Beyond the Mouse – A Short Course on Programming

11. Backup and Debugging Solving Major (and minor) Crises

Ronni Grapenthin

Geophysical Institute, University of Alaska Fairbanks

November 22, 2010

"The Uncomfortable Truths Well", http://xkcd.com/568 (April 13, 2009)

Today's schedule . . .

1. Backup Strategies
2. Debugging

Today's schedule

1. Backup Strategies
2. Debugging

What is a *backup*?

Backup, backup!

- Creating a copy of something that must never get lost.
- data, results, settings, figures, writing (YOUR THESIS), . . .
- . . . because hard drives sometimes die, laptops get lost, houses burn down, etc.

What is a *backup*?

Backup, backup!

- Creating a copy of something that must never get lost.
- data, results, settings, figures, writing (YOUR THESIS), . . .
- . . . because hard drives sometimes die, laptops get lost, houses burn down, etc.

General strategies

- Episodically create a physical copy on a medium different from primary hard drive (e.g. usb drive).

What is a *backup*?

Backup, backup!

- Creating a copy of something that must never get lost.
- data, results, settings, figures, writing (YOUR THESIS), . . .
- . . . because hard drives sometimes die, laptops get lost, houses burn down, etc.

General strategies

- Episodically create a physical copy on a medium different from primary hard drive (e.g. usb drive).
- Use one of the gazillion tools that help you with this.

What is a *backup*?

Backup, backup!

- Creating a copy of something that must never get lost.
- data, results, settings, figures, writing (YOUR THESIS), . . .
- . . . because hard drives sometimes die, laptops get lost, houses burn down, etc.

General strategies

- Episodically create a physical copy on a medium different from primary hard drive (e.g. usb drive).
- Use one of the gazillion tools that help you with this.
- We'll concentrate on `rsync`

What is a *backup*?

Backup, backup!

- Creating a copy of something that must never get lost.
- data, results, settings, figures, writing (YOUR THESIS), . . .
- . . . because hard drives sometimes die, laptops get lost, houses burn down, etc.

General strategies

- Episodically create a physical copy on a medium different from primary hard drive (e.g. usb drive).
- Use one of the gazillion tools that help you with this.
- We'll concentrate on `rsync`
- Whatever method you choose, make sure the files can indeed be recovered (i.e. test the backup)

rsync: a fast, versatile, remote (and local) file-copying tool

Command line syntax (see man page!)

```
Local:  rsync [OPTION...] SRC... [DEST]

Access via remote shell:
  Pull: rsync [OPTION...] [USER@]HOST:SRC... [DEST]
  Push: rsync [OPTION...] SRC... [USER@]HOST:DEST

Access via rsync daemon:
  Pull: rsync [OPTION...] [USER@]HOST::SRC... [DEST]
        rsync [OPTION...] rsync://[USER@]HOST[:PORT]/SRC... [DEST]
  Push: rsync [OPTION...] SRC... [USER@]HOST::DEST
        rsync [OPTION...] SRC... rsync://[USER@]HOST[:PORT]/DEST

Usages with just one SRC arg and no DEST arg will list the source
files instead of copying.
```

rsync: a fast, versatile, remote (and local) file-copying tool

Command line syntax (see man page!)

```
Local:  rsync [OPTION...] SRC... [DEST]

Access via remote shell:
  Pull: rsync [OPTION...] [USER@]HOST:SRC... [DEST]
  Push: rsync [OPTION...] SRC... [USER@]HOST:DEST

Access via rsync daemon:
  Pull: rsync [OPTION...] [USER@]HOST::SRC... [DEST]
        rsync [OPTION...] rsync://[USER@]HOST[:PORT]/SRC... [DEST]
  Push: rsync [OPTION...] SRC... [USER@]HOST::DEST
        rsync [OPTION...] SRC... rsync://[USER@]HOST[:PORT]/DEST

Usages with just one SRC arg and no DEST arg will list the source
files instead of copying.
```

- If any of the files to copy already exist, rsync sends only the differences.

`rsync`: a fast, versatile, remote (and local) file-copying tool

Command line syntax (see man page!)

```
Local:  rsync [OPTION...] SRC... [DEST]

Access via remote shell:
  Pull: rsync [OPTION...] [USER@]HOST:SRC... [DEST]
  Push: rsync [OPTION...] SRC... [USER@]HOST:DEST

Access via rsync daemon:
  Pull: rsync [OPTION...] [USER@]HOST::SRC... [DEST]
        rsync [OPTION...] rsync://[USER@]HOST[:PORT]/SRC... [DEST]
  Push: rsync [OPTION...] SRC... [USER@]HOST::DEST
        rsync [OPTION...] SRC... rsync://[USER@]HOST[:PORT]/DEST

Usages with just one SRC arg and no DEST arg will list the source
files instead of copying.
```

- If any of the files to copy already exist, `rsync` sends only the differences.
- `-avz` transfer in “archive” mode: ensures that symbolic links, permissions, etc. are preserved. Compression is used to reduce the size of data portions.

rsync: example – my backup solution

```
#!/bin/csh
# archives list of folders to /media/backup

#backup destination, external HDD
set BACKUP = /media/backup

#if disk doesn't exist, we've got to mount it
if !(-e $BACKUP/eolan ) then
   echo "No backup disk! Trying to mount  external disk to /media/backup."
   mntbackup
endif

#check whether my remote folders are mounted
if !(-e ~/tintina/projects) then
   echo "LAB not mounted ... do that now!"
   mnttintina
endif

#DO IT!
echo "Starting Backup ..."
rsync -avz ~ $BACKUP/eolan/roon

#my pictures live on a different HDD, check whether
#backup possible and do it.
if (-e /media/disk/photos) then
   rsync -avz /media/disk/photos $BACKUP/backup_disk_160
endif
```

Listing ~/bin/backup

Today's schedule

1. Backup Strategies
2. Debugging

Review: Software Development Cycle

1. Design
2. Coding
3. Test
4. **Debugging**
5. go back to 1,2, or 3, ...

What is “debugging”?

Debugging is the **art** of finding and fixing mistakes in computer programs. To be successful you need insight, creativity, logic, and determination.

What is “debugging”?

Debugging is the **art** of finding and fixing mistakes in computer programs. To be successful you need insight, creativity, logic, and determination.

> *Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it.*
>
> Brian Kernighan

Truths about bugs and debugging . . .

- Bugs are static – they won't run away.
- Often, the problem is **simple**.
- You created the bug! It's nobody else's fault - suck it up!
- Debugging is a great way to learn being self-critical. Good luck!
- Be critical – did you mean '<', '<=', '>', '>='?
- Don't panic – be systematic!
- Sleep, go for a walk, come back later.

Debugging Styles

- **echoing**: place print statements at useful points in a program (function entry, exit)
- **unit testing**: write calls to particular function, throw artificial values at it
- **exception handling**: in high level languages: sources of mistakes easier to spot
- **online debuggers**: for our purposes not necessary, useful if you want to step through your code, or for memory problems
- **version control**: have a tool keep track of changes you make; roll back to bug-free code is simple

Debugging Styles: echoing

. . . we find stepping through a program less productive than thinking harder and **adding output statements and self-checking code at critical places**. *Clicking over statements takes longer than scanning the output of judiciously-placed displays. It takes less time to decide where to put print statements than to single-step to the critical section of code, even assuming we know where that is. More important,* **debugging statements stay with the program; debugging sessions are transient**.

From: Brian Kernighan, Rob Pike "The Practice of Programming"

Debugging Styles: echoing

> *. . . we find stepping through a program less productive than thinking harder and* **adding output statements and self-checking code at critical places**. *Clicking over statements takes longer than scanning the output of judiciously-placed displays. It takes less time to decide where to put print statements than to single-step to the critical section of code, even assuming we know where that is. More important,* **debugging statements stay with the program; debugging sessions are transient**.
>
> *From: Brian Kernighan, Rob Pike "The Practice of Programming"*

- write method that displays text only if a global DEBUG flag is set
- find ways to implement such external switches – for SHELL: environment vars, Matlab: create your own preferences
- call this method whenever necessary: entry, exit of functions, to display certain values, to follow the program flow, . . .

Debugging Styles: echoing

. . . see `t_debug` demo . . .

Debugging Styles: unit testing

- at the simplest: write calls to your functions with artificial values
- execute these calls at the beginning of your code, check function results
- this helps to detect errors due to changes in functions immediately
- also: assertion that function works for tested TYPES
- can be done for any language (some languages come with fancy frameworks)

Debugging Styles: exception handling

Full exception handling support in Matlab:

Matlab – `try-catch`

```
% try, STATEMENT, catch ME, STATEMENT, end.
%
% EXAMPLE: file opening
clc;
try
    fid  = fopen('whatever.txt', 'r'); % open a non-existing file
    data = fread(fid);                 % now try to get its data
catch myException                      % any name for error message object
    %let the user know, implement graceful program termination ...
    disp(myException);                 % display full error object
    disp(myException.message);         % actual message is more accessible
    disp(myException.stack);           % where did things occur?
end

disp('------> We do get here!')

%now without try-catch ...
fid  = fopen('whatever.txt', 'r');
data = fread(fid);

disp('We cannot get here!')
```

Debugging Styles: version control

Full exception handling support in Matlab:

Matlab – `try-catch`

```
% try, STATEMENT, catch ME, STATEMENT, end.
%
% EXAMPLE: file opening
clc;
try
    fid  = fopen('whatever.txt', 'r'); % open a non-existing file
    data = fread(fid);                 % now try to get its data
catch myException                      % any name for error message object
    %let the user know, implement graceful program termination ...
    disp(myException);                 % display full error object
    disp(myException.message);         % actual message is more accessible
    disp(myException.stack);           % where did things occur?
end

disp('------> We do get here!')

%now without try-catch ...
fid  = fopen('whatever.txt', 'r');
data = fread(fid);

disp('We cannot get here!')
```

Version control (with subversion)

What is ‘version control’?

“Version control is the art of managing changes to information.” (svnbook)

- a fileserver that remembers every change ever written to it.
- traditionally used by programmers: change little bits of code on one day only to undo it the next day.
- well, that’s just what we do with papers, theses, . . .

Version control (with subversion)

What is ‘version control’?

“Version control is the art of managing changes to information.” (svnbook)

- a fileserver that remembers every change ever written to it.
- traditionally used by programmers: change little bits of code on one day only to undo it the next day.
- well, that’s just what we do with papers, theses, . . .

What is ‘version control’ NOT?

- NOT a backup: creates value (history, log entries, . . .)
- Backup your repository every now and then.

Version control (with subversion)

What is ‘version control’?

“Version control is the art of managing changes to information.” (svnbook)

- a fileserver that remembers every change ever written to it.
- traditionally used by programmers: change little bits of code on one day only to undo it the next day.
- well, that’s just what we do with papers, theses, . . .

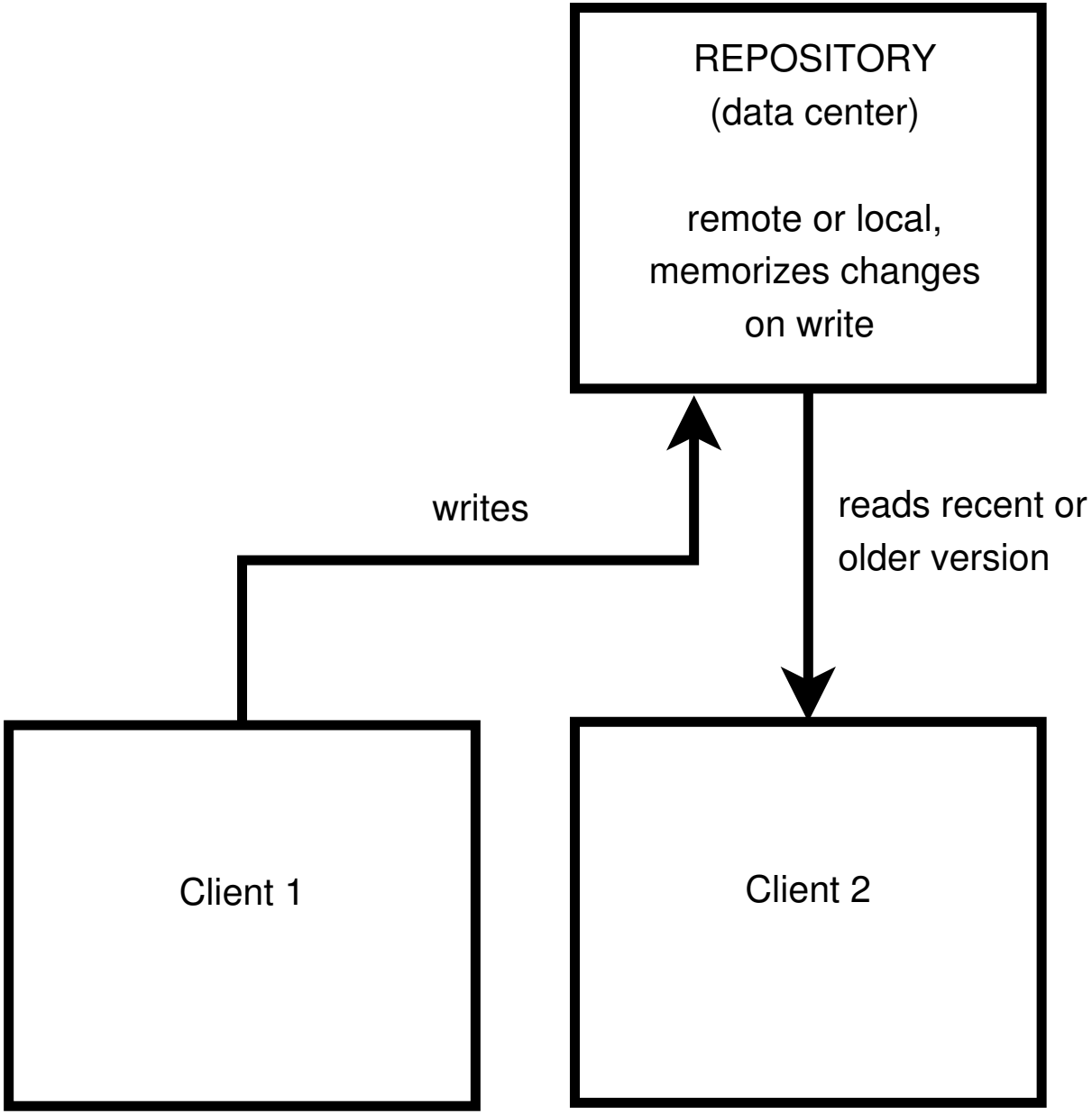
What is ‘version control’ NOT?

- NOT a backup: creates value (history, log entries, . . .)
- Backup your repository every now and then.

What can be under version control?

Depends on tool: CVS – only text files, subversion – text and binary files

How it works



$> svnadmin

$> svn

Creating/managing a repository: `svnadmin, svn`

`svnadmin` Command line syntax

usage: `svnadmin SUBCOMMAND REPO_PATH [ARGS & OPTIONS ...]`

Type '`svnadmin help <subcommand>`' for help on a specific subcommand.

subcommands: many! Type '`svnadmin help`' to see them

Creating/managing a repository: `svnadmin`, `svn`

`svnadmin` Command line syntax

usage: `svnadmin SUBCOMMAND REPO_PATH [ARGS & OPTIONS ...]`
Type '`svnadmin help <subcommand>`' for help on a specific subcommand.
subcommands: many! Type '`svnadmin help`' to see them

`svn` Command line syntax

usage: `svn <subcommand> [options] [args]`
Type '`svn help <subcommand>`' for help on a specific subcommand.
subcommands: even more! Type '`svn help`' to see them

Creating/managing a repository: `svnadmin, svn`

Repository creation (in your current directory)

```
$> svnadmin create --fs-type fsfs $PWD/repos
```

Creating/managing a repository: `svnadmin, svn`

Repository creation (in your current directory)

```
$> svnadmin create --fs-type fsfs $PWD/repos
```

Preparing your project (repository layout):

```
$> mkdir my_project
$> cd my_project
$> mkdir trunk branches tags
$> mv <project-files> trunk
```

Creating/managing a repository: `svnadmin, svn`

Repository creation (in your current directory)

```
$> svnadmin create --fs-type fsfs $PWD/repos
```

Preparing your project (repository layout):

```
$> mkdir my_project
$> cd my_project
$> mkdir trunk branches tags
$> mv <project-files> trunk
```

Putting your stuff under version control

```
$> svn import my_project \
file:///$PWD/repos/my_project
```

Creating/managing a repository: `svnadmin, svn`

Your work is now in the repository, get your local copy!

```
$> mv my_project my_project_old
$> svn checkout file:///$PWD/repos/my_project/trunk
\
my_project
```

Creating/managing a repository: `svnadmin, svn`

Your work is now in the repository, get your local copy!

```
$> mv my_project my_project_old
$> svn checkout file:///$PWD/repos/my_project/trunk
\
my_project
```

Work cycle

```
$> svn update
```

edit files locally

```
$> svn commit
```

Creating/managing a repository: `svnadmin, svn`

Log of a session (local repository):

```
eolan:~/../07_unix_tools2> svnadmin create --fs-type fsfs $PWD/repos
eolan:~/../07_unix_tools2> ls repos
conf  db  format  hooks  locks  README.txt
eolan:~/../07_unix_tools2> mkdir BTM
eolan:~/../07_unix_tools2> mkdir BTM/trunk BTM/tags BTM/branches
eolan:~/../07_unix_tools2> cp ../../beyond_the_mouse/* ./BTM/trunk/
eolan:~/../07_unix_tools2> ls BTM/trunk/
01_thinking_programs.aux   02_fundamentals.pdf        ...
eolan:~/../07_unix_tools2> svn import BTM file:///$PWD/repos/BTM -m "initial import"
Adding         BTM/trunk
...
Committed revision 1.
eolan:~/../07_unix_tools2> mv BTM BTM_old
eolan:~/../07_unix_tools2> svn checkout file:///$PWD/repos/BTM/trunk BTM
A    BTM/04_fundamentals.snm
...
Checked out revision 3.
```

Creating/managing a repository: `svnadmin, svn`

Log of a session (local repository):

```
eolan:~/../07_unix_tools2> svnadmin create --fs-type fsfs $PWD/repos
eolan:~/../07_unix_tools2> ls repos
conf  db  format  hooks  locks  README.txt
eolan:~/../07_unix_tools2> mkdir BTM
eolan:~/../07_unix_tools2> mkdir BTM/trunk BTM/tags BTM/branches
eolan:~/../07_unix_tools2> cp ../../beyond_the_mouse/* ./BTM/trunk/
eolan:~/../07_unix_tools2> ls BTM/trunk/
01_thinking_programs.aux   02_fundamentals.pdf        ...
eolan:~/../07_unix_tools2> svn import BTM file:///$PWD/repos/BTM -m "initial import"
Adding         BTM/trunk
...
Committed revision 1.
eolan:~/../07_unix_tools2> mv BTM BTM_old
eolan:~/../07_unix_tools2> svn checkout file:///$PWD/repos/BTM/trunk BTM
A    BTM/04_fundamentals.snm
...
Checked out revision 3.
```

- remote repositoy: `ssh` into server, use `svnadmin` as shown above
- `svn import my_project svn+ssh://user@server/repos/my_project`
- `svn checkout svn+ssh://user@server/repos/my_project/trunk my_project`