# Beyond the Mouse – A Short Course on Programming

## 7. Unix Tools II

Ronni Grapenthin

Geophysical Institute, University of Alaska Fairbanks

October 22, 2009


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

# Today's schedule . . .

1. Introduction
2. Remote access: ssh
3. Backup Strategies
4. Makefiles
5. Version control (with subversion)
6. Putting it all together . . .

# Today's schedule

# Introduction

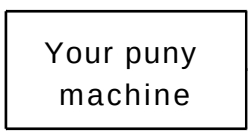
## Goal for today:

- go over several book shelves
- introduce a couple tools: ssh, rsync, make, svn
- explain how they work by themselves
- show how you can orchestrate them into a decent project management suite

# Today's schedule

# Remote access: ssh

`ssh` (secure shell): log into and execute commands on remote machine

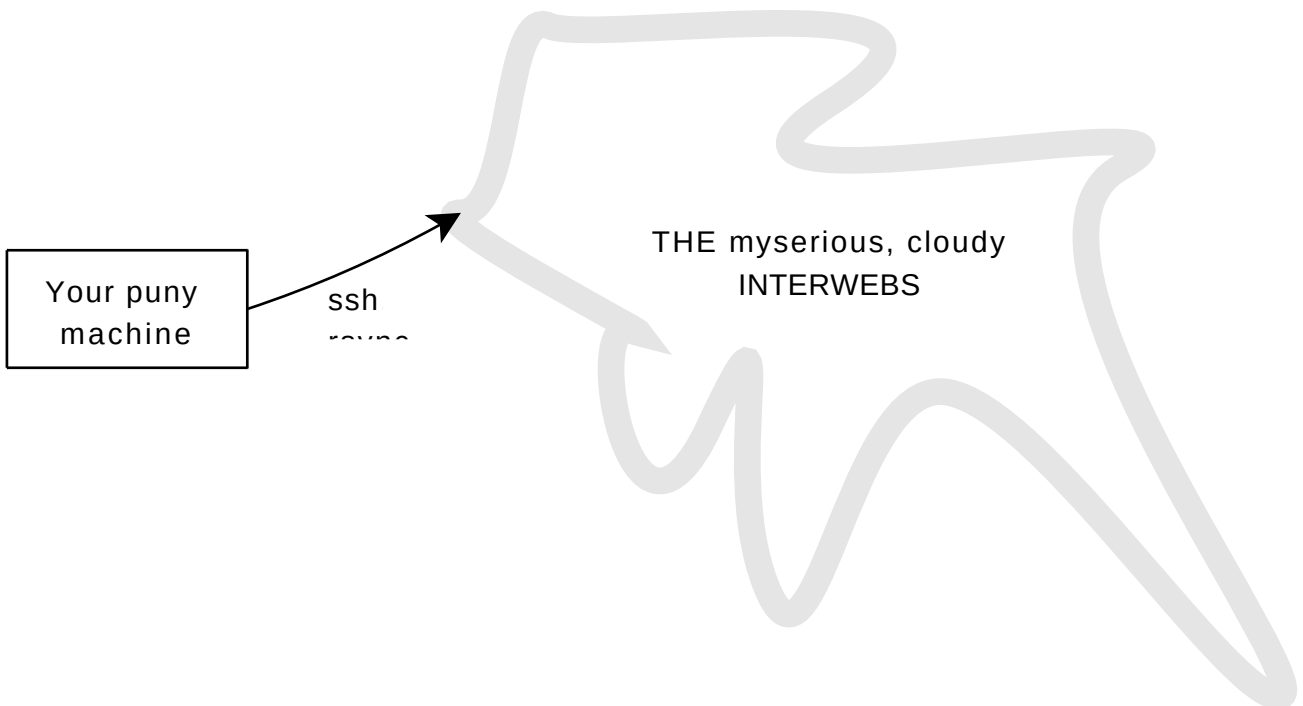


# Remote access: ssh

`ssh` (secure shell): log into and execute commands on remote machine

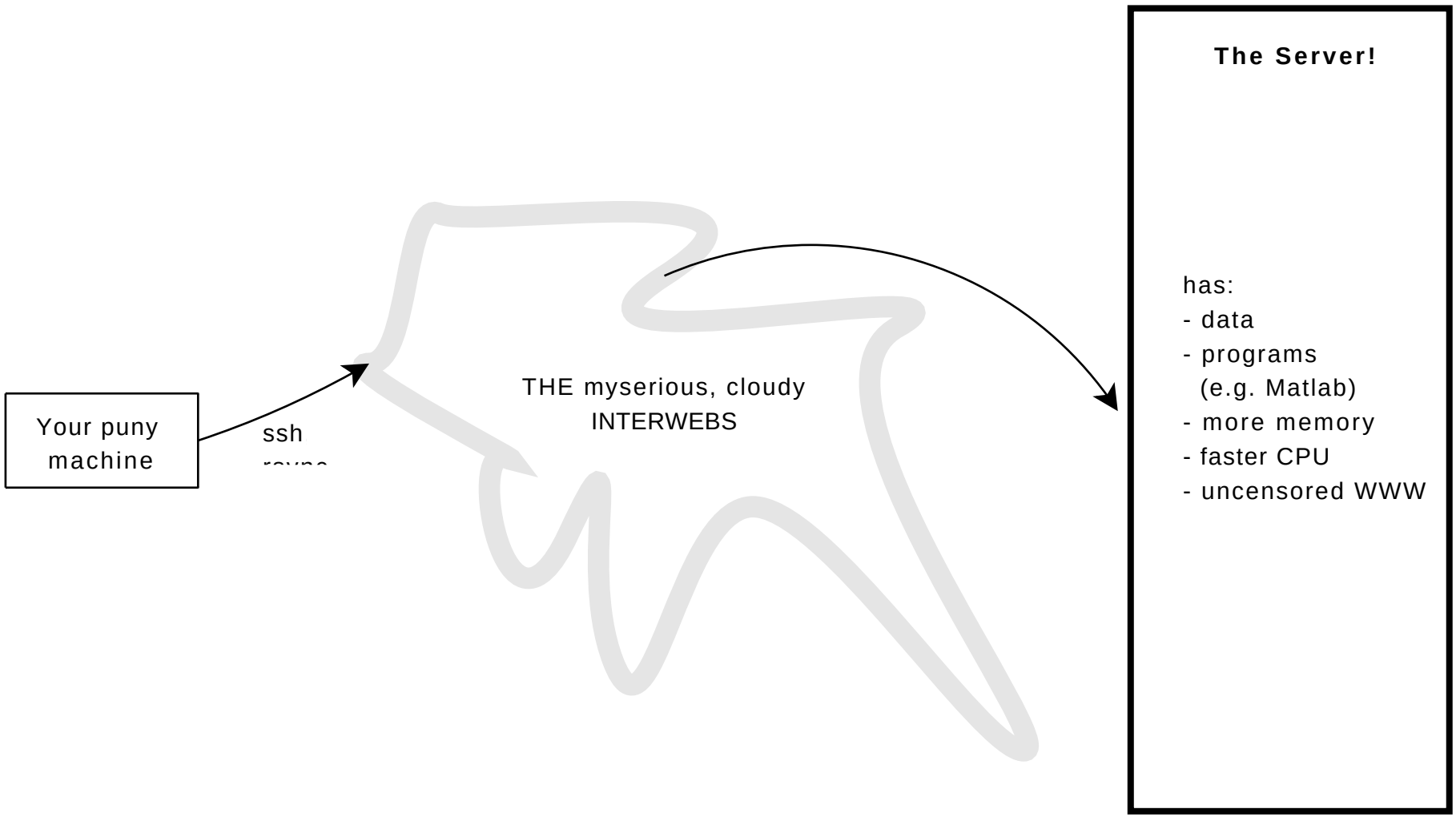
# Remote access: ssh

`ssh` (secure shell): log into and execute commands on remote machine



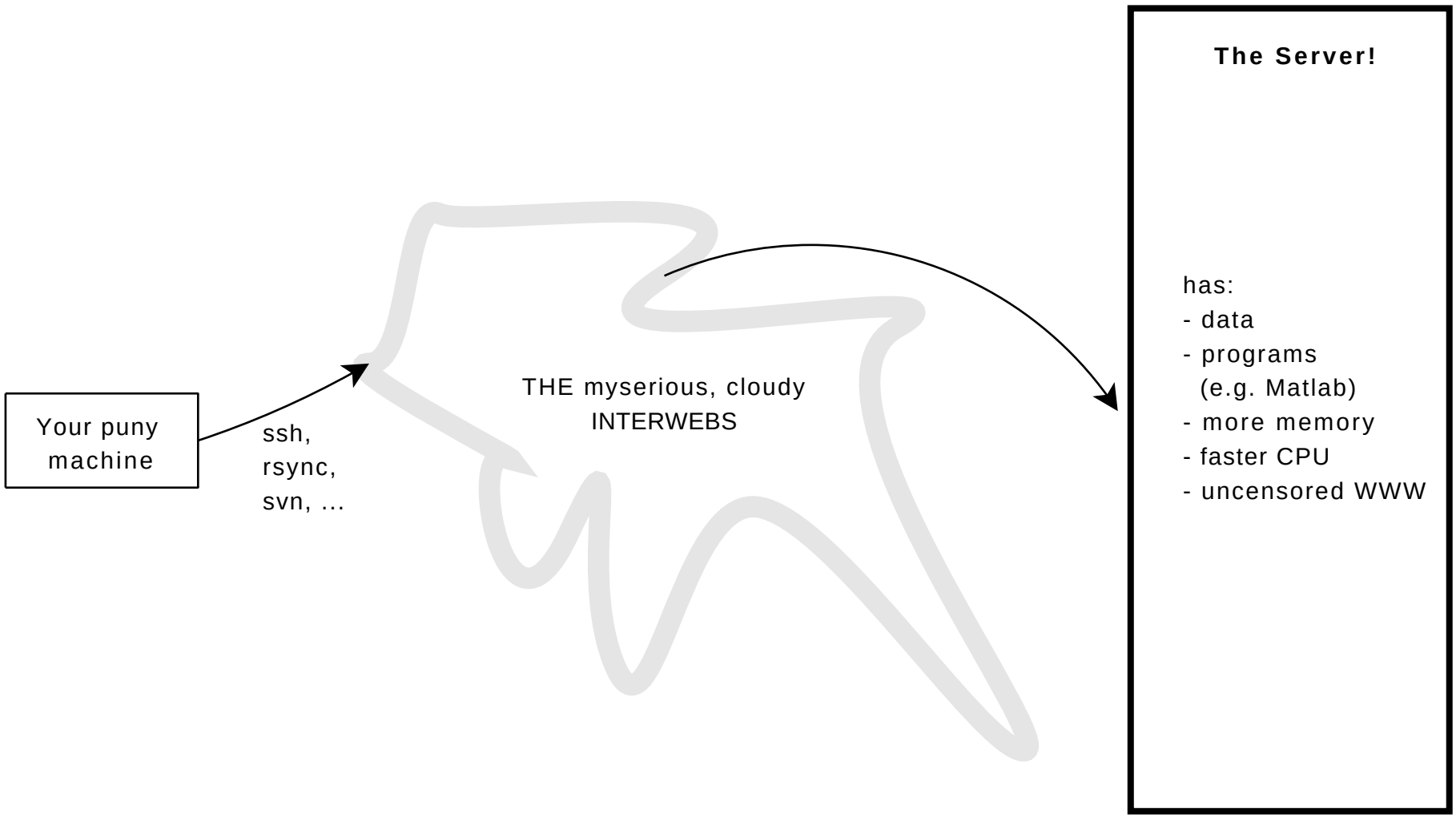
# Remote access: ssh

`ssh` (secure shell): log into and execute commands on remote machine



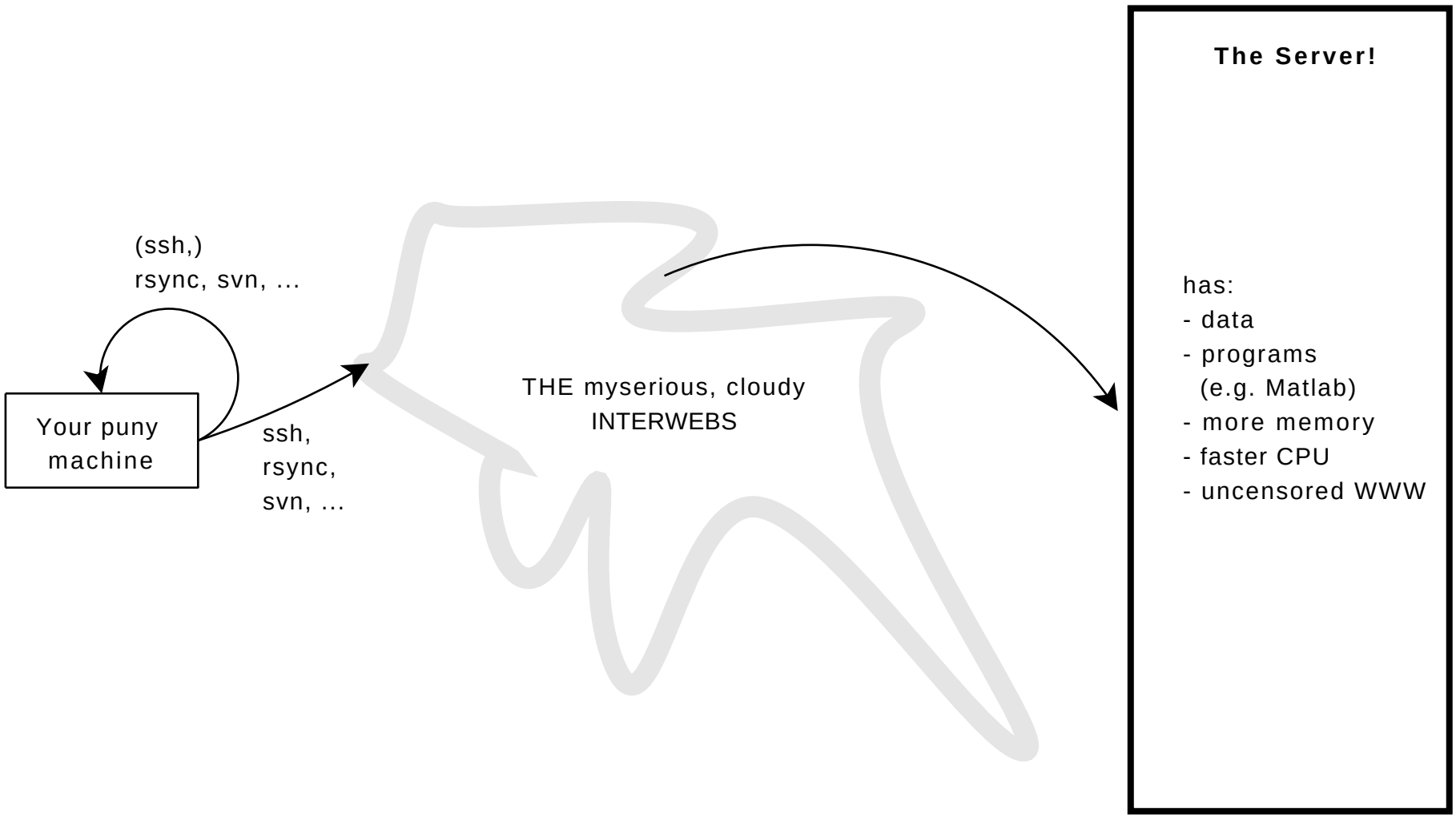
# Remote access: ssh

`ssh` (secure shell): log into and execute commands on remote machine



# Remote access: ssh

`ssh` (secure shell): log into and execute commands on remote machine



# Remote access: ssh

## Command line syntax (see man page!)

```
ssh [A LOT OF OPTIONS] [user@]hostname [command]
```

# Remote access: ssh

## Command line syntax (see man page!)

```
ssh [A LOT OF OPTIONS] [user@]hostname [command]
```

## Example – Logging into GPS webserver

```
> ssh -2Y ronni@fairweather.gps.alaska.edu
```

Opens a new session on host `fairweather.gps.alaska.edu` for user `ronni` using protocol SSH 2 with trusted X11 forwarding.

# Remote access: ssh

## Command line syntax (see man page!)

```
ssh [A LOT OF OPTIONS] [user@]hostname [command]
```

## Example – Logging into GPS webserver

```
> ssh -2Y ronni@fairweather.gps.alaska.edu
```

Opens a new session on host `fairweather.gps.alaska.edu` for user `ronni` using protocol SSH 2 with trusted X11 forwarding.

## Why/when would you need that?

- Whenever you don't want to walk to the machine.
- Can't access data locally.
- You are actually, physically, and really on that machine (isn't the Internet great?)
- Many tools (svn, rsync, ...) offer to use ssh tunnels (they do their job after an SSH session has been established).

# Today's schedule

# 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, fires burn down houses, you get the idea.

# 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, fires burn down houses, you get the idea.

## General strategies

- Episodically create a physical copy on a medium different from your hard drive (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, fires burn down houses, you get the idea.

## General strategies

- Episodically create a physical copy on a medium different from your hard drive (usb drive).
- OR 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, fires burn down houses, you get the idea.

## General strategies

- Episodically create a physical copy on a medium different from your hard drive (usb drive).
- OR 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, fires burn down houses, you get the idea.

## General strategies

- Episodically create a physical copy on a medium different from your hard drive (usb drive).
- OR use one of the gazillion tools that help you with this.
- We'll concentrate on `rsync`
- Whatever method you choose, every now and then make sure the files can indeed be recovered!

# 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 already exist on the remote system then 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 already exist on the remote system then `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

```
#!/bin/csh
# takes folder in ~/www that's to be updated on fairweather as
# argument

if ($#argv < 1) then
   echo "Usage: $0 <folder in ~/www>"
   exit
endif

rsync -avz --delete ~/www/$1 ronni@fairweather.gps.alaska.edu:/export/ftpweb/htdocs
```

# rsync: example

```
#!/bin/csh
# takes folder in ~/www that's to be updated on fairweather as
# argument

if ($#argv < 1) then
   echo "Usage: $0 <folder in ~/www>"
   exit
endif

rsync -avz --delete ~/www/$1 ronni@fairweather.gps.alaska.edu:/export/ftpweb/htdocs
```

```
#!/bin/csh

#pulling selected data for a project from a server
rsync -avz --include="*/" --include="BEZ*" --include="BZ*" --exclude="*" \
   ronni@fairweather.gps.alaska.edu:/gps/data/NEAsia2.5_timeseries/ ./data
```

# Today's schedule

# `make` my day!

- `make` is a program that (usually) lives in `/usr/bin`
- determines which parts of a project need to be updated depending on those that changed
- `make` does that according to rules defined in a `Makefile`
- has its roots in the programming world, but can be used for anything (link coffee machine to USB port, write rules, `make coffee`)

# `make` my day!

## Make Rules

```
target ...  :  prerequisites ...
<TAB> command 1
<TAB> command 2
...
<TAB> command N
```

# `make` my day!

## Make Rules

```
target ...  :  prerequisites ...
<TAB> command 1
<TAB> command 2
...
<TAB> command N
```

- `target`: name of a file to be created or an action to be carried out (e.g. `update`)
- `prerequisite`: a file/target necessary to create the target, often there are many prerequisite**s**, is optional.
- `command`: action that `make` carries out. Tabulator, tab, <TAB>, whatever this one character MUST be at the beginning of each command line

## Make Example

```
# simple Makefile that shows which files in
# directory tree have changed since they've
# last been displayed
# call: make -f Makefile-delta <rule>

# 1st rule: target 'changes' depends on all the files
# that contain a dot in the current directory
#
# 1. command: display all the prerequisites that changed since
#    last display, internal variable $? contains this list,
#    display each on separate line using BASH-shell for-loop
# 2. command: touch (i.e. update) empty file 'changes', so that make
#    knows about the last time this rule has been carried
#    out
# the '@' says that the command should not be echoed in the shell

changes: *.*
    @for i in $?; do echo $$i; done
    @touch changes


# 2nd rule: remove file 'changes', Implicit understanding of this rule:
# Reset everything to the state before make was executed the first time
# no '@' - see the difference
clean:
    rm changes
```

# `make` my day even better!

## Make can do variables (and a lot more), too:

```
# simple Makefile demonstrating the use of variables
# call: make -f Makefile-vars <rule>

# Defining a variable
FILELIST := $(shell find ./ -type f)


# Accessing a variable ... as a list and then entry by entry
all:
    @echo
    @echo files:
    @echo $(FILELIST)
    @echo
    @echo files:
    @for i in $(FILELIST); do echo $$i; done
```

# Today's schedule

# 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, . . .

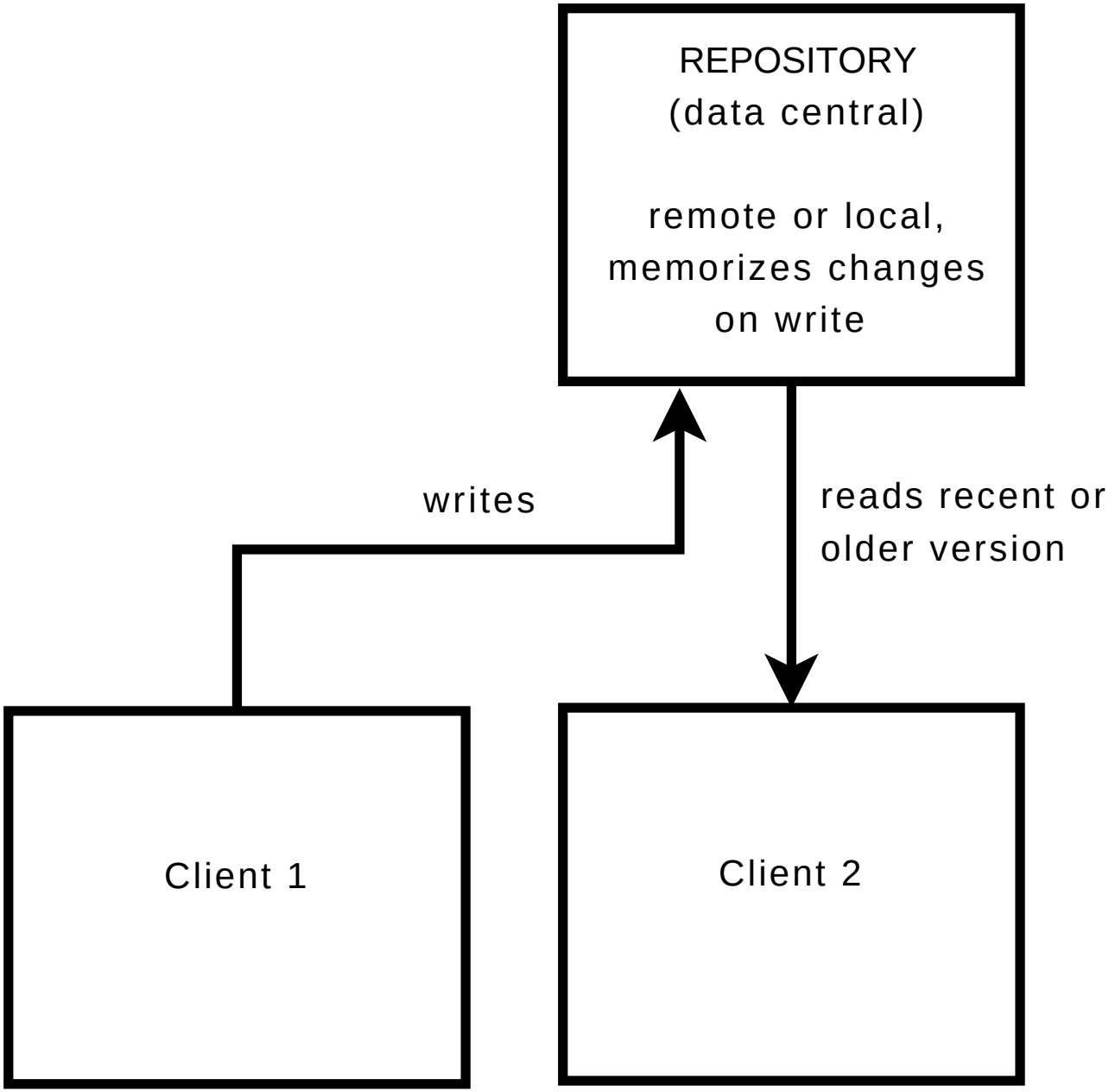
## What is ‘version control’ NOT?

- 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

```
general usage:  svnadmin SUBCOMMAND REPOS_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

```
general usage:  svnadmin SUBCOMMAND REPOS_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`

# Today's schedule

1. Introduction
2. Remote access: ssh
3. Backup Strategies
4. Makefiles
5. Version control (with subversion)
6. Putting it all together . . .

# Putting it all together . . . (assuming we're the only ones working on a project)

- create directory for all your projects: `mkdir /projects`
- create a new project: `new_project.csh <project-name>`
- coming into the office: `make start-day`
- do your work: `make all`, every now and then
- finish your day: `make end-day`