twitter: @akrabat

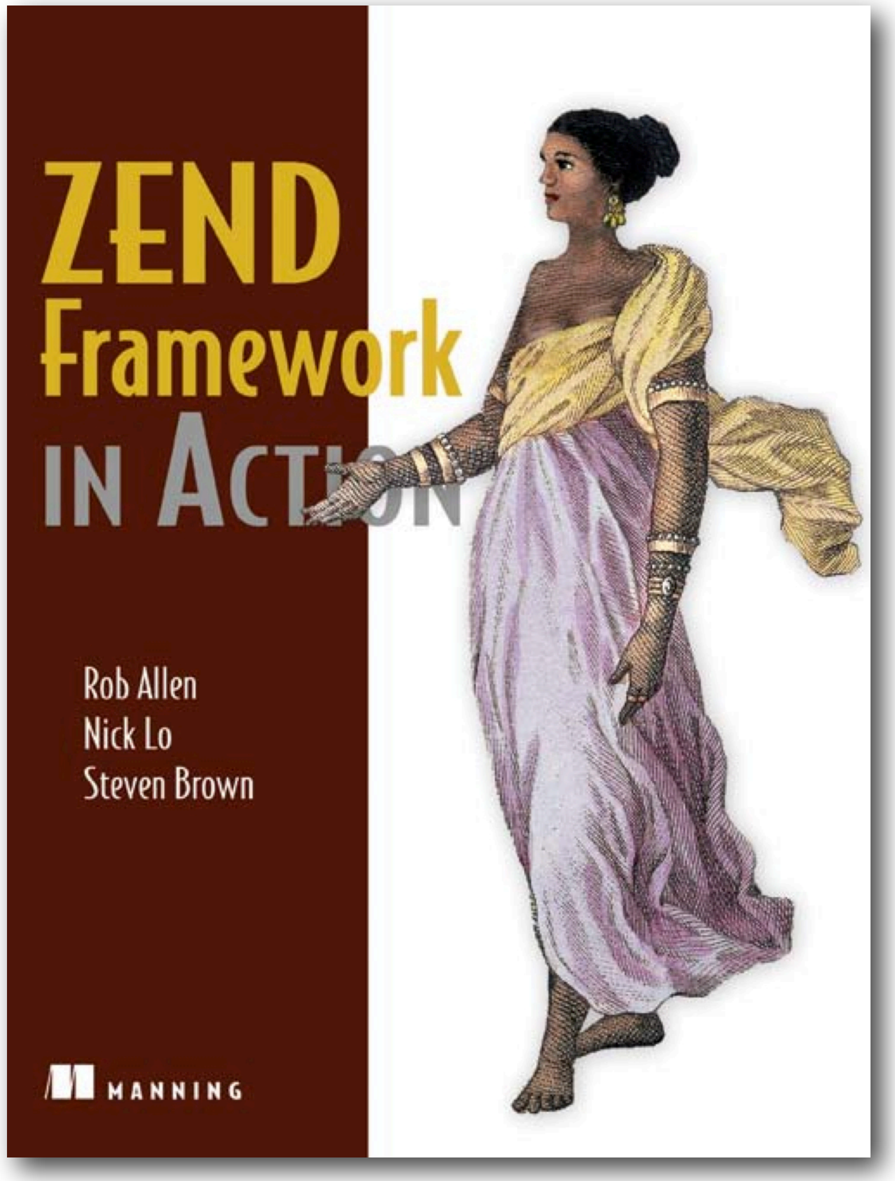
Stress-free Deployment

Rob Allen

WooWeb, March 2013

Rob Allen?

- PHP developer since 1999
- ZF1 and ZF2 team member
- ZF tutorial at akrabat.com
- Zend Framework in Action!



Why automate deployment?

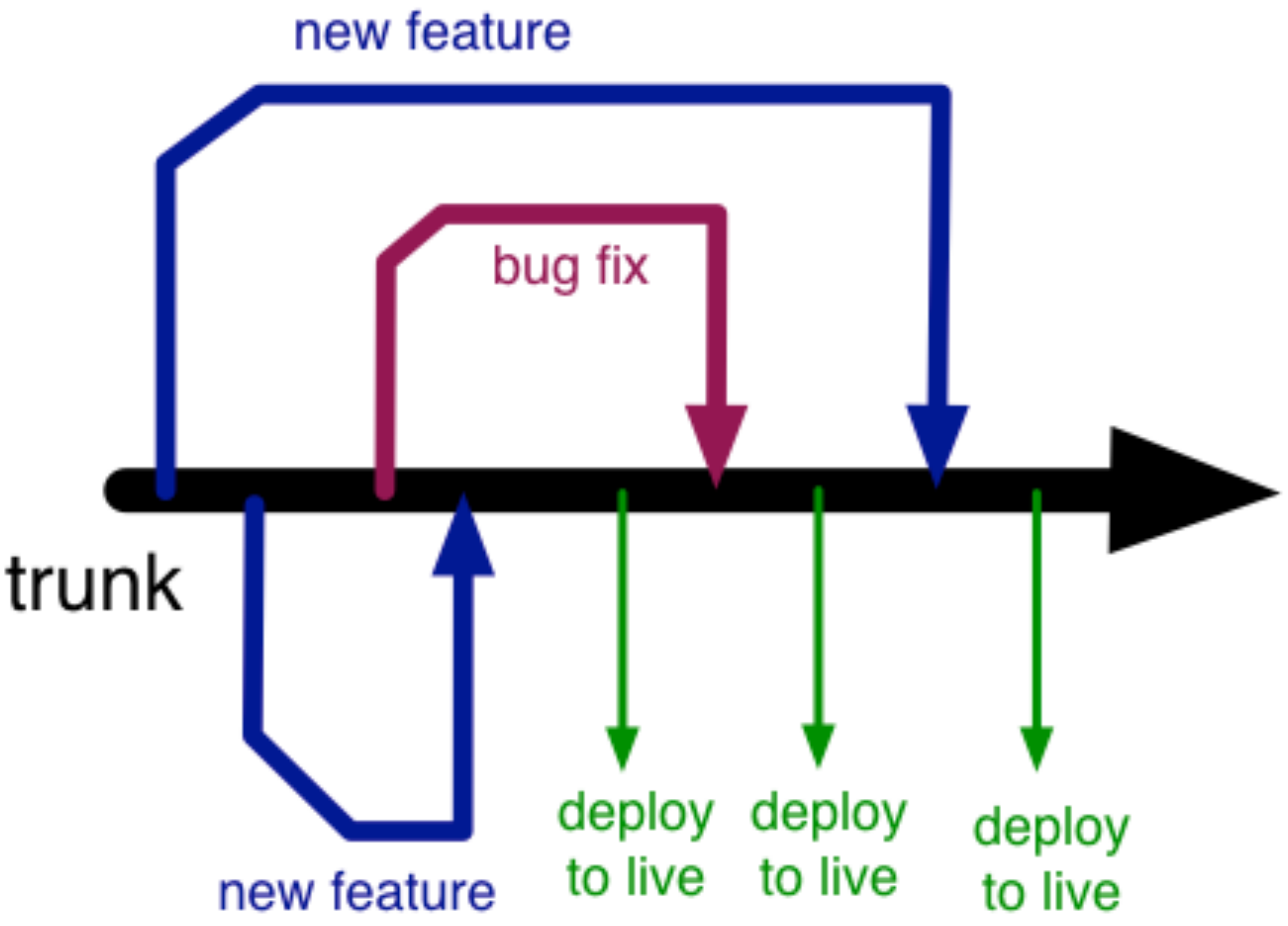
Getting your house in order

Source code control

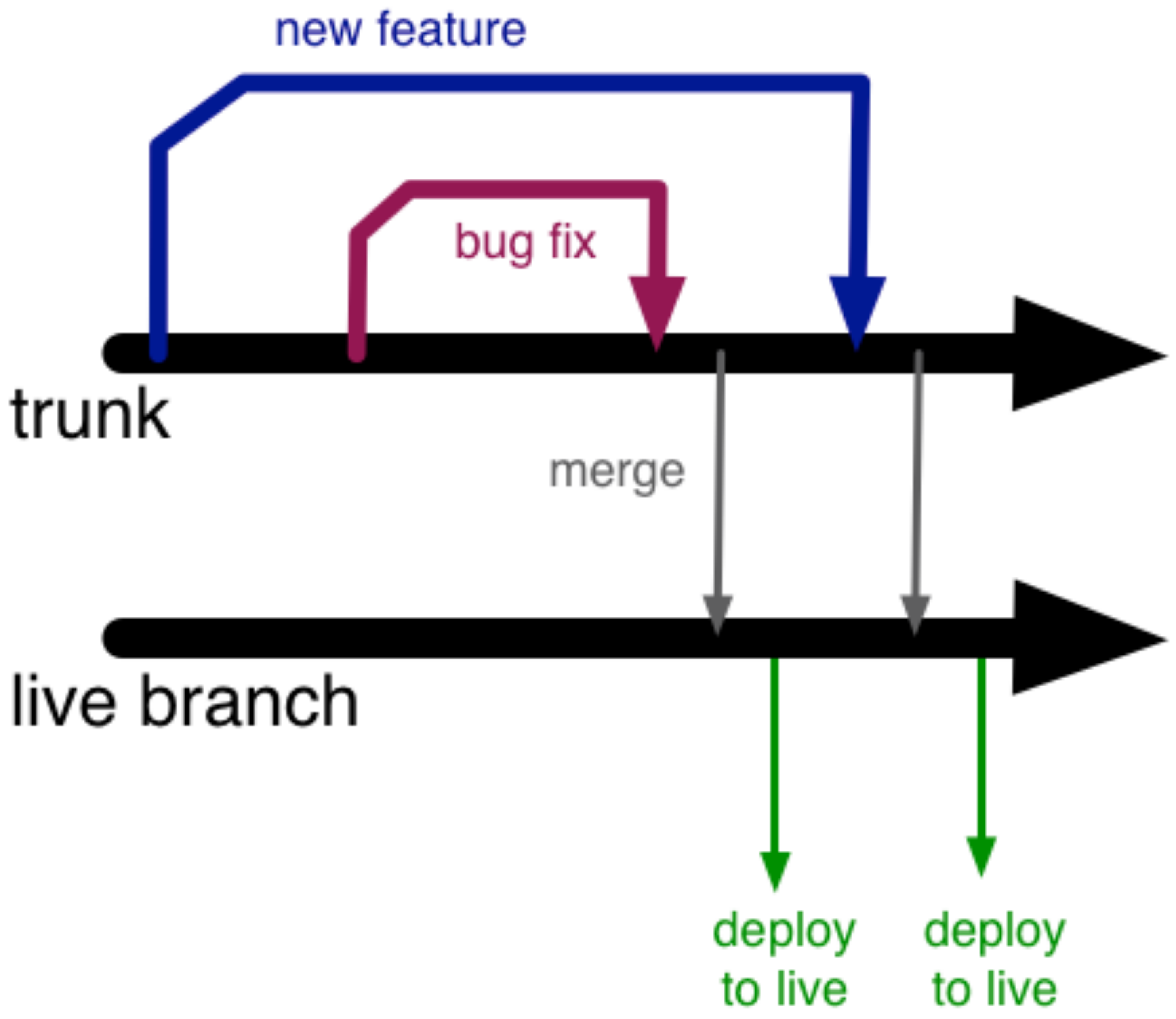
Branch!

- Branch every new feature
  - (that includes bug fixes)
- Be ready go live at all times
  - Trunk deployment
  - Live branch deployment
  - Git flow

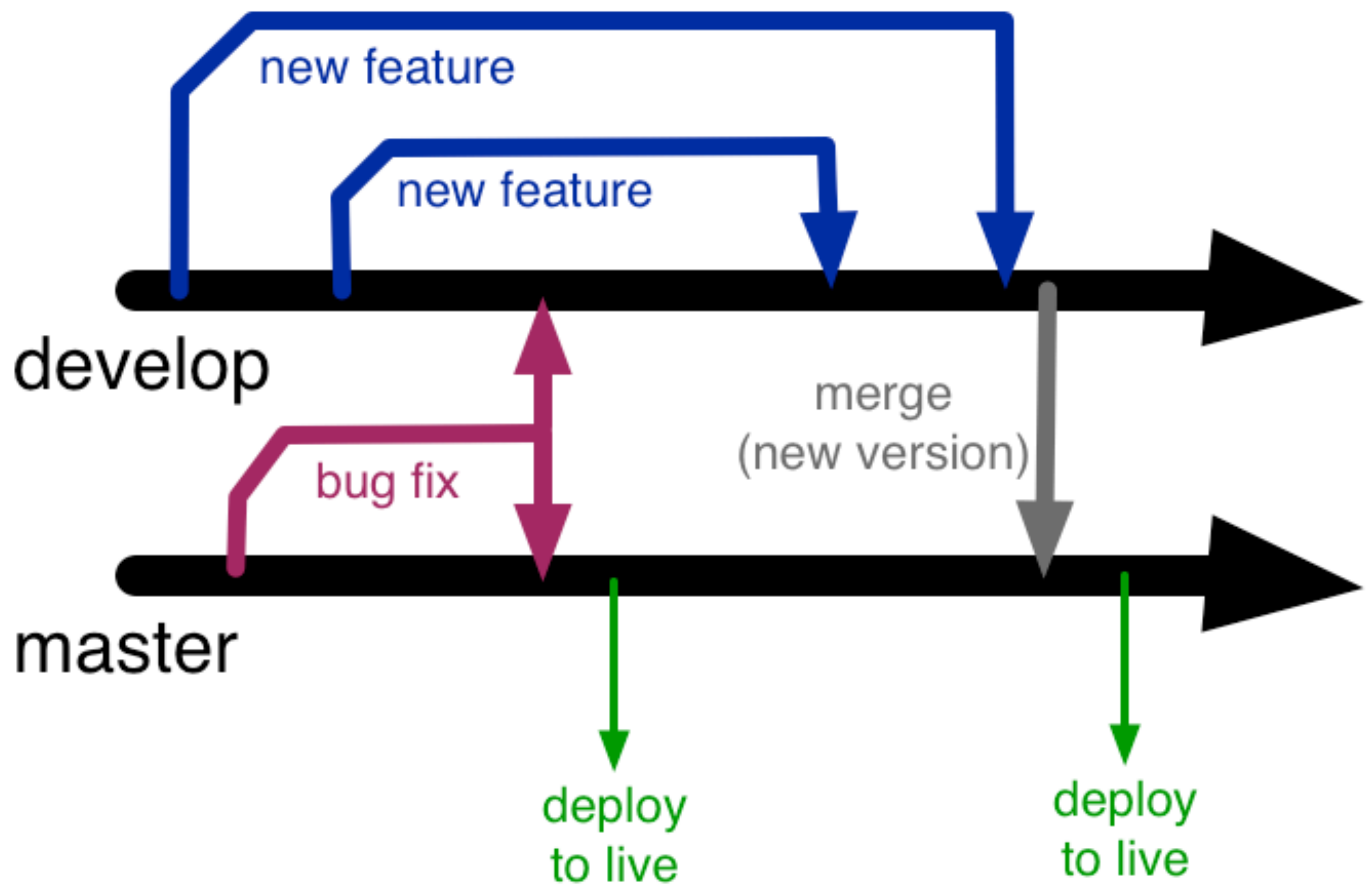
Trunk deployment



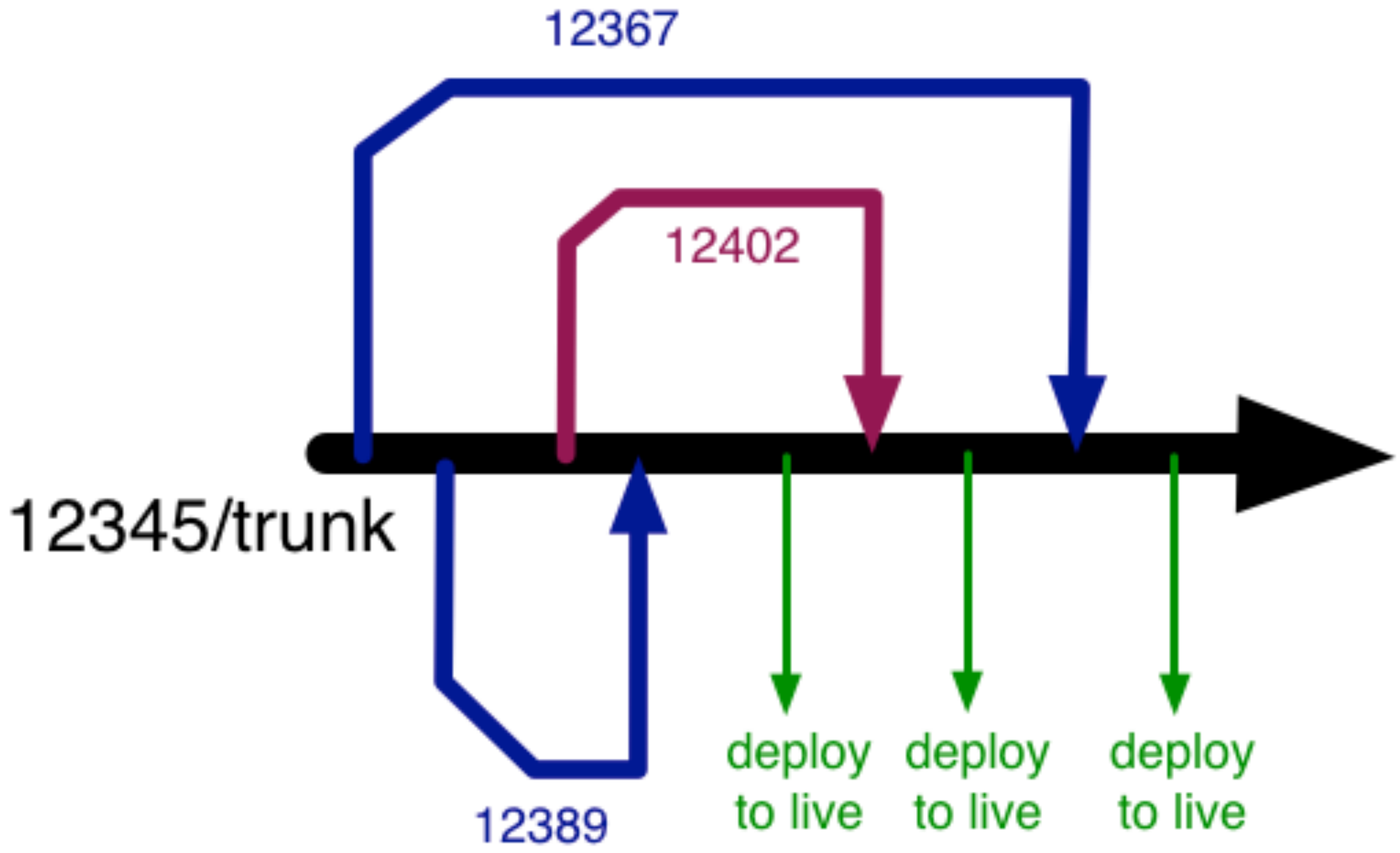
Live branch deployment



Git flow


new feature
new feature
develop
bug fix
merge
(new version)
master
deploy
to live
deploy
to live

This is what I do



Database considerations

One master database

- Live database holds master structure
- Copy master to everywhere else
- Advantages:
  - Simple to implement
- Disadvantages:
  - Backwards compatibility required
  - Doesn’t scale for multiple devs well
  - Easy to make mistakes

Migrations

- Versioned schemas
- Use *delta* files with UP and DOWN functionality
- Advantages:
  - version controlled
  - destructive changes possible (but don’t do it!)
- Disadvantages:
  - Can’t think of any!

Migrations tools

- LiquiBase
- DbDeploy
- Phinx
- South for the Winter (PDO)
- Framework specific
  - Doctrine, Cake, Akrabat_Db_Schema_Manager
- Home-brew script

For more info

“Database version control without the pain”
by Harrie Verveer

http://slidesha.re/gwq0aw

Also read:

- http://techportal.inviqa.com/?p=2864
- http://www.harrieverveer.nl/?p=247

Code considerations

Context awareness

- Configuration based on where the code has been deployed
- Automatic
  - Automatic detection based on URL?
  - Environment variable set in `vhost` defintion?
- Local configuration file

So what’s deployment all about?

Things to think about

- Transport to server
  - FTP? rsync? svn? git?
- File permissions
- Preserve user uploaded files
- Steps after upload
  - Stale cache?
  - Cache priming?

Server organisation

- Much easier if you control `vhosts`
- For multiple sites on same server:
  - Use predictable file locations for all sites
  - e.g:
    - /home/www/{site name}/live/current
    - /home/www/{site name}/staging/current
- With multiple servers for one site:
  - Keep them the same!

The deployment plan

Typical steps

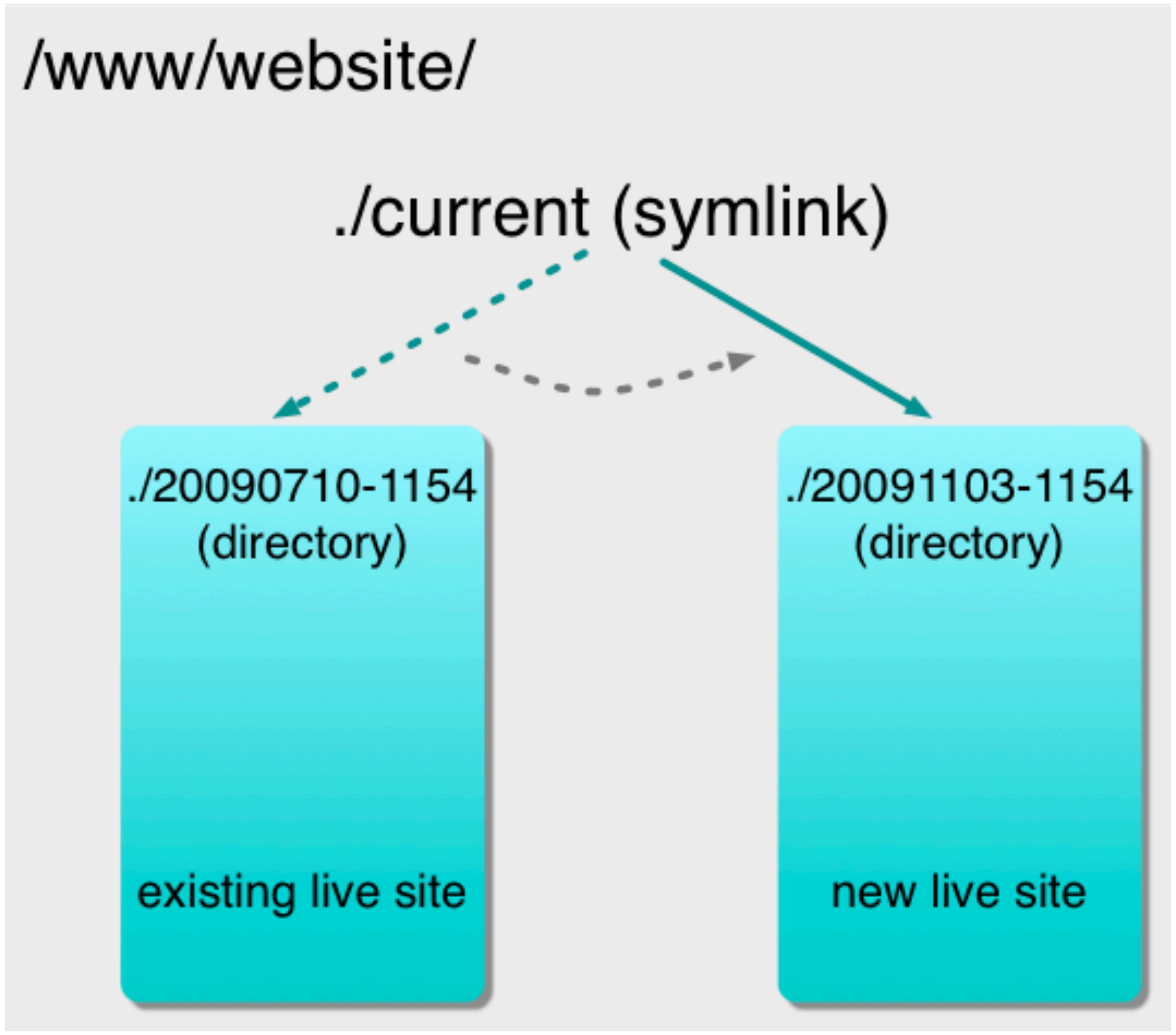
- Tag this release
- Set “under maintenance” page
- Transfer files to server
- Set file permissions as required
- Delete old cache files
- Run database migrations if required
- Remove “under maintenance” page

Here’s mine:

1. ssh into server
2. Ensure staging is up to date (`svn st -u`)
   If not, stop here!
3. Branch `master` to `release-{yymmdd-hhmm}`
4. `git clone` new release branch to a *new folder* in live directory
5. Set permissions on the /tmp folder for cache files

Finally

- Switch “current” symlink to new directory



Tools for automation

Simple scripts

- Pick a scripting language!
  - Perl, Python, Ruby, PHP, Bash, Bat(!)
- Simple to write and run
- However: hard to reconfigure

Example PHP script

```php
$cmd = "git tag -a $date -m \"Tag for automatic deployment\"";

ob_start();
system($cmd, $returnValue);
$output = ob_get_clean();

if (0 < $returnValue) {
    throw new Exception("Tagging failed.\n" . $output);
}
echo "Tagged to $date\n";
```

Phing

- PHP based build system based on Ant
- XML configuration files
- PEAR installation
- Integration with Git, Subversion and DbDeploy
- Expects to run `build.xml` in current directory
- `build.properties` contains config info

Phing philosophy

- Like *make*, build scripts consist of targets
- Targets can depend on other targets
  - “live” depends on “tag”, “checkout”, “migrate”
- Each target does the minimum it can

  e.g.
  - Create tag
  - checkout files to destination
  - migrate database

Example build.xml

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<project name="BRIBuild" default="deploy" basedir=".">
  <tstamp>
    <format property="date" pattern="%Y%m%d-%H%M" />
  </tstamp>
  <property file="build.properties" />
  <property name="trunkpath" value="${svnpath}/${website}/trunk" />
  <property name="tagpath"
        value="${svnpath}/${website}/tags/${date}" />

  <target name="deploy" depends="tag" />
  <target name="tag" description="Tag trunk">
    <exec command="svn cp -m 'Tag for automatic deployment'
          ${trunkpath} ${tagpath}" />
    <echo msg="Tagged trunk to ${date}" />
  </target>
</project>
```

Capistrano

- Ruby-based
- Multi-server
- Parallel
- SSH into servers
- Write tasks like Phing/Ant/etc

Example capfile

```
role :webserver, "snoopy", "yogi", "garfield"

desc "Show free diskspace"
task :df, :roles => :webserver do
  run "df -h /"
end
```

Running capistrano

```
$ cap df
  * 2013-02-18 07:57:00 executing `df'
  * executing "df -h /"
    servers: ["dangermouse", "daffy", "batfink"]
    [batfink] executing command
    [daffy] executing command
 ** [out :: daffy] Filesystem            Size  Used Avail Use% Mounted on
 ** [out :: daffy] /dev/sda2             924G  457G  421G  53% /
 ** [out :: batfink] Filesystem            Size  Used Avail Use% Mounted on
 ** [out :: batfink] /dev/sda1             184G   31G  144G  18% /
    [dangermouse] executing command
 ** [out :: dangermouse] Filesystem            Size  Used Avail Use% Mounted
on
 ** [out :: dangermouse] /dev/mapper/VolGroup00-LogVol00
 ** [out :: dangermouse] 287G  180G   93G  66% /
    command finished in 172ms
```

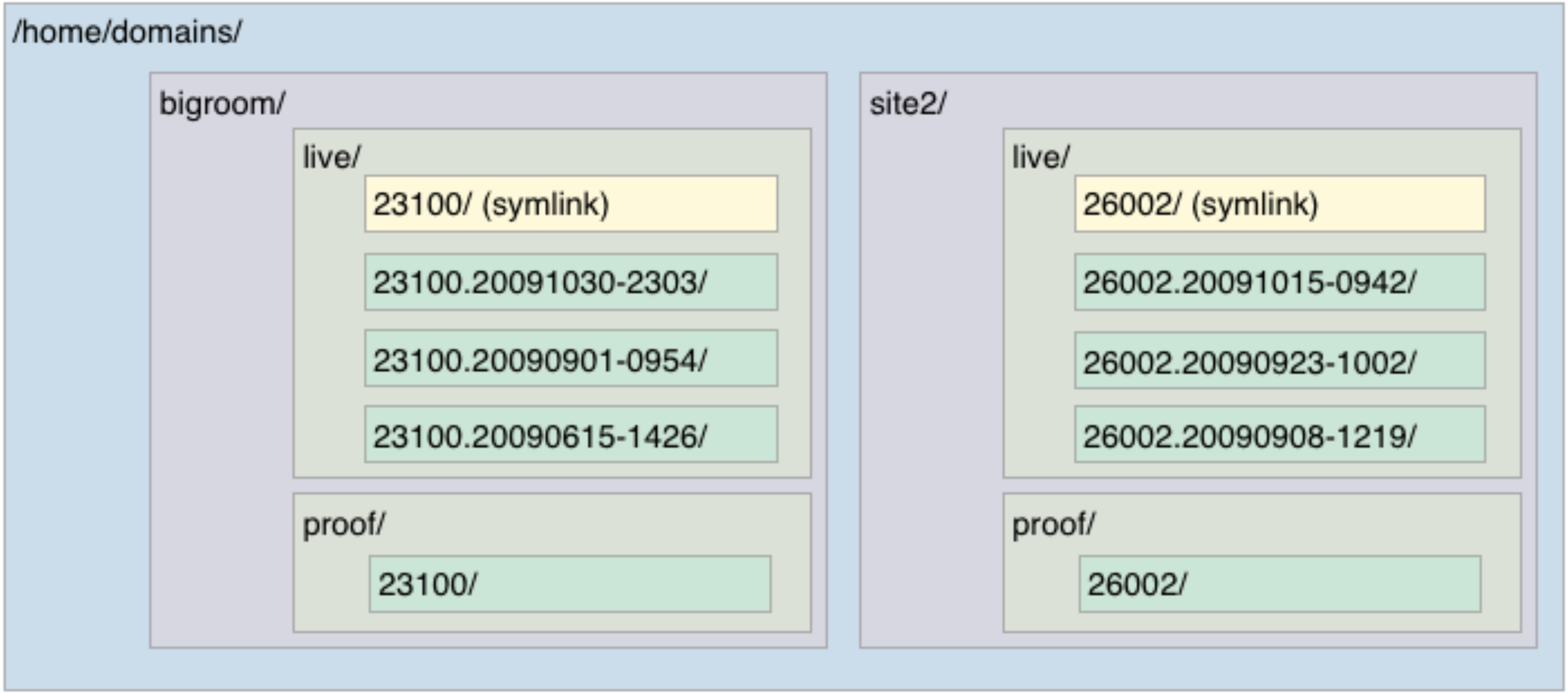
My deployment system

deploy.php

```
~$ deploy.php 23100
BRI Server side deploy script
Version 1.2, 2012

Found bigroom/23100/live/
Deploying master to 23100.20130219-2047 subfolder... done
Tagging to 20130219-2047... done
Checking out tag... done
Changing symlink to new checkout
Cleaning up older checkouts
23100 successfully deployed to bigroom/23100/live/
```

Our server layout



deploy.php

- Custom PHP script
- Relies on environment variable: WWW_DIR
- Advantages:
  - Custom designed to fit our way of working
  - PHP! Quick and easy to write (for us).
- Disadvantages:
  - Hard to alter for a specific server
  - Hard to change methodology

FTP using Phing (1)

```
<project name="project" basedir="." default="deploy">
  <property file="build.properties" />
  <property name="trunkpath"
      value="${svnpath}/${website}/trunk" />
  <fileset dir="${exportdir}/" id="files">
    <include name="**/*" />
  </fileset>

  <target name="deploy" depends="svnexport,ftp-upload" />

  <target name="nexport">
    <delete dir="${exportdir}" />
     <exec
      command="git archive ${git_tag} | tar -x -C ${exportdir}" />
  </target>
```

FTP using Phing (2)

```
  <target name="ftp-upload">
    <echo msg="Deploying application files" />
    <ftpdeploy
      host="${ftp.host}" port="${ftp.port}"
      username="${ftp.username}" password="${ftp.password}"
      dir="${ftp.dir}">
      <fileset refid="${files}" />
    </ftpdeploy>
  </target>

</project>
```

FTP with Phing

- Per-website build.xml for custom deployments
- Advantages:
  - Leverages other people's experiences
  - Was very fast to create
  - Works where ssh not available!
- Disadvantages:
  - New technology to be learnt
  - Phing beta and Pear_Version_SVN alpha

Capistrano: Lib update

```
role :webserver, "snoopy", "yogi", "garfield"

desc "Update M5 library"
task :deploy, :roles => :webserver do
    run "cd /usr/local/include/M5/ && git pull --ff-only origin master 2>&1"
end
```

Capistrano: tasks

```
# standard_tasks.rb
role :webserver, "#{shortname}.bigroominternet.co.uk"

desc "Display hostname that this site is deployed on"
task :hostname, :roles => :webserver do
    run "hostname"
end

desc "Update proof site"
task :updateproof, :roles => :webserver do
    run "cd $WWW_DIR/#{shortname}/proof/#{job_number}
        && git pull --ff-only origin master 2>&1
end

desc "Deploy live site"
task :deploylive, :roles => :webserver do
    run "deploy #{job_number}"
end
```

Capistrano: capfile

```
# project’s capfile
set :job_number, "bingo"
set :shortname, "bingo"

load '/usr/local/bri_scripts/capistrano/standard_tasks.rb'
```

Rollback

Emergency roll-back

Just change the symlink!

Complete roll-back

- Write `rollback.php` or create a Phing build task

- Put the server back to where it was before
  - Change the symlink
  - Delete the deployed directory
- Database rollback
  - Run `down()` delta in your migration tool

To summarise

1. Automated deployment prevents mistakes
2. It’s not hard
3. Easy roll-back is priceless

Questions?

email: rob@akrabat.com
twitter: @akrabat

Thank you

email: rob@akrabat.com
twitter: @akrabat