import the file
- Important: After this step, local directory is not erased, nor is it under Subversion control.

Committed revision 1.



Adding and Deleting

- Assumes your are manipulating a working copy
- If you have added or deleted files or directories...
- ... and you want to publish these additions/deletions
- ... then you must explicitly state this
 - svn add <file or dir>
 - svn delete <file or dir>
- Note that this only changes the Subversion metadata stored in your working copy (i.e., your local copy)
 - These actual changes are only transferred/copies to the repository on the next commit
 - Note that other changes from other users may cause conflicts...





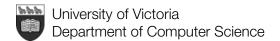
Never store generated files in the repository.

For example, if your project includes C source code, you would store the .c and .h files. You would not store the .o or executables.

If your project includes Java source code, you would store the .java file, but <u>you wouldn't</u> store the generated .class or .jar files.

Taking "snapshots"

- What if you have a version of your project...
- ... and you would like to take a snapshot of that project?
 - Can be useful for release management (i.e., can recover source for an earlier release at a later date)
 - Also useful for versioning and testing workflows
- One way is to make a copy of it.



Copying

```
$ svn copy https://svn.example.com/repos/calc/trunk \
   https://svn.example.com/repos/calc/tags/release-1.0 \
   -m "Tagging the 1.0 release of the 'calc' project."

Committed revision 902.
$
```

Note:

- Assumes were have already created a subdirectory named tags
- Assumes we keep track of the meaning of each release (ie. what "release-1.0" denotes)
- Slightly complicated command as both source and destination can be either WC or URL.



What about "conflicts"?

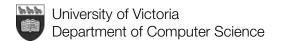
- Suppose you've updated your project and you see the report on the right:
- Codes:
 - U means changes to file are absorbed cleanly
 - G means merged (i.e., there were changes, but they overlapped cleanly with local changes)
 - A means added
 - D means deleted
 - C means conflict
- Conflicts must be resolved...

```
$ svn update
U INSTALL
G README
C bar.c
Updated to revision 46
```



Conflicts

- Means:
 - Repository copy of a file has had changes made to it since you last committed / updated / checked out that file
 - Your working copy of the file has had changes made to it since the last commit / update / checkout
 - These changes overlap in a way that svn cannot resolve
- To resolve a conflict:
 - You must examine the overlaps
 - Then you must manually choose between them ("resolved")
- Subversion does help us by giving us as much information as possible



Subversion's "conflict" assistance

- Prints a C during the update (visual cue)
- Places conflict markers into the file to visibly demonstrate overlapping areas
- For every file in conflict, places three extra files in your working copy
 - filename.mine: File as it existed in your working copy before the update (i.e., no markers)
 - filename.rOLDREV: this is the BASE revision before you updated (i.e., what is was before last series of edits)
 - filename.rNEWREV: this is the version your Subversion client just received



Steps to resolve the conflict

- At this point you will not be allowed to commit the "conflicted" file until all three extra files are removed!
- You must do one of three things:
 - merge file's text by hand (i.e., examine and edit conflict markers in the file), or
 - copy one of the temporary files on top of your working file, or
 - run svn revert <filename> to throw away all your local changes
- After you have performed one of these steps, you let Subversion know the conflict is resolved
 - svn resolved <filename>
- This last command removes all extra files.

