Version Control Using Subversion
Version Control

• In team projects, software engineers:
  – Share and extend a common code base (and comply with standards, coding conventions, comment templates, …)
  – Work *concurrently* with each other

• **Best practice** is for a team to use a *version control system*
  – We will use one called *Subversion*, but others are essentially similar
Version Control

• In team projects:
  – Share and extend a common code base (and comply with standards, coding conventions, comment templates, …)
  – Work concurrently with each other

• **Best practice** is for a team to use a **version control system**
  – We will use one called **Subversion**, but others are essentially similar

This is not limited to code! A version control system can handle non-code files as well, which makes it handy for other sorts of team projects, too.
Key Idea: The Repository

- A central **repository** keeps all files (in our case, Java code) and a history of all modifications to them
  - A new team member can **check out** their own private copy from the repository
  - Each member can **update** their own copy to reflect the latest changes in the repository
  - Each member can **commit** changes from their own private copy to the repository
Workflow Model: An Example

- Repository
- Commit
- Update
- Commit
- Matt
- Ayesha
How Work Gets Done

• Repository holds *master copy* of all files
  – Never edited directly
  – Stores complete history, too!
• Each team member has a *local copy* (or *working copy*) in their own workspace
  – All file creation, editing, deletion occurs here
• Update and commit commands are used to *synchronize* local and master copies
The “Optimistic” Model

- Any team member can modify their local copy of any file at any time
  - No “locking” or other synchronization among team members takes place on local copies
- On an update, the latest version from the repository often can be merged automatically into the local copy
  - This is especially so when team members edit different files, so conflicts are rare
Some Things That Can Happen

update

commit

Matt

repository

Ayesha
Some Things That Can Happen

Matt does an update to get the latest version to work on.
Some Things That Can Happen

- update
- commit

Matt

repository

Ayesha

Ayesha does an update to get the latest version to work on.
Some Things That Can Happen

Matt does a commit to put his latest edits into the repository.
Some Things That Can Happen

update
commit

Ayesha does an update to get the latest version before she commits.
The latest revision in the repository is *merged* into Ayesha’s copy.
Some Things That Can Happen

Suppose this merge is successful; then Ayesha commits.
Some Things That Can Happen

Matt tries to commit more changes, but has not updated recently.
Some Things That Can Happen

Ayesha has committed recently, and Matt may not do so now.

- Error: working version out-of-date
- Merge

Matt

repository

Ayesha
Some Things That Can Happen

- update
- commit

Matt needs to update first.

Matt
repository
Ayesha

Error: working version out-of-date
Some Things That Can Happen

Suppose this merge has conflicts; then Matt must resolve them.

Error: working version out-of-date
Conflict: requires attention
Some Things That Can Happen

After Matt resolves conflicts, he can continue and commit his changes.

Matt

repository

Ayesha

update

commit

Merge

Error: working version out-of-date

Conflict: requires attention
Some Things That Can Happen

Meanwhile, Ayesha updates and continues with the latest version.

Error: working version out-of-date
Conflict: requires attention

Matt
repository
Ayesha

update
commit

Merge

Error: working version out-of-date
Conflict: requires attention