DEVELOPER TOOLS

- Eclipse and Java versions
- Remote access
- Version control
- Eclipse debugging
WHAT IS AN SSH CLIENT?

- Uses the secure shell protocol (SSH) to connect to a remote computer
  + Enables you to work on a lab machine from home
  + Similar to remote desktop
- Windows users: PuTTY and WinSCP
  + PuTTY: ssh connection
  + WinSCP: transfer or edit files
- Mac/Linux users: Terminal application
  + Go to Applications/Utilities/Terminal
  + Type in “ssh cseNetID@attu.cs.washington.edu”
  + “ssh -XY cseNetID@attu.cs.washington.edu” lets you use GUIs
PUTTY

PuTTY Configuration

Category:

Session
- Logging
- Terminal
- Keyboard
- Bell
- Features

Window
- Appearance
- Behaviour
- Translation
- Selection
- Colours

Connection
- Data
- Proxy
- Telnet
- Rlogin
- SSH
- Serial

Basic options for your PuTTY session

Specify the destination you want to connect to
Host Name (or IP address) Port
meganca@attu.cs.washington.edu 22

Connection type:
- Raw
- Telnet
- Rlogin
- SSH
- Serial

Load, save or delete a stored session
Saved Sessions
Default Settings
meganca@helsinki.cs.washington.edu

Close window on exit:
- Always
- Never
- Only on clean exit

Using username "meganca"
meganca@attu.cs.washington.edu's password:

Please remove core files when you are done with them, as they tend to take up a lot of space on the disk. If everyone removes them when they are done debugging, there is going to be a lot more disk space to go around.

Thanks!

[meganca@attu ~]$

Contact support@cs if you need assistance.
TERMINAL (LINUX, MAC)

meganca@charmander:~$ ssh meganca@attu.cs.washington.edu
meganca@attu.cs.washington.edu's password:

Use passwd to change your password.
Use chsh to change your shell.

Contact support@cs if you need assistance.

Please remove core files when you are done with them, as they tend to take up a lot of space on the disk. If everyone removes them when they are done debugging, there is going to be a lot more disk space to go around.

Thanks!

[meganca@attu3 ~]$
• Get Java 7
• Important: Java separates compile and execution, eg:
  + javac Example.java
    Example.class
  + Both compile and execute have to be the same Java!
**WHAT IS UNIX?**

- Multiuser modular operating system
  - Traditionally command-line based
  - Mac OS X is Unix-based!

<table>
<thead>
<tr>
<th>Command</th>
<th>What it does</th>
</tr>
</thead>
<tbody>
<tr>
<td>pwd</td>
<td>prints the name of the working directory</td>
</tr>
<tr>
<td>ls</td>
<td>lists the files in a directory (i.e., lists stuff)</td>
</tr>
<tr>
<td>cd</td>
<td>changes a directory</td>
</tr>
<tr>
<td>cp</td>
<td>copies a file or directory</td>
</tr>
<tr>
<td>mv</td>
<td>move/rename a file or directory</td>
</tr>
<tr>
<td>rm</td>
<td>removes a file</td>
</tr>
<tr>
<td>mkdir</td>
<td>make a new directory</td>
</tr>
<tr>
<td>rmdir</td>
<td>remove an empty directory</td>
</tr>
<tr>
<td>man</td>
<td>pulls up the manual pages</td>
</tr>
</tbody>
</table>
WHAT IS VERSION CONTROL?

• Also known as source control/revision control
• System for tracking changes to code
  + Software for developing software
• Essential for managing projects
  + See a history of changes
  + Revert back to an older version
  + Merge changes from multiple sources
• We’ll be talking about Subversion, but there are alternatives
  ✓ Git, Mercurial, CVS
  ✗ Email, Dropbox, USB sticks
A repository stores the master copy of the project
- Someone creates the repo for a new project
- Then nobody touches this copy directly
- Lives on a server everyone can access

Each person checks out her own working copy
- Makes a local copy of the repo
- You’ll always work off of this copy
- The version control system syncs the repo and working copy (with your help)
• Can create the repository anywhere
  + Can be on the same computer that you’re going to work on, which might be ok for a personal project where you just want rollback protection

• But, usually you want the repository to be robust:
  + On a computer that’s up and running 24/7
    - Everyone always has access to the project
  + On a computer that has a redundant file system
    - No more worries about that hard disk crash wiping away your project!

• We’ll use attu! (attu.cs.washington.edu)
VERSION CONTROL
COMMON ACTIONS

Most common commands:

- **Commit / checkin**
  - integrate changes *from* your working copy *into* the repository

- **Update**
  - integrate changes *into* your working copy *from* the repository
**Version Control**

**Common Actions (Cont.)**

**More common commands:**

- **Add, delete**
  - add or delete a file in the repository
  - just putting a new file in your working copy does not add it to the repo!

- **Revert**
  - wipe out your local changes to a file

- **Resolve, diff, merge**
  - handle a conflict – two users editing the same code
VERSION CONTROL

Repository

update
svn
commit

Working copy
331 VERSION CONTROL

- create
- check out
- commit
- update
- add

Repository

Working copy for grading

Working copy
331 VERSION CONTROL

- Your repo is at
  /projects(instr/15wi/cse331/YourCSENetID/R EPOS/cse331
- Only check out once (unless you’re working in a lot of places)
- Don’t forget to add files!!
- Check in your work!
# Version Control: Command-Line

<table>
<thead>
<tr>
<th>command</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>svn co repo</code></td>
<td>check out</td>
</tr>
<tr>
<td><code>svn ci [files]</code></td>
<td>commit / check in changed files</td>
</tr>
<tr>
<td><code>svn add files</code></td>
<td>schedule files to be added at next commit</td>
</tr>
<tr>
<td><code>svn help [command]</code></td>
<td>get help info about a particular command</td>
</tr>
<tr>
<td><code>svn merge source1 source2</code></td>
<td>merge changes</td>
</tr>
<tr>
<td><code>svn revert files</code></td>
<td>restore local copy to repo's version</td>
</tr>
<tr>
<td><code>svn resolve files</code></td>
<td>resolve merging conflicts</td>
</tr>
<tr>
<td><code>svn update [files]</code></td>
<td>update local copy to latest version</td>
</tr>
</tbody>
</table>

Others: blame, changelist, cleanup, diff, export, ls/mv/rm/mkdir, lock/unlock, log, propset
This Quarter

- We distribute starter code by adding it to your **repo**
- You will **code** in Eclipse
- You turn in your files by **adding** them to the repo and **committing** your changes
- You will **validate** your homework by **SSHing** onto attu and running an Ant build file
HOW TO USE SUBVERSION

1. Eclipse plugin: Subclipse
2. GUI interface: TortoiseSVN, NautilusSVN
3. Command line: PuTTY
DEMO #2

WHAT IS ECLIPSE?

- Integrated development environment (IDE)
- Allows for software development from start to finish
  + Type code with syntax highlighting, warnings, etc.
  + Run code straight through or with breakpoints (debug)
  + Break code
- Mainly used for Java
  + Supports C, C++, JavaScript, PHP, Python, Ruby, etc.
- Alternatives
  + NetBeans, Visual Studio, IntelliJ IDEA
## Eclipse Shortcuts

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl + D</td>
<td>Delete an entire line</td>
</tr>
<tr>
<td>Alt + Shift + R</td>
<td>Refactor (rename)</td>
</tr>
<tr>
<td>Ctrl + Shift + O</td>
<td>Clean up imports</td>
</tr>
<tr>
<td>Ctrl + /</td>
<td>Toggle comment</td>
</tr>
<tr>
<td>Ctrl + Shift + F</td>
<td>Make my code look nice 😊</td>
</tr>
</tbody>
</table>
ECLIPSE DEBUGGING

• System.out.println() works for debugging…
  + It’s quick
  + It’s dirty
  + Everyone knows how to do it
• …but there are drawbacks
  + What if I’m printing something that’s null?
  + What if I want to look at something that can’t easily be printed (e.g., what does my binary search tree look like now)?
• Eclipse’s debugger is powerful…if you know how to use it
ECLIPSE DEBUGGING

```java
@Test
public void testDupWithOneVal() {
    RatPolyStack stk1 = stack("3");
    stk1.dup();
    assertStackSize(stk1, "33");
    stk1 = stack("123");
    stk1.dup();
    assertStackSize(stk1, "1233");
```
Double click in the grey area to the left of your code to set a breakpoint. A breakpoint is a line that the Java VM will stop at during normal execution of your program, and wait for action from you.
Click the Bug icon to run in Debug mode. Otherwise your program won’t stop at your breakpoints.
Controlling your program while debugging is done with these buttons.
ECLIPSE DEBUGGING

Play, pause, stop work just like you’d expect
Step Into

Steps into the method at the current execution point – if possible. If not possible then just proceeds to the next execution point.

If there’s multiple methods at the current execution point step into the first one to be executed.
Step Over

Steps over any method calls at the current execution point.

Theoretically program proceeds just to the next line.

BUT, if you have any breakpoints set that would be hit in the method(s) you stepped over, execution will stop at those points instead.
Step Out

Allows method to finish and brings you up to the point where that method was called.

Useful if you accidentally step into Java internals (more on how to avoid this next).

Just like with step over though you may hit a breakpoint in the remainder of the method, and then you’ll stop at that point.
Enable/disable step filters

There’s a lot of code you don’t want to enter when debugging, internals of Java, internals of JUnit, etc.

You can skip these by configuring step filters.

Checked items are skipped.
ECLIPSE DEBUGGING

Stack Trace

Shows what methods have been called to get you to current point where program is stopped.

You can click on different method names to navigate to that spot in the code without losing your current spot.
**Variables Window**

Shows all variables, including method parameters, local variables, and class variables, that are in scope at the current execution spot. Updates when you change positions in the stackframe. You can expand objects to see child member values. There’s a simple value printed, but clicking on an item will fill the box below the list with a pretty format.

Some values are in the form of `ObjectName (id=x)`, this can be used to tell if two variables are referring to the same object.
Variables that have changed since the last break point are highlighted in yellow.

You can change variables right from this window by double clicking the row entry in the Value tab.
Variables that have changed since the last break point are highlighted in yellow.

You can change variables right from this window by double clicking the row entry in the Value tab.
There’s a powerful right-click menu.

- See all references to a given variable
- See all instances of the variable’s class
- Add watch statements for that variable’s value (more later)
Show Logical Structure

Expands out list items so it’s as if each list item were a field (and continues down for any children list items)
Breakpoints Window

Shows all existing breakpoints in the code, along with their conditions and a variety of options.

Double clicking a breakpoint will take you to its spot in the code.
Enabled/Disabled Breakpoints

Breakpoints can be temporarily disabled by clicking the checkbox next to the breakpoint. This means it won’t stop program execution until re-enabled.

This is useful if you want to hold off testing one thing, but don’t want to completely forget about that breakpoint.
Hit count

Breakpoints can be set to occur less-frequently by supplying a hit count of $n$.

When this is specified, only each $n$-th time that breakpoint is hit will code execution stop.
Conditional Breakpoints

Breakpoints can have conditions. This means the breakpoint will only be triggered when a condition you supply is true. **This is very useful** for when your code only breaks on some inputs!

Watch out though, it can make your code debug very slowly, especially if there’s an error in your breakpoint.
Disable All Breakpoints

You can disable all breakpoints temporarily. This is useful if you’ve identified a bug in the middle of a run but want to let the rest of the run finish normally.

Don’t forget to re-enable breakpoints when you want to use them again.
Break on Java Exception

Eclipse can break whenever a specific exception is thrown. This can be useful to trace an exception that is being “translated” by library code.
Expressions Window

Used to show the results of custom expressions you provide, and can change any time.

Not shown by default but highly recommended.
Expressions Window

Used to show the results of custom expressions you provide, and can change any time.

Resolves variables, allows method calls, even arbitrary statements “2+2”

Beware method calls that mutate program state – e.g. stk1.clear() or in.nextLine() – these take effect immediately
Expressions Window

These persist across projects, so clear out old ones as necessary.
ECLIPSE DEBUGGING

• The debugger is awesome, but not perfect
  + Not well-suited for time-dependent code
  + Recursion can get messy
• Technically, we talked about a “breakpoint debugger”
  + Allows you to stop execution and examine variables
  + Useful for stepping through and visualizing code
  + There are other approaches to debugging that don’t involve a debugger