SECTION 2:
CODE REASONING + PROGRAMMING TOOLS

cse331-staff@cs.washington.edu

slides borrowed and adapted from Alex Mariakis and CSE 390a

OUTLINE

- **Reasoning about code**
- **Developer tools**
  - Eclipse and Java versions
  - ssh
  - Version control

REASONING ABOUT CODE

- **Two purposes**
  - *Prove* our code is correct
  - Understand *why* code is correct
- **Forward reasoning: determine what follows from initial conditions**
- **Backward reasoning: determine sufficient conditions to obtain a certain result**

FORWARD REASONING

```
// {x >= 0, y >= 0}
y = 16;
//
x = x + y
//
x = sqrt(x)
//
y = y - x
//
```

FORWARD REASONING

```
// {x >= 0, y >= 0}
y = 16;
// {x >= 0, y = 16}
x = x + y
//
x = sqrt(x)
//
y = y - x
//
```

FORWARD REASONING

```
// {x >= 0, y >= 0}
y = 16;
// {x >= 0, y = 16}
x = x + y
// {x >= 16, y = 16}
x = sqrt(x)
//
y = y - x
//
```

FORWARD REASONING

```
// {x >= 0, y >= 0}
y = 16;
// {x >= 0, y = 16}
x = x + y
// {x >= 16, y = 16}
x = sqrt(x)
// {x >= 4, y = 16}
y = y - x
//
```

FORWARD REASONING

```
// {x >= 0, y >= 0}
y = 16;
// {x >= 0, y = 16}
x = x + y
// {x >= 16, y = 16}
x = sqrt(x)
// {x >= 4, y = 16}
y = y - x
// {x >= 4, y <= 12}
```

FORWARD REASONING

```
// {true}
if (x>0) {
        //
        abs = x
        //
}
else {
        //
        abs = -x
        //
}
//
//
```

FORWARD REASONING

```
// {true}
if (x>0) {
        // {x > 0}
        abs = x
        //
}
else {
        // {x <= 0}
        abs = -x
        //
}
//
//
```

FORWARD REASONING

```
// {true}
if (x>0) {
        // {x > 0}
        abs = x
        // {x > 0, abs = x}
}
else {
        // {x <= 0}
        abs = -x
        // {x <= 0, abs = -x}
}
//
//
```

FORWARD REASONING

```
// {true}
if (x>0) {
        // {x > 0}
        abs = x
        // {x > 0, abs = x}
}
else {
        // {x <= 0}
        abs = -x
        // {x <= 0, abs = -x}
}
// {x > 0, abs = x OR x <= 0, abs = -x}
//
```

FORWARD REASONING

```
// {true}
if (x>0) {
        // {x > 0}
        abs = x
        // {x > 0, abs = x}
}
else {
        // {x <= 0}
        abs = -x
        // {x <= 0, abs = -x}
}
// {x > 0, abs = x OR x <= 0, abs = -x}
// {abs = |x|}
```

BACKWARD REASONING

```
//
a = x + b;
//
c = 2b - 4
//
x = a + c
// {x > 0}
```

BACKWARD REASONING

```
//
a = x + b;
//
c = 2b - 4
// {a + c > 0}
x = a + c
// {x > 0}
```

BACKWARD REASONING

```
//
a = x + b;
// {a + 2b - 4 > 0}
c = 2b - 4
// {a + c > 0}
x = a + c
// {x > 0}
```

BACKWARD REASONING

```
// {x + 3b - 4 > 0}
a = x + b;
// {a + 2b - 4 > 0}
c = 2b - 4
// {a + c > 0}
x = a + c
// {x > 0}
```

IMPLICATION

- **Hoare triples are just an extension of logical implication**
  - Hoare triple: {P} S {Q}
  - P → Q after statement S

P	Q	P → Q
T	T	
T	F	
F	T	
F	F	

IMPLICATION

- **Hoare triples are just an extension of logical implication**
  - Hoare triple: {P} S {Q}
  - P → Q after statement S
- **Everything implies true**
- **False implies everything**

P	Q	P → Q
T	T	T
T	F	F
F	T	T
F	F	T

WEAKER VS. STRONGER

- **If P1 → P2, then**
  - P1 is stronger than P2
  - P2 is weaker than P1
- **Weaker statements are more general, stronger statements say more**
- **Stronger statements are more restrictive**
- **Ex: `x = 16` is stronger than `x > 0`**
- **Ex: `"Alex is an awesome TA"` is stronger than `"Alex is a TA"`**

WEAKEST PRECONDITION

- **The most lenient assumptions such that a postcondition will be satisfied**
- **If P* is the weakest precondition for {P} S {Q}, then P → P* for all P that make the Hoare triple valid**
- **WP = wp(S, Q), which can be found using backward reasoning**
  - Ex: wp(x = y+4, x > 0) = y+4>0

DEVELOPER TOOLS

- **Eclipse and Java versions**
- **Remote access**
- **Version control redux**

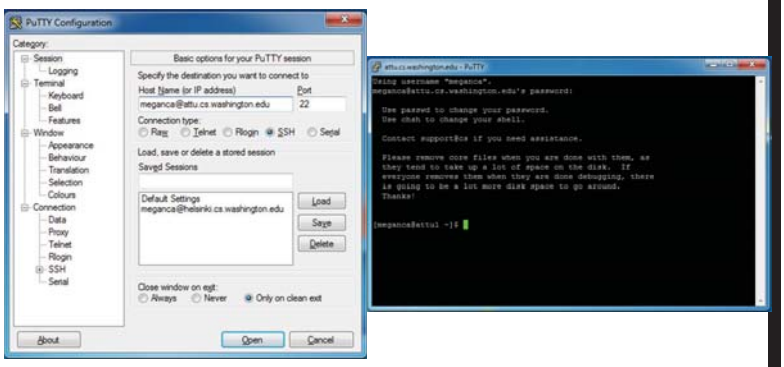
ECLIPSE

- Get Java 7
- Important: Java separates compile and execution, eg:
  - javac Example.java —produces→ Example.class
  - Both compile and execute have to be the same Java!

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



TERMINAL (LINUX, MAC)

DEMO #1

http://courses.cs.washington.edu/courses/cse331/14au/tools/WorkingAtHome.html

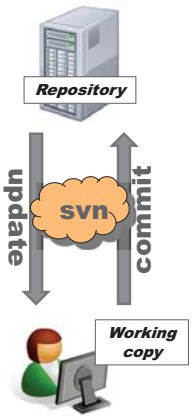
WHAT IS UNIX?

Multiuser modular operating system

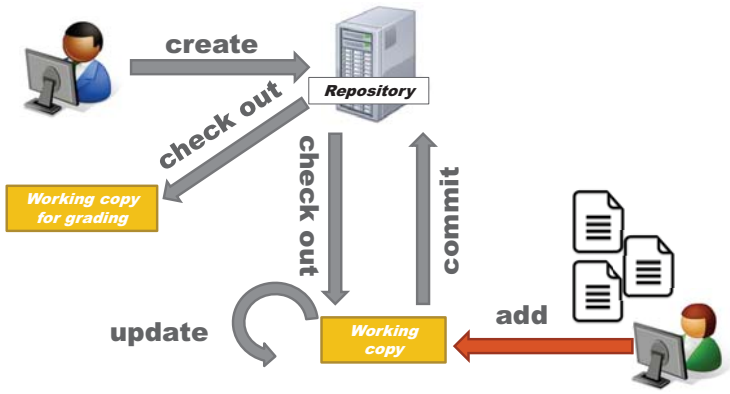
- Traditionally command-line based
- Mac OS X is Unix-based!

Command	What it does
pwd	**p**rints the name of the **w**orking **d**irectory
ls	lists the files in a directory (i.e., **l**ists **s**tuff)
cd	**c**hanges a **d**irectory
cp	**c**o**p**ies a file or directory
mv	**m**o**v**e/rename a file or directory
rm	**r**e**m**oves a file
mkdir	**m**a**k**e a new **dir**ectory
rmdir	**r**e**m**ove an empty **dir**ectory
man	pulls up the **man**ual pages

VERSION CONTROL



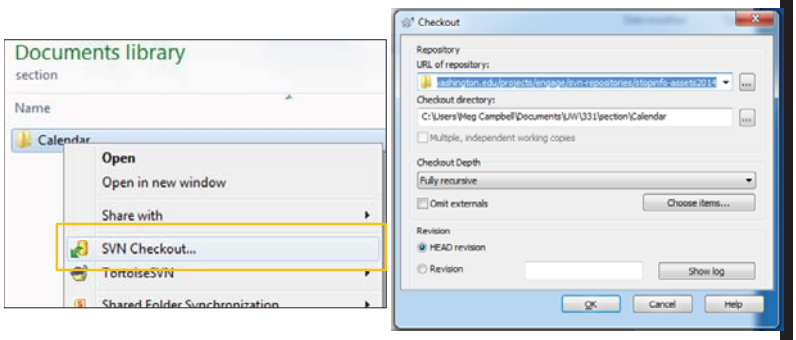
331 VERSION CONTROL



331 VERSION CONTROL

- Your repo is at /projects/instr/14au/cse331/*YourCSENetID*/REPOS/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: GUI



VERSION CONTROL: COMMAND-LINE

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

others: `blame`, `changelist`, `cleanup`, `diff`, `export`, `ls/mv/rm/mkdir`, `lock/unlock`, `log`, `propset`

DEMO #2

https://courses.cs.washington.edu/courses/cse331/14au/tools/versioncontrol.html