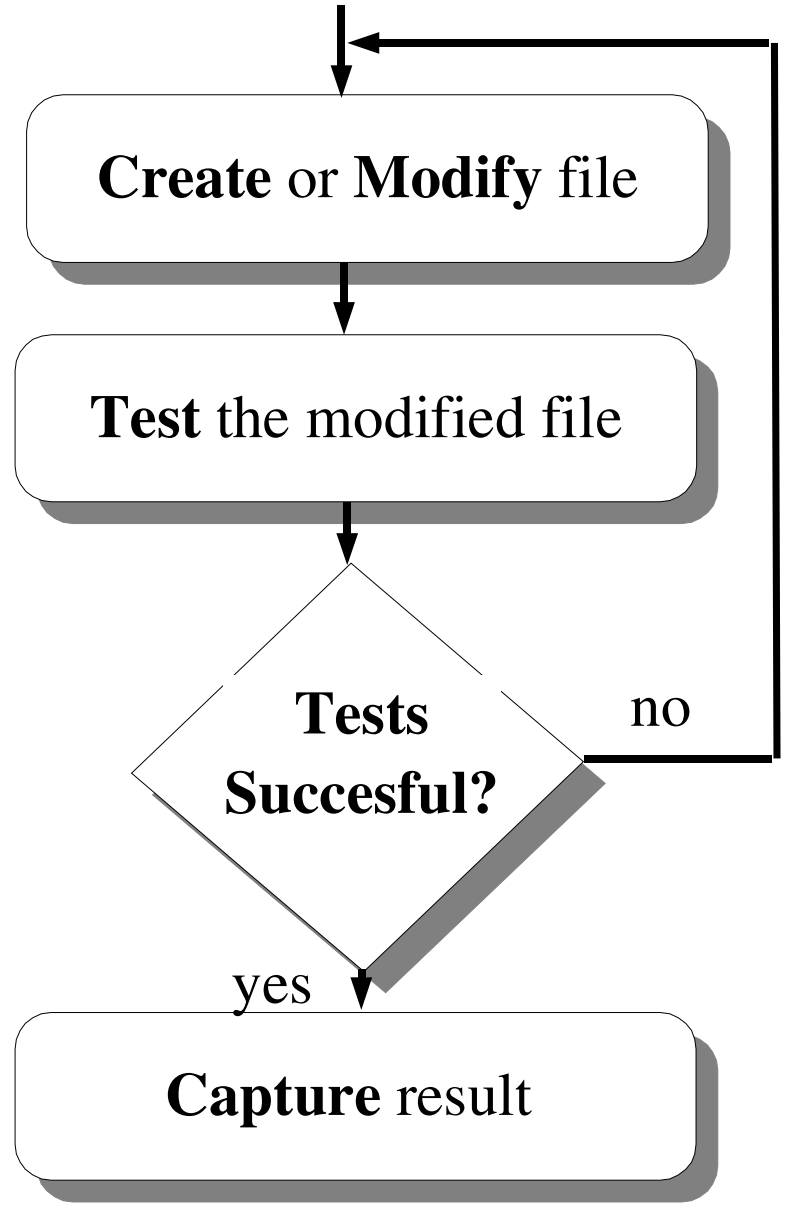
Version Control

CS2023 Winter 2004

Outcomes: Version Control

- *Applying RCS and SCCS*, by Don Bolinger and Tan Bronson, on reserve in the library. Official RCS homepage: www.cs.purdue.edu/homes/trinkle/RCS/
- After the conclusion of this section you should be able to
  - Use RCS for personal version control for personal projects, and for collaboration on small projects
  - Understand the principles of version control, so that other VC systems (CVS, Subversion, ...) can be mastered quickly

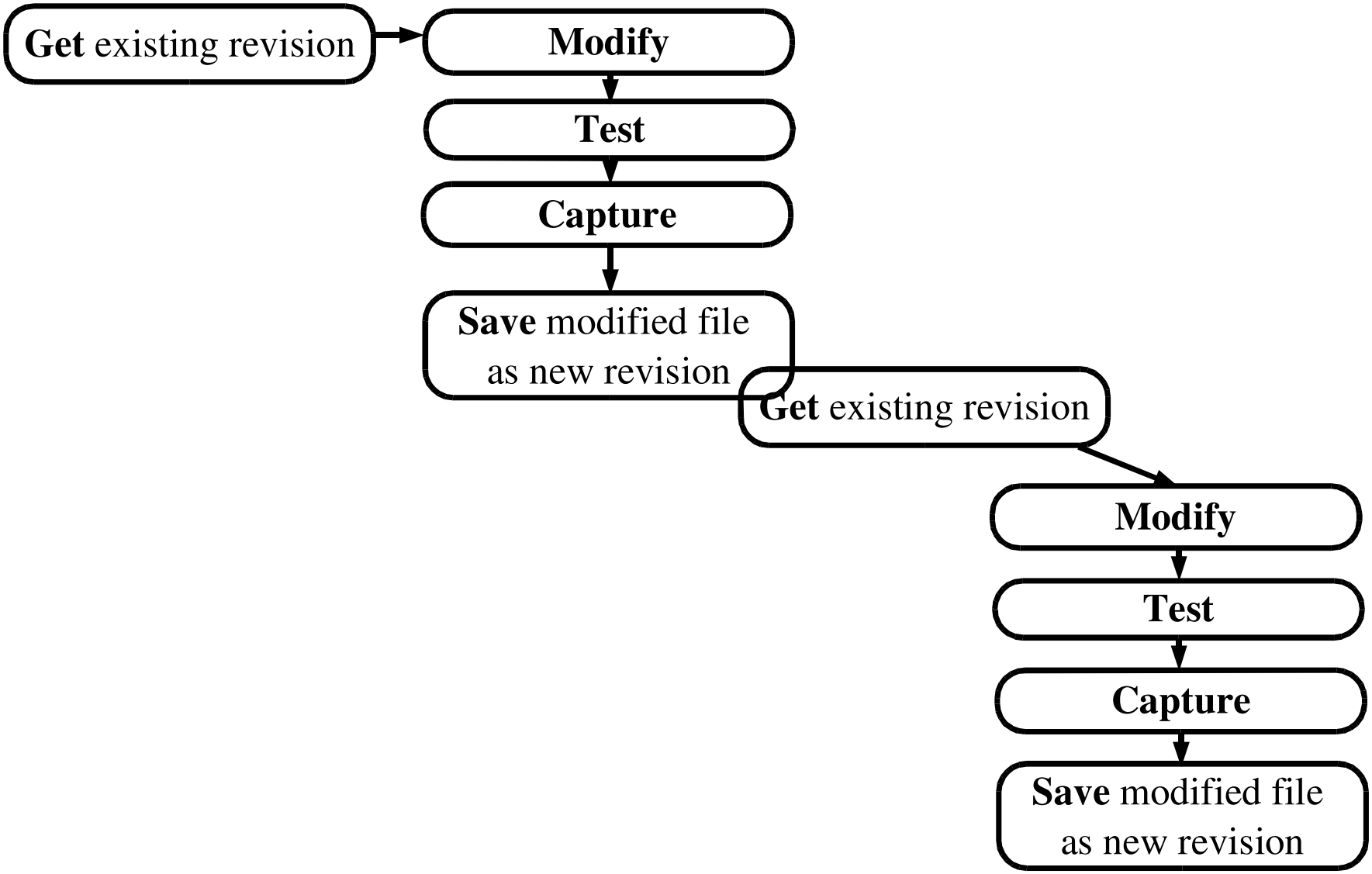
Source File Modification Cycle



Managing Source Code Files and Revisions

- Very important to be able to trace the changes made throughout the life of sofware
- May want to revert to a previous version of a file, in the event of a problem, or simply to compare versions
- Could create a backup of each version
- Better way is to have changes to a file tracked for us
  - this is what version control does

Modification Cycle with Archiving



Goals of Source Control

- Ability to record file revisions
- Ability to retrieve previous file revisions
- Control over new revision creation
  - sometimes need to modify older version
- Ability to record why a new revision was made
- Control of file modification
  - locking of revisions
- Easy access to all file revisions
  - all revisions kept in a single file

Basics of Source Control

- So what do I archive?
  - dynamic source files
  - files that can't be reconstructed from other files
- What don't I archive
  - derived files
  - static (read-only files)

Basics of Source Control

- Nomenclature
  - **revision**
    - each archived version of a source file
    - associated with a revison number
  - **check-in**
    - adding a new revision to an archive file
  - **check-out**
    - removing new revision from an archive file
  - **working file**
    - source file revision checked out

Basics of Source Control

- Source file revision numbers
  - *n.m.* : major (*n)* and minor (*m*) revision numbers
  - 1.1, 1.2, ...
  - Not necessarily the same as release numbers
- Source file revision storage
  - only one revision of source file stored in literal form
  - all others stored as differences (or diffs) from that single revision
    - saves disk space!

Basics of Source Control

- Log
  - archive file also contains reasons *why* each revision was checked-in
- Steps in source control:
  - create an archive file for each source file
  - get a working file for reading
    - source control system forces you to say explicitly when you intend to modify a file revision
    - OK if you just want to compile or read
  - get a working file for modification

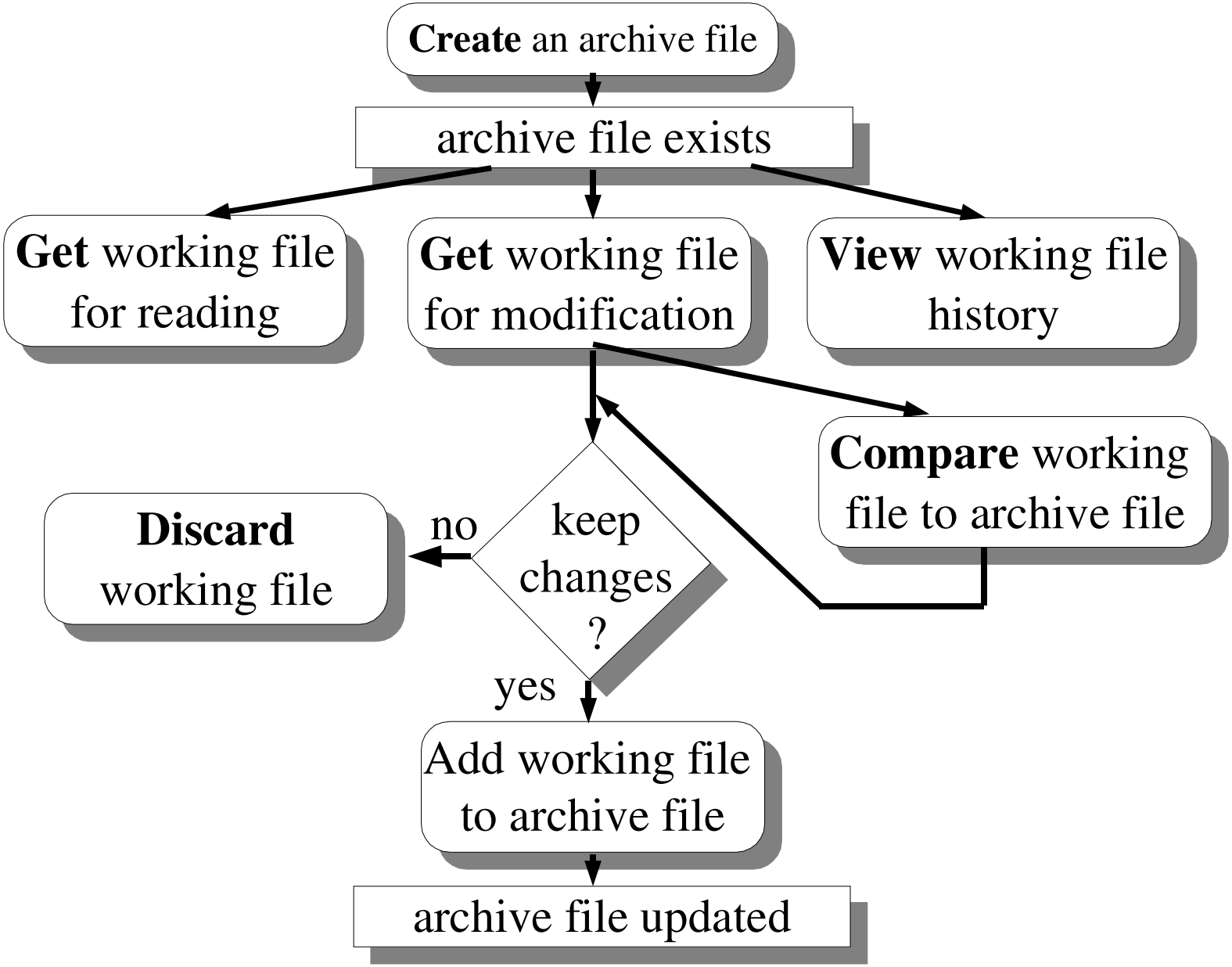
Basics of Source Control

- Steps in source control (cont'd):
  - get a working file for modification
    - if no one else has that revision locked, it will be locked in your name
    - if someone else has that revision locked, you won't be allowed to lock it
    - while lock is set, no one else can modify that revision
  - comparing a working file to its archive file
  - adding a working file to an archive file

Basics of Source Control

- Steps in source control (cont'd):
  - discarding a working file
    - can't just delete working file
    - need also to remove lock
  - viewing history of an archive file

Basic Source Control Operations



SCCS, RCS, CVS

- There are (at least) three major revision control systems
  - SCCS
    - available on most commercial Unices, but not on Linux
  - RCS
    - available on Linux and many BSD-based Unices
    - very popular for personal version control and small projects
  - CVS
    - Built on top of RCS
    - Manages groups of files and projects distributed over a network better than RCS
    - Widely used for Open Source projects

RCS

`ci`	check in RCS revisions
`co`	check out RCS revisions
`rcs`	change RCS file attributes
`rcsdiff`	compare RCS revisions
`rlog`	print log messages

RCS Basics

1. Create an RCS directory in your development directory
2. Place the file under source control

   `ci` *`filename`*
3. Check out a file to make a change

   `co -l` *`filename`*
4. Edit the file as normal (emacs *filename*)
5. Compare working file to its RCS file

   `rcsdiff` *`filename`*
6. Check the file back in, creating a new version

   `ci` *`filename`*

RCS initial check in: ci

```
% ci source.c
enter description, terminated with a
  '.' or end of file
>> Your description here
>> .
initial revision: 1.1
done
$
```

Working file checked in to archive and deleted

RCS check out for modification: co

```
% co -l source.c
RCS/source.c,v --> source.c
revision 1.1
done
$
```

Latest revision checked out and stored in working file

Compare working file to RCS file

```
% rcsdiff source.c
RCS file: source.c,v
retrieving version 1.1
diff -r1.1 source.c
```

`....` (output from diff command)

- Can compare working file to any version:

```
% rcsdiff -r1.3 prog.c
```

- Can compare any version to any other

```
% rcsdiff -r1.2 -r1.3 prog.c
```

Discard working file

```
% rcs -u source.c
RCS file: source.c,v
1.1 unlocked
done
```

Can then remove working file

```
% rm source.c
```

RCS check in: ci

```
% ci source.c
new revison: 1.2; previous
  revision: 1.1
enter log message, terminated with
  a '.' or end of file
>> Your messsage here
>> .
done
$
```

RCS check in with implicit check out: ci -l and ci -u

`% ci -l source.c`

- Checks in source.c, and checks it out again with a lock

`% ci -u source.c`

- Checks in source.c, checks it out again without a lock

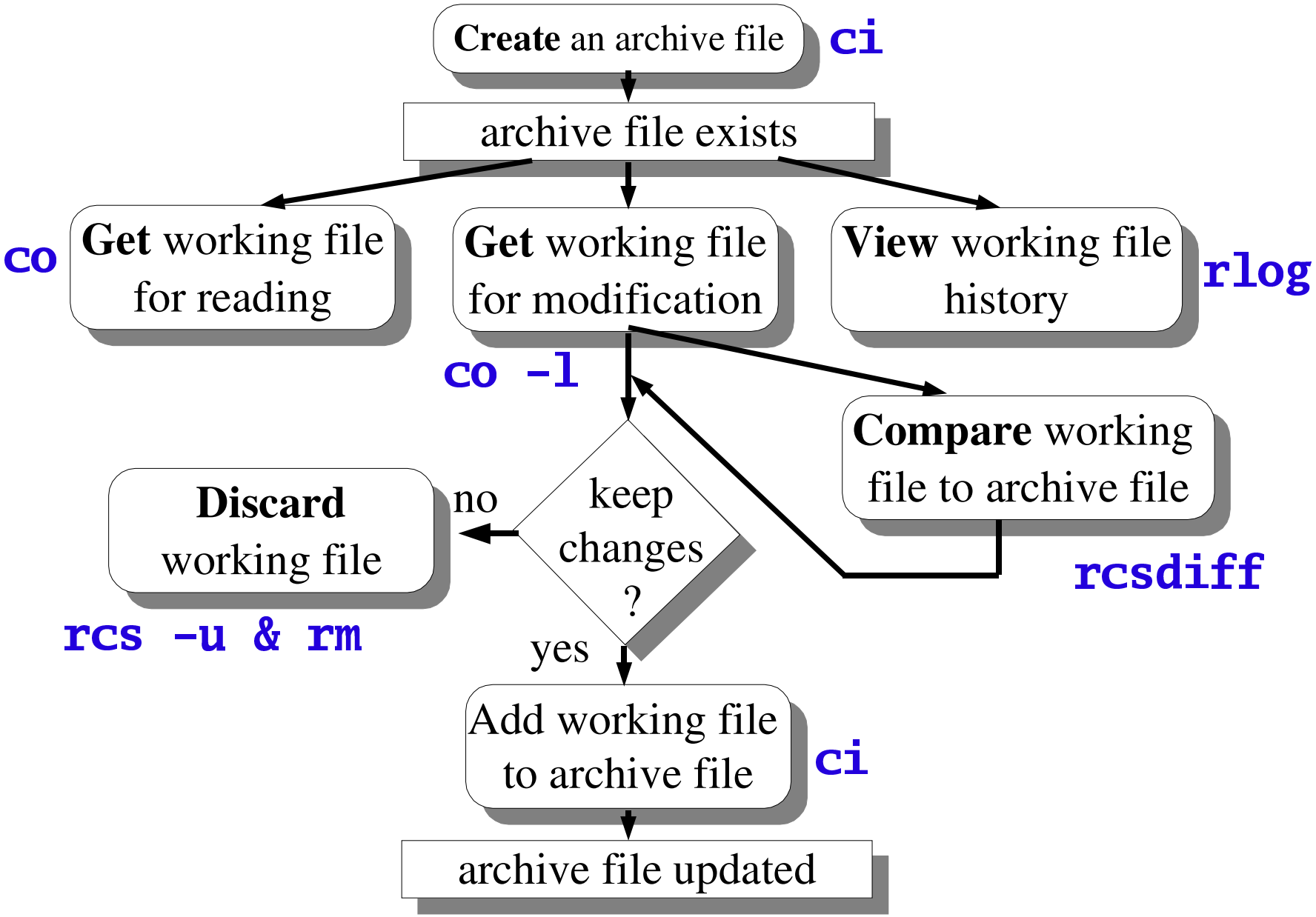
View history of RCS file

```
% rlog source.c

RCS file: source.c,v
working file: source.c
head: 1.2
...
description:
...
revision 1.2
date: ..
your log message

revision 1.1
date: ...
initial revision
```
(default log message)

Basic Source Control Operations



RCS Identifiers

- RCS can put special strings inside your files, to give version information
- These strings automatically get updated each time a new version is created
- Place string markers inside your source files (eg. in comments)
  - Id

    Gives information on version no., date, author,...
  - Log

    Displays entire file history with version description

Version Control within emacs

- Use ctrl-x v v to register, check in/out
- Upon check-in, working file not erased, but changed to read-only.
  - must check out in order to modify
- When checking in changes, enter log message into window, and terminate with crtrl-c ctrl-c