engage

Adm04.The Lazy Admin Wins the Game. A 'How To' guide

Daniele Vistalli @ Factor-y Srl

Matteo Bisi @ Factor-y Srl

engage

Our Agenda

- Our "lazyness" defininition
- Tools available
- Some examples
- Links

engage

Daniele Vistalli – CEO & CTO

IBM Champion Social Business 2017,2018

Social:

- @danielevistalli
- https://www.linkedin.com/in/dvistalli
- daniele.vistalli@factor-y.com

engage

Matteo Bisi – Senior System Engineer

IBM Champion Social Business 2014,15,16,17,18

Blogger – www.msbiro.net, blog.msbiro.net

Social:

- @mbisi78
- https://it.linkedin.com/in/matteobisi
- matteo.bisi@factor-y.com

engage

Let's set expectations

1. This is **not a crash course**
2. You won't be throwing your way of life out of the window right after this presentation
3. This is an **introductory overview** of tools you may or may not know, please interact & **share experience**
4. Everything happens in **steps**, choose the one that fits your need better and start **improving your life** with some laziness. Don't try to do it all at once.. that's **anti-lazy**

engage

"lazyness" defined

Our lazy admin

- Can do his work, he’s a pro
- Does not avoid learning new tricks

but

- He is smart enough to not reinvent the wheel

Everyday problems

Memory loss ... how was that command ???

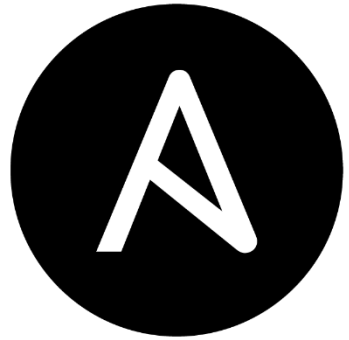
Configuration issues ... I changed it last week.. what was before

Tedious work .... A lot of mouse click (or cut & paste)

engage

Tools of the modern (lazy) admin

- Git
- Ansible
- Jenkins
- Special mention:
  - IBM Installation Manager



engage

Git is your «time machine»

Version everything to prevent memory loss

engage

git

Git is born in 2005 to manage sources of the “simplest” project ever: The Linux Kernel

Today it is the de-facto standard for

- source code (or any kind of file based repository) tracking and versioning
- NPM Modules, Node Projects, Java, Ansible, Jenkins… add your own
- Available everywhere, integrated everywere

engage

Git manages «Repositories»

- Git is extremely complex (if you like to deep dive)
- Git is extremely easy (if you focus on your need)

A **repository** is a directory containing:

- Your code, let's call this the «**working copy**»
- Git history files and configuration, in the .git hidden folder

Create a repository >> ***git init***

- It adds a «.git» folder to your directory
- Nothing is added to the history so far

Working on sources and staging

Work in your directory as you do today

- Add files
- Make changes
- Delete files

Now it’s time to manage your changes:

git status: shows you changes

git add: tells git you want to «track» changes on a file

git rm: removes a file

git commit: saves your chages into a «commit» giving the changeset a trackable identifier and a comment

Many options exist, you learn over time, or you use a powerful client (eg. SourceTree)

Introducing «Remotes»

- You can use git «locally» and have an entire version control but if you need to
  - Share & work with others
  - Secure yourself a backup

Then you need «Remotes» or «Remote repository»

A server-based replica of the repository from which you can

- pull changes
- and then push your modifications

Servers: GitHub / BitBucket / GitLab / Git Server

engage

Branches

If you need to test changes:

1. Create a «branch»
2. Make your changes
3. Stage your changes & commit
4. Test your changes (build, deploy, whatever)

All your changes are in a «branch», those changes to not impact other's work till you decide to «merge».

You can have as many branches as you like.

You want to have a «master» branch which represents the real/true/valid content for your project

engage

Merges

After you created branches you may need to consolidate

- Make a valid change into the «master» branch
- Accept changes done by others and consolidate

Merging is a **process**:

1. Involves 2 branches
2. A source and a target
3. Applies «diffs» and resolution to changes on files
4. Creates a new «commit» in the target branch with the merged files

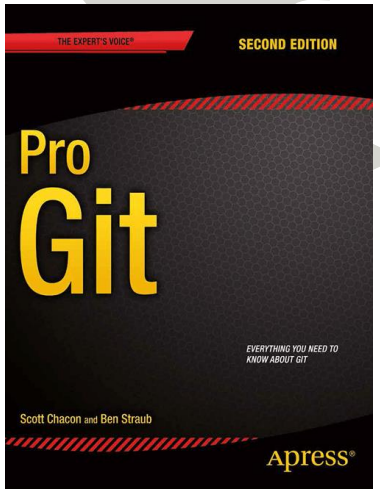
engage

A few resources

- SourceTree (my favorite UI client)

- The Git Book: https://git-scm.com/book/en/v2
  - This is a good ready, don't try to memorize it

- The GitFlow Workflow (most important for developers, but better you know)
  - https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow
  - http://nvie.com/posts/a-successful-git-branching-model/



Admin’s usage of Git (a few ideas)

In a CI/CD environment

- Is the source for code / scripts

In an «ansible» context

- Manage your inventories and playbooks
- Create modules & roles

In an IBM Connections World (we use it)

- Store & track changes to LotusConnectionsConfig folder
- Store & track connections customizations
- Store & track «TDISOL»

Jenkins, your butler

Computers were created to take work away from you … let's do it

Jenkins

Jenkins

Jenkins is a CI / CD server to do work in place of you

- CI : Continuous integration
- CD: Continuous delivery

Born in 2004 at Sun as “Hudson” it later was forked by the community as Jenkins. Oracle claimed Hudson as its trademarks. The community went away.

Guess what… **Jenkins** is today the **winner**.

Get it at: https://jenkins.io/

engage

What can Jenkins do for me

If it’s repeatable, Jenkins can help, and adds:

- Configuration repository for «Jobs» & «Pipelines»
- Credentials store (so you don’t save password in code/scripts)
- Plugins (to integrate anything)
- Historical perspective on executions
- Pipeline «mindset», follows you from dev to production.

engage

What's a Job

A set of steps to achieve a result

No, it's not a script because:

- A job has an history
- Stores outputs
- Track exit code
- Track the execution environment
- Can be run on remote hosts (Jenkins nodes)

engage

What's a Job, part 2 – Build it

1. Check out code from git (scripts ?)
2. Set the context, parameters, variables
3. Use jenkins plugins to do things
4. Get credentials from the credential store
5. Manage outcomes conditionally
6. Fire other cascading jobs

When a jub runs …

Executes steps you defined

After a jub has run

- Review & keep the output
- Check the execution environment
- Run «post» steps to consolidate results
  - Save files
  - Fire events
  - Send notifications
- More plugins -> More capabilities

Explore jenkins plugins: https://plugins.jenkins.io/

engage

Other tricks

- Invoke a jenkins job as a REST API
- Deploy remote «nodes» to distribute load
- Use Ansible with Jenkins to improve automation (key for system administrators)

- If you need something (and it makes sense) there’s probably a plugin for that.

engage

What we use it for ?

- Building our code (this is for devs)
- Deploying apps to 6 lines of WebSphere Portal Servers at a customer
  - Applications
  - Configurations
  - Reporting
- Deploying apps to customers
  - Track every deploy, notify changes and generate reports

engage

Ansible

Scripting for admins, done right

Ansible

Agentless and powerful open source IT automation tool

- Agentless , it runs over SSH (or PowerShell remote)
- Powerfull , based on Python

Use Cases

- Provisioning
- Configuration management
- Application deployement
- Continuous delivery
- Security & Compliance
- Orchestration

Inventory

INI

```
mail.example.com

[webservers]
foo.example.com
bar.example.com

[dbservers]
one.example.com
two.example.com
three.example.com
```

YAML

```
all:
  hosts:
    mail.example.com:
  children:
    webservers:
      hosts:
        foo.example.com:
        bar.example.com:
    dbservers:
      hosts:
        one.example.com:
        two.example.com:
        three.example.com:
```

Default /etc/ansible/hosts

Multiple inventory allowed to , use –i <path> on command line to specify others

engage

Variables

stored inside Inventory

host

```
[atlanta]
host1 http_port=80 maxRequestsPerChild=808
host2 http_port=303 maxRequestsPerChild=909
```

Group

```
[atlanta]
host1
host2

[atlanta:vars]
ntp_server=ntp.atlanta.example.com
proxy=proxy.atlanta.example.com
```

```
atlanta:
  hosts:
    host1:
    host2:
  vars:
    ntp_server: ntp.atlanta.example.com
    proxy: proxy.atlanta.example.com
```

variables

Groups of Groups, and Group Variables

```
[atlanta]
host1
host2

[raleigh]
host2
host3

[southeast:children]
atlanta
raleigh

[southeast:vars]
some_server=foo.southeast.example.com
halon_system_timeout=30
self_destruct_countdown=60
escape_pods=2

[usa:children]
southeast
northeast
southwest
northwest
```

```
all:
  children:
    usa:
      children:
        southeast:
          children:
            atlanta:
              hosts:
                host1:
                host2:
            raleigh:
              hosts:
                host2:
                host3:
          vars:
            some_server: foo.southeast.example.com
            halon_system_timeout: 30
            self_destruct_countdown: 60
            escape_pods: 2
        northeast:
        northwest:
        southwest:
```

engage

variables

Default groups

- All
- ungrouped

The order/precedence is (from lowest to highest)

- all group (because it is the ‘parent’ of all other groups)
- parent group
- child group
- Host

Ansible use Jinja2 templating to enable dynamic expressions and access to variables

Ready to run commands?

engage

Play with Ansible

Ad Hoc Command (some examples)

```
$ ansible webservers -m file -a "dest=/path/to/c mode=755 owner=mdehaan group=mdehaan state=directory"
```

```
$ ansible webservers -m file -a "dest=/path/to/c state=absent"
```

```
$ ansible webservers -m yum -a "name=acme state=latest"
```

Play with Ansible

- Playbook

It’s a READABLE collections of Ansible commands , written inside a YAML file and could be launched against host(s) or group(s).

Commands are grouped by tasks who calls ansible modules

```
- hosts: webservers
  remote_user: root

  tasks:
  - name: ensure apache is at the latest version
    yum:
      name: httpd
      state: latest
  - name: write the apache config file
    template:
      src: /srv/httpd.j2
      dest: /etc/httpd.conf
```

engage

Play with Ansible

- Role(s)

It’s a collections of files with folder structur that allow admins to keep simple and flexible complex authomation.

Example projext structure:

```
site.yml
webservers.yml
fooservers.yml
roles/
   common/
     tasks/
     handlers/
     files/
     templates/
     vars/
     defaults/
     meta/
   webservers/
     tasks/
     defaults/
     meta/
```

engage

IBM Installation Manger

It’s an installer

- /opt/ibm/InstallationManger/eclipse/IBMIM

It’s a SMART installer ☺

- Install from local ad remote repository
- It could record and play cofigurations
- It could roll back versions and config
- It owns also a Web Interface !

  /opt/ibm/InstallationManger/eclipse/web/ibmim-web

  http://serverIP:9090/ibmim

engage

Record and play !

Record

/opt/IBM/InstallationManager/eclipse/IBMIM -record /root/connections6dev.xml -skipInstall /tmp/SkipInstall

Play

/opt/IBM/InstallationManager/eclipse/tools/imcl -acceptLicense -sVP -log /tmp/swInstall.log -input /tmp/response/swResp.rsp

Let’s go to use this with Ansible !

engage

Intsallation manager inside Ansible playbooks

```
- name:                  Install IBM WAS ND Fixes Software
  command:
    chdir={{ __tmp_dir }}
    {{ __iim_install_location }}/eclipse/tools/imcl -acceptLicense -sVP -log {{ __log_file }} input {{ __rsp_file }}
  register:              cout
  changed_when:          ( cout.stdout.find(__version_check) != -1)
  when:                  version_already_installed.rc != 0
```

Example: deploy DB2 11.1FP3

Response file -> template

```
*-------------------------------------------------------
* Generated response file used by the DB2 Setup wizard
* generation time: 3/28/18 2:46 PM
*-------------------------------------------------------
*  Product Installation
LIC_AGREEMENT        = ACCEPT
PROD         = DB2_SERVER_EDITION
FILE         = {{ db2_install_location }}
INSTALL_TYPE         = CUSTOM
COMP         = TSAMP
COMP         = IINR_APPLICATIONS_WRAPPER
COMP         = SQL_PROCEDURES
COMP         = ORACLE_DATA_SOURCE_SUPPORT
COMP         = ODBC_DATA_SOURCE_SUPPORT
COMP         = IINR_SCIENTIFIC_WRAPPER
COMP         = INSTANCE_SETUP_SUPPORT
COMP         = TERADATA_DATA_SOURCE_SUPPORT
COMP         = LDAP_EXPLOITATION
COMP         = FED_DATA_SOURCE_SUPPORT
COMP         = CONNECT_SUPPORT
```

engage

Example: deploy DB2 11.1

```
- name:                  Download db2 Server
  get_url:
    url:                 "{{ db2_repository_url }}/{{ __bin_file }}"
    dest:                "{{ __tmp_dir }}/{{ __bin_file }}"
    mode:                0755
    checksum:            sha256:{{ __bin_file_sha256 }}
  when:                  version_already_installed.rc != 0

- name:                  Extract db2 Server
  unarchive:
    src:                 "{{ __tmp_dir }}/{{ __bin_file }}"
    dest:                "{{ __tmp_dir }}/"
    copy:                no
  when:                  version_already_installed.rc != 0

- name:                  Generate Response file
  template:
    src:                 "{{ __tpl_file }}"
    dest:                "{{ __rsp_file }}"
  when:                  version_already_installed.rc != 0

- name:                  Install/Update db2 Server
  command:
    chdir={{ __tmp_dir }}/universal
    {{ __tmp_dir }}/universal/db2setup -r {{ __rsp_file }} -i silent
  ignore_errors:         yes
  when:                  version_already_installed.rc != 0
```


Q&A

engage

Links

http://www.ansible.com

https://github.com/ebasso/ansible-ibm-websphere

https://github.com/Factor-y/ansible-ibm-websphere

engage