IT/Dev[op]s Army Knives Tools for the researcher
a journey from SSH to Git

Sebastien Varrette

Parallel Computing and Optimization Group (PCOG), University of Luxembourg (UL), Luxembourg
Summary

1 Introduction
   Agenda
   Overview of managed IT Infrastructure

2 IT/Dev[op]s Army Knives Tools
   SSH Secure Shell
   PGP / GPG: Gnu Privacy Guard
   Vagrant
   Puppet
   Ruby / Python / Markdown-based Documentations
   Password Management

3 Research Computing Platform @ UL

4 Git[Lab] @ UL and VCS

Git[Lab] Around You
   About Version Control System (VCS)

5 Git Basics
   Installing Git
   Git theory
   Basic Commands
   Branching and Merging

6 Collaborating / Working together

7 Advanced Git Topics
   Git Submodules
   Rebasng
   Using Git over Subversion Repository
   More Cool stuff
Introduction

Agenda
Overview of managed IT Infrastructure

IT/Dev[op]s Army Knives Tools
SSH Secure Shell
PGP / GPG: Gnu Privacy Guard
Vagrant
Puppet
Ruby / Python / Markdown-based Documentations
Password Management

Research Computing Platform @ UL

Git[Lab] @ UL and VCS

Git[Lab] Around You
About Version Control System (VCS)

Git Basics
Installing Git
Git theory
Basic Commands
Branching and Merging

Collaborating / Working together

Advanced Git Topics
Git Submodules
Rebasing
Using Git over Subversion Repository
More Cool stuff
Introduction

Agenda
Overview of managed IT Infrastructure

IT/Dev[op]s Army Knives Tools
SSH Secure Shell
PGP / GPG: Gnu Privacy Guard
Vagrant
Puppet
Ruby / Python / Markdown-based Documentations
Password Management

Research Computing Platform @ UL

Git[Lab] @ UL and VCS

Git[Lab] Around You
About Version Control System (VCS)

Git Basics
Installing Git
Git theory
Basic Commands
Branching and Merging

Collaborating / Working together

Advanced Git Topics
Git Submodules
Rebasing
Using Git over Subversion Repository
More Cool stuff
Review some of the most sensible tools every researcher

→ ... in computer science but not only

Focus on key IT [DevOps] Tools: **SSH, PGP, Vagrant** ...

Review also some best-practice for your daily work

→ Sand-boxing in Python and Ruby for your prototyping
→ Password Management

Overview of the Research Computing platforms @ UL

Overview of Version Control System (VCS) and **Git** in particular
Introduction

Agenda Part I (9h45-12h00)

**Location**: room B21, campus Kirchberg

- **SSH** Secure Shell
  - → Overview and Basic usage
  - → Advanced usage (proxy SOCKS, multi-jump w. ProxyCommand)
- **PGP / GPG**: Gnu Privacy Guard
- **Vagrant**: Development environment made easy
- **Puppet**: Configuration Management
  - Sandbox and reproducible running environment across developers
  - → Ruby & Python
- **Markdown**-based documentation, articles and slides
  - → Overview of the Markdown syntax
  - → Git-based Markdown Wiki: gollum, mkdocs
  - → using Markdown with LaTeX and Beamer
- **Password Management**
- Overview of the Research Computing platforms @ UL
Introduction

Agenda Part II (13h15 - 15h00)

Location: room B21, campus Kirchberg

- Introduction to Version Control System (VCS)
- Git Basics
  - Installing Git
  - Git theory
  - Basic Commands Branching and Merging
- Collaborating / Working together
- Advanced Git Topics
  - Git Submodules
  - Rebasing
  - Using Git over Subversion Repository
- More Cool stuff
Introduction

Technical Recommendation

- All attendees are strongly encouraged to bring their computer laptop to the talk to integrate a set of hands-on exercises.

- Start to install some components:
  - Mac OS: Homebrew (http://brew.io)
  - Virtualbox: (https://www.virtualbox.org/)
1 Introduction
   Agenda
   Overview of managed IT Infrastructure

2 IT/Dev[op]s Army Knives Tools
   SSH Secure Shell
   PGP / GPG: Gnu Privacy Guard
   Vagrant
   Puppet
   Ruby / Python / Markdown-based Documentations
   Password Management

3 Research Computing Platform @ UL

4 Git[Lab] @ UL and VCS

5 Git Basics
   Installing Git
   Git theory
   Basic Commands
   Branching and Merging

6 Collaborating / Working together

7 Advanced Git Topics
   Git Submodules
   Rebasings
   Using Git over Subversion Repository
   More Cool stuff
Introduction

Typical [UL] IT Infrastructure

*.<domain>-intern.com

bastion[1-N].<domain>.com

S. Varrette (PCOG Research unit)
Introduction

Typical [UL] IT Infrastructure

*.<domain>-intern.com

backup

gforge, gitlab

jenkins

ad

mssql[-demo]

oracle[-demo]

etc...

bastion[1-N].<domain>.com

S. Varrette (PCOG Research unit)  IT/Dev[op]s Army Knives Tools for the researcher
Introduction

Typical [UL] IT Infrastructure

*.<domain>-intern.com

backup1

gforge, gitlab

ipa, ldap

*apptest*

ad

mssql[-demo]

oracle[-demo]

jenkins

ejc

etc...

*.<domain>-intern.com

bastion[1-N].<domain>.com

S. Varrette (PCOG Research unit)

IT/Dev[op]s Army Knives Tools for the researcher
**Introduction**

**Typical [UL] IT Infrastructure**

*<domain>-intern.com*

backup1, gforge, gitlab, ipa, ldap, ad, mssql[-demo], oracle[-demo], jenkins, *apptest*, etc...

*<domain>*-intern.com

**TOTAL resources managed**

- 222 servers
  - 66 physical
  - 156 VMs (Xen, KVM guests)
- 492 computing nodes (part of UL HPC facility)
- 5.35 PB raw shared storage (NFS / GPFS / Lustre)
- 4 system administrators / 2 sites
- Operating System: Debian, CentOS

**IT/Dev[op]s Army Knives Tools for the researcher**

S. Varrette (PCOG Research unit)
Introduction

UL HPC Software Stack

- **Operating System**: Linux Debian (CentOS on storage servers)
- **Remote connection to the platform**: SSH
- **User SSO**: OpenLDAP-based
- **Resource management**: job/batch scheduler: OAR
- **(Automatic) Computing Node Deployment**:
  - FAI (Fully Automatic Installation)
  - Puppet
  - Kadeploy
- **Platform Monitoring**: OAR Monika, OAR Drawgantt, Ganglia, Nagios, Puppet Dashboard etc.
- **Commercial Softwares**:
  - Intel Cluster Studio XE, TotalView, Allinea DDT, Stata etc.
Introduction
Agenda
Overview of managed IT Infrastructure

IT/Dev[op]s Army Knives Tools
SSH Secure Shell
PGP / GPG: Gnu Privacy Guard
Vagrant
Puppet
Ruby / Python / Markdown-based Documentations
Password Management

Research Computing Platform @ UL

Git[Lab] @ UL and VCS

Git[Lab] Around You
About Version Control System (VCS)

Git Basics
Installing Git
Git theory
Basic Commands
Branching and Merging

Collaborating / Working together

Advanced Git Topics
Git Submodules
Rebasing
Using Git over Subversion Repository
More Cool stuff
IT/Dev[op]s Army Knives Tools

Summary

1 Introduction
   Agenda
   Overview of managed IT Infrastructure

2 IT/Dev[op]s Army Knives Tools
   SSH Secure Shell
   PGP / GPG: Gnu Privacy Guard
   Vagrant
   Puppet
   Ruby / Python / Markdown-based Documentations
   Password Management

3 Research Computing Platform @ UL

4 Git[Lab] @ UL and VCS

5 Git Basics
   Installing Git
   Git theory
   Basic Commands
   Branching and Merging

6 Collaborating / Working together

7 Advanced Git Topics
   Git Submodules
   Rebasings
   Using Git over Subversion Repository
   More Cool stuff

S. Varrette (PCOG Research unit)
SSH Secure Shell: Overview

- Ensure secure connection to remote (UL) server
- (optional) policy: **restricted** to public key authentication
  - over non-standard port (8022)
- Rule: 1 machine = 1 key pair (ideally with passphrase protection)
SSH Setup on Linux / Mac OS

- OpenSSH natively supported; configuration directory: ~/.ssh/
  - package openssh-client (Debian-like) or ssh (Redhat-like)

```bash
$> ssh-keygen -t dsa # SSH DSA Key-Pair generation
```

- Public key: ~/.ssh/id_{rsa,dsa}.pub
  - This one is the only one SAFE to distribute.

- Private (identity) key ~/.ssh/id_{rsa,dsa}

- Configuration: ~/.ssh/config. Format:

  ```
  Host <shortname>
  Port <port>
  User <login>
  Hostname <hostname>
  ```
SSH Setup on Windows

- **Putty Suite**, includes: http://www.chiark.greenend.org.uk/~sgtatham/putty/
  - PuTTY, the free SSH client
  - Pageant, an SSH authentication agent for PuTTY tools
  - PLink, the PuTTY CLI
  - PuTTYgen, an RSA and DSA key generation utility
SSH Setup on Windows

- **Putty Suite**, includes:
  - PuTTY, the free SSH client
  - Pageant, an SSH authentication agent for PuTTY tools
  - PLink, the PuTTY CLI
  - PuTTYgen, an RSA and DSA key generation utility

PuTTY ≠ OpenSSH
IT/Dev[op]s Army Knives Tools

### SSH Setup on Windows

**Putty Suite**, includes:  
- **PuTTY**, the free SSH client  
- **Pageant**, an SSH authentication agent for PuTTY tools  
- **PLink**, the PuTTY CLI  
- **PuTTYgen**, an RSA and DSA key generation utility

---

**PuTTY ≠ OpenSSH**

- Putty keys are **NOT** supported by OpenSSH (yet can be exported)  
- Binding Pageant with OpenSSH agent is **NOT** natively supported  
  - Third-party tools like `ssh-pageant` are made for that  
- with PLink, hostnames eventually refer to **PuTTY Sessions**  
  - **NEVER** to SSH entries in `~/.ssh/config`  
  - This usage might be hidden... Ex: `$GIT_SSH` etc.
SSH Basic Usage

*.<domain>-intern.com

gitlab

bastion1.<domain>.com

S. Varrette (PCOG Research unit)
SSH Basic Usage

*.<domain>-intern.com

gitlab

bastion1.<domain>.com

SSH
SSH Basic Usage

```
*.<domain>-intern.com

OpenSSH ~/.ssh/config (Mac / Linux)
Host uni *
  User <LOGIN>
  Port 8822
Host uni_bastion1
  Hostname bastion1.<domain>.com
```

S. Varrette (PCOG Research unit)
SSH Basic Usage

**OpenSSH**

```
~/.ssh/config (Mac / Linux)
Host uni_*
  - Hostname: bastion1.<domain>.com
  - Port: 8822
  - Connection/Data: username: <LOGIN>
```

**PuTTY / PLink / Pageant (Windows)**

```
Session "uni_bastion1"
  - Hostname: bastion1.<domain>.com
  - Port: 8822
  - Connection/Data: username: <LOGIN>
```

**SSH**

- Hostname: bastion1.<domain>.com
- Port: 8822
- Connection/Data: username: <LOGIN>
SSH Advanced Usage: SOCKS Proxy

*.<domain>-intern.com

gitlab

bastion1.<domain>.com

S. Varrette (PCOG Research unit)
SSH Advanced Usage: SOCKS Proxy

**OpenSSH**  
~/.ssh/config  (Mac / Linux)

- Host `uni_*`
- Hostname `bastion1.<domain>.com`
- Port 8822
- Connection/Data: username: `<LOGIN>`

**PuTTY / PLink / Pageant (Windows)**

Session “uni_bastion1”
- Hostname: `bastion1.<domain>.com`
- Port: 8822
- Connection/Data: username: `<LOGIN>`

**Example Configuration**

```
*.<domain>-intern.com
  Host uni_`
  Hostname `bastion1.<domain>.com`
  Port 8822
  Connection/Data: username: `<LOGIN>`
```

```
https://gitlab.<domain>-intern.com
```

S. Varrette  (PCOG Research unit)
SSH Advanced Usage: SOCKS Proxy

PuTTY / PLink / Pageant (Windows)
- Session "uni_bastion1"
  - Hostname: bastion1.<domain>.com
  - Port: 8022
  - Connection/Data: username: <LOGIN>

OpenSSH ~/.ssh/config (Mac / Linux)
- Host uni_+
  - User: <LOGIN>
  - Port 8022
- Host uni_bastion1
  - Hostname bastion1.<domain>.com

HTTPS/gitlab.<domain>-intern.com
DOMAIN <domain>-intern.com
NOT FOUND
SSH Advanced Usage: SOCKS Proxy

**OpenSSH**
- *~/.ssh/config* (Mac / Linux)
  - Host `uni_*`
  - User `<LOGIN>`
  - Port 8822
  - Hostname `bastion1.<domain>.com`

**PuTTY / PLink / Pageant (Windows)**
- Session "uni_bastion1"
  - Hostname: `bastion1.<domain>.com`
  - Port: 8822
  - Connection/Data: username: `<LOGIN>`
  - Connection/SSH/Tunnels: Port 1080, Dynamic

**SOCKS LISTEN:** `localhost:1080`

- `bastion1.<domain>.com`
- `*.<domain>-intern.com`
- `SSH -D 1080 (SOCKS 5 Proxy)`

S. Varrette (PCOG Research unit)
SSH Advanced Usage: SOCKS Proxy

**OpenSSH**

1. 
   - Hostname: `bastion1.<domain>.com`
   - Port: 8822
   - Connection/Data: username: `<LOGIN>`
   - Connection/SSH/Tunnels: Port 1080, Dynamic

2. 
   - SOCKS LISTEN: `localhost:1080`

3. 
   - `SSH -D 1080 (SOCKS 5 Proxy)`

**PuTTY / PLink / Pageant (Windows)**

1. 
   - Session “uni_bastion1”
     - Hostname: `bastion1.<domain>.com`
     - Port: 8822
     - Connection/Data: username: `<LOGIN>`
     - Connection/SSH/Tunnels: Port 1080, Dynamic

2. 
   - SOCKS LISTEN: `localhost:1080`

**Example URL**

- `https://gitlab.<domain>-intern.com`
SSH Advanced Usage: SOCKS Proxy

**PuTTY / PLink / Pageant (Windows)**
- Session "uni_bastion1"
  - Hostname: bastion1.<domain>.com
  - Port: 8822
  - Connection/Data: username: <LOGIN>
  - Connection/SSH/Tunnels: Port 1080, Dynamic

**OpenSSH ~/.ssh/config (Mac / Linux)**
- Host uni_
  - User: <LOGIN>
  - Port 8822
- Host uni_bastion1
  - Hostname bastion1.<domain>.com

**IT/Dev[op]s Army Knives Tools**

SSH -D 1080 (SOCKS 5 Proxy)

SOCKS LISTEN: localhost:1080

HTTPS://gitlab.<domain>-intern.com
SSH Advanced Usage: SOCKS Proxy

**PuTTY / PLink / Pageant (Windows)**

- Session "uni_bastion1"
  - Hostname: bastion1.<domain>.com
  - Port: 8022
  - Connection/Data: username: <LOGIN>
  - Connection/SSH/Tunnels: Port 1080, Dynamic

**OpenSSH ~/.ssh/config (Mac / Linux)**

- Host uni *
  - User: <LOGIN>
    - Port 8822
- Host uni_bastion1
  - Hostname bastion1.<domain>.com

**SOCKS LISTEN: localhost:1080**

SSH -D 1080 (SOCKS 5 Proxy)

\[.*<domain>-intern.com\]

\[gitlab.<domain>-intern.com\]

\[bastion1.<domain>.com\]
SSH Advanced Usage: SOCKS Proxy

- Host uni_*
  - Hostname: bastion1.<domain>.com
  - Port: 8822
  - Connection/Data: username: <LOGIN>
  - Connection/SSH/Tunnels: Port 1080, Dynamic

SSH -D 1080 (SOCKS 5 Proxy)

PuTTY / PLink / Pageant (Windows)

OpenSSH ~/.ssh/config (Mac / Linux)

SOCKS LISTEN: localhost:1080

https://gitlab.<domain>-intern.com

S. Varrette (PCOG Research unit)
SSH Advanced Usage: ProxyCommand

- `*.<domain>-intern.com`
- `gitlab`
- `bastion1.<domain>.com`
SSH Advanced Usage: ProxyCommand

**OpenSSH**
```
~/.ssh/config (Mac / Linux)
```
- Host: `uni_`
- User: `<LOGIN>`
- Port: 8822
- Hostname: `bastion1.<domain>.com`

**PutTY / PLink / Pageant (Windows)**
```
Session "uni_bastion1"
- Hostname: `bastion1.<domain>.com`
- Port: 8822
- Connection/Data: username: `<LOGIN>`
```

** bastion1.<domain>.com **

---

S. Varrette (PCOG Research unit)
SSH Advanced Usage: ProxyCommand

**OpenSSH**

- `~/.ssh/config` (Mac / Linux)
  - Host: uni_\*  
  - Hostname: bastion1.<domain>.com  
  - Port: 8022  
  - Connection/Data: username: <LOGIN>
  - Host: uni_bastion1  
  - Hostname: bastion1.<domain>.com  
  - Port: 8022  
  - Connection/Data: username: <LOGIN>
  - Connection/Proxy:
    - type: local
    - Proxy hostname: bastion1.<domain>.com  
    - Port: 8022  
    - Username: <LOGIN>
    - Local proxy command:
      `plink -load "uni_bastion1" -nc %host:%port`

**PuTTY / PLink / Pageant (Windows)**

- Session “uni_bastion1”  
  - Hostname: bastion1.<domain>.com  
  - Port: 8022  
  - Connection/Data: username: <LOGIN>
- Session “uni_gitlab”  
  - Hostname: gitlab.<domain>-intern.com  
  - Port: 8022  
  - Connection/Data: username: <LOGIN>
  - Connection/Proxy:
    - type: local
    - Proxy hostname: bastion1.<domain>.com  
    - Port: 8022  
    - Username: <LOGIN>
    - Local proxy command:
      `plink -load "uni_bastion1" -nc %host:%port`
SSH Advanced Usage: ProxyCommand

**OpenSSH**

```
~/.ssh/config  (Mac / Linux)

Host uni_*
  Hostname: bastion1.<domain>.com
  Port: 8822
  Connection/Data: username: <LOGIN>

Host uni_bastion1
  Hostname: bastion1.<domain>.com
  Port: 8822
  Connection/Data: username: <LOGIN>
  Connection/Proxy:
    - type: local
      - Proxy hostname: bastion1.<domain>.com
      - Port: 8822
      - Username: <LOGIN>
    - Local proxy command:
      plink -load "uni_bastion1" -nc %host:%port
```

**PuTTY / Plink / Pageant (Windows)**

```
Session "uni_bastion1"
  - Hostname: bastion1.<domain>.com
  - Port: 8822
  - Connection/Data: username: <LOGIN>

Session "uni_gitlab"
  - Hostname: gitlab.<domain>-intern.com
  - Port: 8822
  - Connection/Data: username: <LOGIN>
  - Connection/Proxy:
    - type: local
      - Proxy hostname: bastion1.<domain>.com
      - Port: 8822
      - Username: <LOGIN>
    - Local proxy command:
      plink -load "uni_bastion1" -nc %host:%port
```

**SSH + ProxyCommand + netcat**

```
*.<domain>-intern.com
```

```
bastion1.<domain>.com
```

```
SSH + ProxyCommand + netcat
```
SSH Advanced Usage: ProxyCommand

**OpenSSH**  
`.ssh/config` (Mac / Linux)  
- **Host uni_***  
  - Hostname: `bastion1.<domain>.com`  
  - Port: 8022  
  - Connection/Data: username: `<LOGIN>`  
  - Connection/Proxy:  
    - type: local  
    - Proxy hostname: `bastion1.<domain>.com`  
    - Port: 8022  
    - Username: `<LOGIN>`  
    - Local proxy command:  
      `plink -load "uni_bastion1" -nc %host:%port`  

**PuTTY / PLink / Pageant (Windows)**  
- Session “uni_bastion1”  
  - Hostname: `gitlab.<domain>-intern.com`  
  - Port: 8022  
  - Connection/Data: username: `<LOGIN>`  
  - Connection/Proxy:  
    - type: local  
    - Proxy hostname: `bastion1.<domain>.com`  
    - Port: 8022  
    - Username: `<LOGIN>`  
    - Local proxy command:  
      `plink -load "uni_bastion1" -nc %host:%port`
DSH – Distributed / Dancer’s Shell

http://www.netfort.gr.jp/~dancer/software/dsh.html.en

- SSH wrapper that allows to run commands over multiple machines.
  → Linux / Mac OS only

```
$> { apt-get | yum | brew } install dsh
# Installation
```

**Configuration:** in `~/.dsh/`

- `~/.dsh/dsh.conf`: main configuration file
- `~/.dsh/machines.list`: list of all nodes
- `~/.dsh/group/:` holds group definition

- `<name>` **Group** definition: `~/.dsh/group/<name>`:
  → simply list SSH shortnames (one name by line)

- Bash completion file for DSH:
  [https://gist.github.com/920433.git](https://gist.github.com/920433.git)
DSH configuration ~/.dsh/dsh.conf

############################################################
# ~/.dsh/dsh.conf
# Configuration file for dsh (Distributed / Dancer’s Shell).
# ‘man dsh.conf’ for details
############################################################
verbose = 0

remoteshell = ssh
showmachinenames = 1

# Specify 1 to make the shell wait for each individual invocation.
# See -c and -w option for dsh(1)
waitshell = 0  # whether to wait for execution

# Number of parallel connection to create at the same time.
#forklimit=8

remoteshellopt = -q
**DSH Basic Usage**

```bash
$> dsh [-c | -w] { -a | -g <group> | -m <hostname> } <command>
```

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-c</td>
<td>run the commands in parallel (default)</td>
</tr>
<tr>
<td>-w</td>
<td>run the commands in sequential</td>
</tr>
<tr>
<td>-a</td>
<td>run the command on all nodes listed in machines.list</td>
</tr>
<tr>
<td>-g &lt;group&gt;</td>
<td>restrict the commands to the hosts group &lt;group&gt;</td>
</tr>
<tr>
<td>-m &lt;hostname&gt;</td>
<td>run the command only on hostname</td>
</tr>
</tbody>
</table>

**FAQ:** sudo: sorry, you must have a tty to run sudo

→ requires to change the default configuration of sudo
→ Ex to **not** requiring a tty to launch a sudo command

Defaults:<login> !requiretty
Summary

1. Introduction
   - Agenda
   - Overview of managed IT Infrastructure

2. IT/Dev[op]s Army Knives Tools
   - SSH Secure Shell
   - PGP / GPG: Gnu Privacy Guard
   - Vagrant
   - Puppet
   - Ruby / Python / Markdown-based Documentations
   - Password Management

3. Research Computing Platform @ UL

4. Git[Lab] @ UL and VCS

5. Git Basics
   - Installing Git
   - Git theory
   - Basic Commands
   - Branching and Merging

6. Collaborating / Working together

7. Advanced Git Topics
   - Git Submodules
   - Rebasings
   - Using Git over Subversion Repository
   - More Cool stuff
GPG: Gnu Privacy Guard

- **GnuPG**: implementation of the OpenPGP standard aka RFC4880
  - Hybrid encryption framework based on Web of Trust
  - Mail | Document | Git commit... encryption / signature
GPG Setup (Mac OS)

- GPGTools Suite
  - GPG for Apple Mail and GPG Keychain
  - GPG Services and MacGPG
GPG Setup (Mac OS)

- **GPGTools Suite**
  - GPG for Apple Mail and GPG Keychain
  - GPG Services and MacGPG

![GPG Keychain Access screenshot](image)
GPG Tools Suite

- GPG for Apple Mail and GPG Keychain
- GPG Services and MacGPG
IT/Dev[op]s Army Knives Tools

GPG Setup (Mac OS)

- GPGTools Suite
  - GPG for Apple Mail and GPG Keychain
  - GPG Services and MacGPG

![Image of encrypted email]

Lukas Pitsch
To: hello@gpgtools.org
My first encrypted & signed message
Security:  ■ Encrypted, ⊳ Signed (lukele@gpgtools.org)

Yes, it's really this easy to send secure messages!
Just try it for yourself!

Best,

GPGTools
GPG Setup (Mac OS)

- GPGTools Suite
  - GPG for Apple Mail and GPG Keychain
  - GPG Services and MacGPG

![GPG setup example](image)
GPG Setup (Windows)

- **GPG4Win – Tutorial**
  - GnuPG, GnuPG for Outlook (GpgOL)
  - **Kleopatra + GNU Privacy Assistant (GPA)** (to be checked)
  - GPG Explorer eXtension (GpgEX)

- **[All OS] Thunderbird + Enigmail**
GPG Setup (Windows)

- **GPG4Win** – Tutorial
  - GnuPG, GnuPG for Outlook (GpgOL)
  - **Kleopatra + GNU Privacy Assistant (GPA)** (to be checked)
  - GPG Explorer eXtension (GpgEX)

- **[All OS]** Thunderbird + Enigmail
GPG Setup (Windows)

- **GPG4Win – Tutorial**
  - GnuPG, GnuPG for Outlook (GpgOL)
  - Kleopatra + GNU Privacy Assistant (GPA) (to be checked)
  - GPG Explorer eXtension (GpgEX)

- **[All OS] Thunderbird + Enigmail**
GPG Setup (Windows)

- **GPG4Win – Tutorial**
  - GnuPG, GnuPG for Outlook (GpgOL)
  - **Kleopatra + GNU Privacy Assistant (GPA)** (to be checked)
  - GPG Explorer eXtension (GpgEX)

- **[All OS]** Thunderbird + Enigmail
GPG CLI Usage

$> gpg --gen-key  # Generate your PGP key
$> gpg --list-keys [pattern]  # List available PGP key(s)
$> gpg --keyserver pgp.mit.edu --search-keys <pattern> # Search & Import
$> gpg --keyserver pgp.mit.edu --recv-keys <ID>  # Import
GPG CLI Usage

$> gpg --gen-key
# Generate your PGP key
$> gpg --list-keys [pattern]
# List available PGP key(s)
$> gpg --keyserver pgp.mit.edu --search-keys <pattern>
# Search & Import
$> gpg --keyserver pgp.mit.edu --recv-keys <ID>
# Import

- Send encrypted mails to user@domain.org ⇔ you trust his key
  → i.e. sign (after careful check) this key (using GPG Keychain / GPA)
$> \text{gpg} --\text{gen-key} \quad \# \text{Generate your PGP key}

$> \text{gpg} --\text{list-keys} \ [\text{pattern}] \quad \# \text{List available PGP key(s)}

$> \text{gpg} --\text{keyserver} \text{pgp.mit.edu} --\text{search-keys} \ [\text{pattern}] \quad \# \text{Search \& Import}

$> \text{gpg} --\text{keyserver} \text{pgp.mit.edu} --\text{recv-keys} \ [\text{ID}] \quad \# \text{Import}

- Send \textbf{encrypted} mails to user@domain.org ⇔ you \textbf{trust} his key
  → \textit{i.e. sign} (after careful check) this key (using GPG Keychain / GPA)

$> \text{gpg} [-K] --\text{fingerprint} \ [\text{mail}] \quad \# \text{Get (with -K) \& Check fingerprint}

$> \text{gpg} --\text{sign-key} --\text{ask-cert-level} \ [\text{ID}] \quad \# \text{Sign Key} \ [\text{ID}] \ \text{AFTER check}

$> \text{gpg} --\text{keyserver} \text{pgp.mit.edu} --\text{send-keys} \ [\text{ID}] \quad \# \text{Send back signed key}
GPG CLI Usage

$> gpg --encrypt [-r <recipient>] <file> # => <file>.gpg

**WARNING**: encryption does not delete the input (clear-text) file
GPG CLI Usage

$> gpg --encrypt [-r <recipient>] <file>    # => <file>.gpg

- **WARNING**: encryption does not delete the input (clear-text) file

$> gpg --decrypt <file>.gpg    # Decrypt PGP encrypted file
GPG CLI Usage

$> gpg --encrypt [-r <recipient>] <file>  # => <file>.gpg

**WARNING**: encryption does not delete the input (clear-text) file

$> gpg --decrypt <file>.gpg  # Decrypt PGP encrypted file

$> gpg --armor --detach-sign <file>  # Do signature <file>.asc
GPG CLI Usage

$> gpg --encrypt [-r <recipient>] <file>  # => <file>.gpg

**WARNING:** encryption does not delete the input (clear-text) file

$> gpg --decrypt <file>.gpg  # Decrypt PGP encrypted file

$> gpg --armor --detach-sign <file>  # Do signature <file>.asc

**GPG Keychain / Keyring**

→ Linux / Mac OS:  
~/ .gnupg/

← Windows:  
C:\\Documents and Settings\\<LOGIN>\\Application Data\\gnupg\"
Recall: Security = Noble Goal, yet...
Summary

1. Introduction
   - Agenda
   - Overview of managed IT Infrastructure

2. IT/Dev[op]s Army Knives Tools
   - SSH Secure Shell
   - PGP / GPG: Gnu Privacy Guard
   - Vagrant
   - Puppet
   - Ruby / Python / Markdown-based Documentations
   - Password Management

3. Research Computing Platform @ UL

4. Git[Lab] @ UL and VCS

5. Git Basics
   - Installing Git
   - Git theory
   - Basic Commands
   - Branching and Merging

6. Collaborating / Working together

7. Advanced Git Topics
   - Git Submodules
   - Rebasing
   - Using Git over Subversion Repository
   - More Cool stuff

S. Varrette (PCOG Research unit)
What is Vagrant?

http://vagrantup.com/

Development environments made easy.

Create and configure lightweight, reproducible, and portable development environments.

DOWNLOAD  GET STARTED
What is Vagrant?

Create and configure **lightweight, reproducible, and portable** development environments

- Command line tool
- Automates VM creation with
  - VirtualBox
  - VMWare etc.
- Integrates well with configuration management tools
  - Shell
  - Puppet etc.

Runs on Linux, Windows, MacOS
Why use Vagrant?

- Create new VMs quickly and easily: only one command!

  $$\text{vagrant up}$$

- Keep the number of VMs under control
  - All configuration in VagrantFile

- Reproducability
  - Identical environment in development and production

- Portability
  - avoid sharing 4 GB VM disks images
  - Vagrant Cloud to share your images

- Collaboration made easy:
  $$\text{git clone ...}$$
  $$\text{vagrant up}$$
Installation Notes: Mac OS

Best done using Homebrew and Cask

```
$> brew install caskroom/cask/brew-cask
$> brew cask install virtualbox       # install virtualbox
$> brew cask install vagrant
$> brew cask install vagrant-manager # see http://vagrantmanager.com/
```
Installation Notes: Windows / Linux

- Install Oracle Virtualbox
- Go on the Download Page
  → select the appropriate OS, in 64 bits versions

Notes for Windows users:
  → you will also need both PuTTY and PuTTYGen
  → Vagrant boxes are located in %userprofile%/.vagrant.d/boxes
  → To configure the appropriate Putty profile:
    ✓ run `vagrant ssh-config` to collect IP and port (after `vagrant up`)
    ✓ load %userprofile%/.vagrant.d/insecure_public_key
    ✓ Use Save Public Key to convert the OpenSSH key to PPK format
    ✓ Create the Putty profile accordingly (username: vagrant)
Minimal default setup

$> \textit{vagrant init [-m]} <\textit{user}>/<\textit{name}> \# \textit{setup vagrant cloud image}

- A Vagrantfile is configured
Minimal default setup

```
$> vagrant init [-m] <user>/<name>  # setup vagrant cloud image
```

- A Vagrantfile is configured

```
$> vagrant up  # boot the box(es) set in the Vagrantfile
```

- The base box is downloaded and stored locally
  - in ~/.vagrant.d/boxes/
- A new VM is created and configured with the base box as template
- The VM is booted and (eventually) provisioned
Minimal default setup

```bash
$> vagrant init [-m] <user>/<name>  # setup vagrant cloud image
```

- A Vagrantfile is configured

```bash
$> vagrant up  # boot the box(es) set in the Vagrantfile
```

- The base box is downloaded and stored locally
  → in ~/.vagrant.d/boxes/
- A new VM is created and configured with the base box as template
- The VM is booted and (eventually) provisioned

```bash
$> vagrant ssh  # connect inside it
```
Find a vagrant box

- Vagrant Cloud
- VagrantBox.es

https://vagrantcloud.com/
http://www.vagrantbox.es/
Find a vagrant box

- Vagrant Cloud
- VagrantBox.es

Your Turn!

```
$> vagrant init ubuntu/trusty64  # Ubuntu Server 14.04 LTS
$> vagrant up
$> vagrant ssh
```

<table>
<thead>
<tr>
<th>Box name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ubuntu/trusty64</td>
<td>Ubuntu Server 14.04 LTS</td>
</tr>
<tr>
<td>centos/7</td>
<td>CentOS Linux 7 x86_64</td>
</tr>
<tr>
<td>debian/jessie64</td>
<td>Vanilla Debian 8 “Jessie”</td>
</tr>
<tr>
<td>jhcook/osx-elcapitan-10.11</td>
<td>OS X 10.11 El Capitan</td>
</tr>
</tbody>
</table>

Once within the box:

- `/vagrant`: root directory hosting Vagrantfile
Configuring Vagrant

- Minimal Vagrantfile (Ruby syntax)

```ruby
VAGRANTFILE_API_VERSION = '2'

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
  config.vm.box = 'svarrette/centos-7-puppet'
  config.ssh.insert_key = false
end
```

- Configure Multiple box within the same Vagrantfile
  → See ULHPC/puppet-sysadmins/Vagrantfile
$>\text{vagrant status} \quad \# \text{State of the vagrant box(es)}
Once you have finished your work within a *running* box

- save the state for later with `vagrant halt`
- reset changes / tests / errors with `vagrant destroy`
- commit changes by generating a new version of the box
You might rely on **Falkor/vagrant-vms**

- use it at your own risks
- based on packer and veewee

```
$> git clone https://github.com/Falkor/vagrant-vms.git
$> cd vagrant-vms
$> gem install bundler && bundle install
$> rake setup
```
Vagrant Box Generation

- You might rely on Falkor/vagrant-vms
  → use it at your own risks
  → based on packer and veewee

```
$> git clone https://github.com/Falkor/vagrant-vms.git
$> cd vagrant-vms
$> gem install bundler && bundle install
$> rake setup
```

# initiate a template for a given Operating System:
$> rake packer:{Debian,CentOS,openSUSE,scientificlinux,ubuntu}:init

# Build a Vagrant box
$> rake packer:{Debian,CentOS,openSUSE,scientificlinux,ubuntu}:build

# If things goes fine:
$> vagrant box add packer/<os>-<version>-<arch>/<os>-<version>-<arch>.box
Vagrant Box Customization

- **Obj**: customize / specialize the configuration of a **running** box

This can be done in two ways:

1. use **provisioning** within the Vagrantfile (using puppet etc.)
2. re-package the box via **vagrant package**

```ruby
# (1) Vagrantfile with Puppet provisioning
Vagrant.configure(2) do |config|
  config.vm.box = 'svarrette/centos-7-puppet'
  config.vm.provision :puppet do |puppet|
    puppet.hiera_config_path = 'hieradata/hiera.yaml'
    puppet.working_directory = '/vagrant'
    puppet.manifests_path = 'manifests'
    puppet.module_path = 'modules'
    puppet.manifest_file = 'init.pp'
    puppet.options = ['-v', '--report', '--show_diff', '--pluginsync']
  end
end
```
Box Re-packaging (1/2)

- **WARNING**: ensure you **DO NOT** reset the (insecure) SSH key before `vagrant up`, use the following Vagrantfile configuration:
  ```ruby
  config.ssh.insert_key = false
  ```
- Zero out the free space to save space – run the following script:

  ```bash
  $> dd if=/dev/zero of=/EMPTY bs=1M
  $> rm -f /EMPTY
  ```

- Ensure Virtualbox Guest additions match using the `vbguest` plugin

  ```bash
  $> vagrant plugin install vagrant-vbguest
  $> vagrant vbguest --status
  GuestAdditions versions on your host (5.0.4) & guest (4.3.26) mismatch
  # Upgrade the GuestAdditions
  $> vagrant vbguest --do install --auto-reboot
  ```
# Locate the internal name of the running VM and repackage it

```bash
$ VBoxManage list runningvms
"vagrant-vms_default_1431034026308_70455" {...}

$ vagrant package \
   --base vagrant-vms_default_1431034026308_70455 \
   --output <os>-<version>-<arch>.box
```
# Locate the internal name of the running VM and repackage it

```
$> VBoxManage list runningvms
"vagrant-vms_default_1431034026308_70455" {...}
```

```
$> vagrant package \
    --base vagrant-vms_default_1431034026308_70455 \
    --output <os>-<version>-<arch>.box
```

Now you can upload the generated box on Vagrant Cloud.

- select ‘New version’, enter the new version number
- add a new box provider (Virtualbox)
- upload the generated box
Box Re-packaging (2/2)

# Locate the internal name of the running VM and repackage it

```
$> VBoxManage list runningvms
"vagrant-vms_default_1431034026308_70455" {...}
```

```
$> vagrant package \
   --base vagrant-vms_default_1431034026308_70455 \
   --output <os>-<version>-<arch>.box
```

Now you can upload the generated box on Vagrant Cloud.

- select ‘New version’, enter the new version number
- add a new box provider (Virtualbox)
- upload the generated box

Upon successful upload: release the uploaded box

- by default it is unreleased
- Now people using the <user>/<name> box will be notified of a pending update
Summary

1. Introduction
   Agenda
   Overview of managed IT Infrastructure

2. IT/Dev[op]s Army Knives Tools
   SSH Secure Shell
   PGP / GPG: Gnu Privacy Guard
   Vagrant
   Puppet
   Ruby / Python / Markdown-based Documentations
   Password Management

3. Research Computing Platform @ UL

4. Git[Lab] @ UL and VCS

5. Git Basics
   Installing Git
   Git theory
   Basic Commands
   Branching and Merging

6. Collaborating / Working together

7. Advanced Git Topics
   Git Submodules
   Rebasin
   Using Git over Subversion Repository
   More Cool stuff
Server/Service configuration by Puppet

- **IT Automation** for configuration management
  - idempotent
  - agent/master OR stand-alone architecture
  - cross-platform through Puppet’s Resource Abstraction Layer (RAL)
  - Git-based workflow
  - PKI-based security (X.509)

- **DevOps** tool of choice for configuration management
  - Declarative Domain Specific Language (DSL)
Server/Service configuration by Puppet

- **IT Automation** for configuration management
  - idempotent
  - agent/master OR stand-alone architecture
  - cross-platform through Puppet’s Resource Abstraction Layer (RAL)
  - Git-based workflow
  - PKI-based security (X.509)

- **DevOps** tool of choice for configuration management
  - Declarative Domain Specific Language (DSL)

Average server installation/configuration time: \( \approx 3-6 \text{ min} \)
Configuration Management advantages

- **Infrastructure as Code**: Track, Test, Deploy, Reproduce, Scale
  - Code commits log shows the *history of change* on the infrastructure

- **Reproducible setups**
  - Do once, repeat forever

- **Scale** quickly:
  - Done for one, use on many

- **Coherent** and consistent server setups

- **Aligned Environments** for devel, test, qa, prod nodes

Alternatives to Puppet: Chef, CFEngine, Salt, Ansible
## Software related to Puppet

<table>
<thead>
<tr>
<th>Tool</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Facter</td>
<td>Complementary tool to retrieve system’s data</td>
</tr>
<tr>
<td>MCollective</td>
<td>Infrastructure Orchestration framework</td>
</tr>
<tr>
<td>Hiera</td>
<td>Key-value lookup tool where Puppet data can be placed</td>
</tr>
<tr>
<td>PuppetDB</td>
<td>Stores all the data generated by Puppet</td>
</tr>
<tr>
<td>Puppet DashBoard</td>
<td>A Puppet Web frontend and External Node Classifier (ENC)</td>
</tr>
<tr>
<td>The Foreman</td>
<td>A well-known third party provisioning tool and Puppet ENC</td>
</tr>
<tr>
<td>Geppetto</td>
<td>A Puppet IDE based on Eclipse</td>
</tr>
</tbody>
</table>
IT/Dev[op]s Army Knives Tools

How Puppet work?

1. Define
   Define the desired state of each system using Puppet’s declarative language.

2. Simulate
   Puppet is able to simulate deployment to test the changes without disruption to the infrastructure.

3. Enforce
   Puppet will check the system and updates it to the desired state.

4. Report
   Gain insight into the infrastructure by tracking all changes applied and state of the nodes.
General Puppet Infrastructure

- Puppet Master
  - Client descriptions
  - MCollective / ActiveMQ or XMLRPC/REST over SSL
  - Files
  - Modules/Manifests
  - Certificate Authority
- Environments
  - testing
  - production
  - devel

Client Site A

- CentOS
- Puppet agent
- Puppet agent

- Apple
- Puppet agent
- Puppet agent
- Puppet agent

- Windows
- Puppet agent
- Puppet agent
- Puppet agent
- Puppet agent

- Debian
- Puppet agent

- Red Hat
- Puppet agent

- Ubuntu
- Puppet agent
Puppet Data Flow

1. Facts
   The node sends normalized data about itself to the Puppet Master.

2. Catalog
   Puppet uses the Facts to compile a Catalog that specifies how the node should be configured.

3. Report
   The node reports back to Puppet indicating the configuration is complete, which is visible in the Puppet Dashboard.

4. Report Collector
   (Puppet or 3rd party tool)
   Puppet’s open API can also send data to third party tools.
between 161s and 364s to completely bootstrap a virgin node
→ between 20s and 31s to later check/correct the config

Now proposed as an IT service to external consumers
Puppet Installation

- Debian, Ubuntu (available by default)

  ```
  $> apt-get install puppet       # On clients (nodes)
  $> apt-get install puppetmaster # On server (master)
  ```

- RedHat, Centos, Fedora
  
  → Add EPEL repository or RHN Extra channel

  ```
  $> rpm -ivh https://yum.puppetlabs.com/puppetlabs-release-el-<version>.noarch.rpm
  $> yum install puppet            # On clients (nodes)
  $> yum install puppet-server    # On server (master)
  ```

- Other OS:

  [https://docs.puppet.com/puppet/3.8/reference/pre_install.html](https://docs.puppet.com/puppet/3.8/reference/pre_install.html)
Puppet DSL

- A Declarative Domain Specific Language (DSL)
  - defines **STATES** (and **not** procedures)
- Puppet code is written in **manifests** `<file>.pp`
  - declare resources that affect elements of the system
    - each resource has a type (package, service, file, user, exec ...)
    - each resource has a uniq title
  - resources are grouped in **classes**
- Classes and configuration files are organized in **modules**
- **Example** of resources types:

```ruby
file { '/etc/motd':
  content => "Toto"
}

package { 'openssh':
  ensure => present,
}

service { 'httpd':
  ensure => running,
  enable => true,
}
```
**Containers** of different resources

Can have parameters since Puppet 2.6

```ruby
class mysql (
    $root_password = 'default_value',
    $port        = '3306',
) {
    package { 'mysql-server':
        ensure => present,
    }
    service { 'mysql':
        ensure => running,
    }
    [...]
}
```
To use a class previously defined, we declare it

“Old style” class declaration, without parameters:

```
include mysql
```

“New style” (from Puppet 2.6) with explicit parameters:

```
class { 'mysql':
    root_password => 'my_value',
    port => '3307',
}
```

A class is uniq to a given node
Puppet Defines

- Similar to parametrized classes ...
  \[ \rightarrow \ldots \text{ but can be used multiple times (with different titles).} \]

```ruby
# Definition of a define
define apache::virtualhost (
  $ensure = present,
  $template = 'apache/virtualhost.conf.erb',
  [...] ) {
  file { "ApacheVirtualHost_${name}":
    ensure => $ensure,
    content => template("${template}")},
}

# Declaration of a define:
apache::virtualhost { 'www.uni.lu':
  template => 'site/apache/www.uni.lu-erb'
}
```
Puppet Variables and Facts

- Can be defined in different places and by different actors:
  - by client nodes as facts
  - defined by users in Puppet code, on Hiera or in the ENC
  - built-in and be provided directly by Puppet

- Facts using `facter`:
  - runs on clients and collects facts that the server can use as variables

```bash
$ facter
architecture => x86_64
fqdn => toto.uni.lu
hostname => toto
kernel => Linux
memorytotal => 16.00 GB
netmask => 255.255.255.0
operatingsystem => Centos
operatingsystemrelease => 6.3
osfamily => RedHat
virtual => physical
[...]```
Puppet User Variables

- In Puppet manifests:

  ```
  $role = 'mail'

  $package = $::operatingsystem ? {
    /(?i:Ubuntu|Debian|Mint)/ => 'apache2',
    default => 'httpd',
  }
  ```

- In an External Node Classifier (ENC)
  - Commonly used ENC are Puppet Dashboard, the Foreman, Puppet Enterprise.
- In an Hiera backend

  ```
  $syslog_server = hiera(syslog_server)
  ```
Puppet Nodes

- A node is identified by the PuppetMaster by its **certname**
  - defaults to the node’s fqdn

```puppet
node 'web01' {
    include apache
}
```

```puppet
node /^www\d+$/ {
    include apache
}
```

- Nodes classification can be done by External Node Classifier (ENC)
  - Puppet Dashboard, The Foreman and Puppet Enterprise
- Nodes classification can be done also by Hiera
  - In /etc/puppet/manifests/site.pp

```puppet
hiera_include('classes')
```
Puppet Operational modes

- **Masterless** - apply Puppet manifests directly on the target system.
  - No need of a complete client-server infrastructure.
  - Have to distribute manifests and modules to the managed nodes.

```
puppet apply --modulepath /modules/ /manifests/file.pp
```
Puppet Operational modes

**Masterless** - apply Puppet manifests directly on the target system.
- No need of a complete client-server infrastructure.
- Have to distribute manifests and modules to the managed nodes.

```bash
cpyuppet apply --modulepath /modules/ /manifests/file.pp
```

**Master / Client** Setup
- server (running as puppet) listening on 8140 on the Puppet Master
- client (running as root) on each managed node.
  - Run as a service (default), via cron (with random delays), manually or via MCollective
- Client and Server have to share SSL certificates
  - certificates must be signed by the Master CA

```bash
cpyuppet agent --test [--noop] [--environment <environment>]
```
Components of a Puppet architecture

**Tasks to be deal with:**
- definition of the classes to be included in each node
- definition of the parameters to use for each node
- definition of the configuration files provided to the nodes

**Components**
- Master, CA, and agents
- (optional) ENC - External Node Classifier
- (optional) Dap/IPA backend
- Hiera - Data key-value backend
- Public modules - Public shared modules
- Site modules - Local custom modules
**ULHPC Puppet Infrastructure**

**TOTAL resources managed**
- 191 servers (130 VMs)
- 488 computing nodes (5196 cores)
- 5.35 PB raw shared storage (NFS / GPFS / Lustre)
- 4 system administrators / 2 sites

**Managed hosts**
- LCSB site (Belval): 35 hosts
- LCSB site (Belval) puppet agent
- Puppet Master LCSB
- Puppet (Root) CA

- gaia cluster (Belval): 57+17 hosts
- gaia cluster (Belval) puppet agent
- Puppet Master gaia cluster
- Puppet (Root) CA

- chaos cluster (Kirchberg): 14 hosts
- chaos cluster (Kirchberg) puppet agent
- Puppet Master chaos cluster
- Puppet (Root) CA

**UL HPC Platform**
- Computing nodes (271+98+14)
- Managed computing nodes: 454 (4830 cores)
  (+38 [Grid5000 Luxembourg])

**Puppet Infrastructure**
- Puppet (Root) CA
- Managed hosts: 160 (+31 Grid5000 Luxembourg)
- Puppet Master LCSB
- Puppet Master gaia cluster
- Puppet Master chaos cluster
- Puppet (Root) CA

**Modules/Manifests**
- Root CA
- Puppet Master LCSB
- Puppet Master gaia cluster
- Puppet Master chaos cluster

**Environments**
- Files
- testing
- production
- devel

**Managed resources**
- 191 servers (130 VMs)
- 488 computing nodes (5196 cores)
- 5.35 PB raw shared storage (NFS / GPFS / Lustre)
- 4 system administrators / 2 sites

**S. Varrette (PCOG Research unit)**

**IT/Dev[op]s Army Knives Tools for the researcher**
Summary

1. Introduction
   - Agenda
   - Overview of managed IT Infrastructure

2. IT/Dev[op]s Army Knives Tools
   - SSH Secure Shell
   - PGP / GPG: Gnu Privacy Guard
   - Vagrant
   - Puppet
   - Ruby / Python / Markdown-based Documentations
   - Password Management

3. Research Computing Platform @ UL

4. Git[Lab] @ UL and VCS

5. Git Basics
   - Installing Git
   - Git theory
   - Basic Commands
   - Branching and Merging

6. Collaborating / Working together

7. Advanced Git Topics
   - Git Submodules
   - Rebasings
   - Using Git over Subversion Repository
   - More Cool stuff

S. Varrette (PCOG Research unit)
IT/Dev[op]s Army Knives Tools

Ruby / RVM / Bundler

- Bring the flexibility of Rakefile (Makefile + Ruby)
- Bundler: **reproducible** running environment **across** developpers
  - easy configuration through Gemfile[.lock] + bundle command
- RVM: sandboxed environment per project (alternative: rbenv)
  - easy configuration through .ruby-{version,gemset} files
Bring the flexibility of Rakefile (Makefile + Ruby)

- **Bundler**: reproducible running environment across developpers
  - easy configuration through Gemfile[.lock] + bundle command
- **RVM**: sandboxed environment per project (alternative: rbenv)
  - easy configuration through .ruby-{:version,gemset} files

**Typical setup of a freshly cloned project:**

```bash
$> gem install bundler  # assuming it is not yet available
$> bundle             # clone ruby deps/env as defined in Gemfile
$> rake -T            # To list the available tasks
```
Ruby / RVM / Bundler

- Bring the flexibility of Rakefile (Makefile + Ruby)
- Bundler: **reproducible** running environment **across** developers
  - easy configuration through Gemfile[.lock] + bundle command
- RVM: sandboxed environment per project (**alternative**: rbenv)
  - easy configuration through .ruby-{version,gemset} files

**Typical setup of a freshly cloned project:**

```bash
$> gem install bundler  # assuming it is not yet available
$> bundle             # clone ruby deps/env as defined in Gemfile*
$> rake -T            # To list the available tasks
```

**Recommended Gems**

- falkorlib, rake, bundler, git_remote_branch
**Python / Pip**

- **pip**: Python package manager
  - “nice” python packages: mkdocs...
  - Windows: install via Chocolatey

```
$ pip install <package>  # install <package>
```
**Python / Pip**

- **pip**: Python package manager
  - “nice” python packages: mkdocs...
  - Windows: install via Chocolatey

```
$> pip install <package>  # install <package>
```

```
$> pip install -U pip       # upgrade on Linux/Mac OS
```
Python / Pip

- **pip**: Python package manager
  - “nice” python packages: mkdocs...
  - Windows: install via Chocolatey

```
$> pip install <package>  # install <package>
```

```
$> pip install -U pip  # upgrade on Linux/Mac OS
```

- Dump python environment to a requirements file

```
$> pip freeze -l > requirements.txt  # as Ruby Gemfiles
```
**IT/Dev[op]s Army Knives Tools**

**Pyenv / VirtualEnv / Autoenv**

- **pyenv**: ≃ RVM/rbenv for Python
- **virtualenv**: ≃ RVM Gemset
- **(optional) autoenv**
  - Directory-based shell environments
  - Easy config through .env file. **Ex:**

```bash
# (rootdir)/.env : autoenv configuration file
pyversion='head .python-version'
pvenv='head .python-virtualenv'

pyenv virtualenv --force --quiet ${pyversion} ${pvenv}-${pyversion}
# activate it
pyenv activate ${pvenv}-${pyversion}
```
Privileged Markdown-based documentation

- easy to track over Git (text files, not Word/RFT etc.)
- easy to export to any format using pandoc / multimarkdown
- focus on writing, viewers for all platform
  
  - Mac OS: MOU, Marked 2
  - Linux: Remarkable, Retext
  - Windows: MarkdownPad, Remarkable

Git-based Markdown Wiki

- Gollum, as embedded in GitLab
- Mkdocs
Gollum / MkDocs

- **Advantage:** possibility to serve the HTML *locally*
  - **Gollum:** `gollum` (from root directory) → [http://localhost:4567](http://localhost:4567)
  - **Mkdocs:** `mkdocs serve` (from root directory) → [http://localhost:8000](http://localhost:8000)
Gollum / MkDocs

- Advantage: possibility to serve the HTML **locally**
  - Gollum: `gollum` (from root directory) → http://localhost:4567
  - Mkdocs: `mkdocs serve` (from root directory) → http://localhost:8000

```bash
$> mkdocs new  # initialize 'mkdocs.yml' and docs/ directory
```
Gollum / MkDocs

- **Advantage:** possibility to serve the HTML **locally**
  - Gollum: gollum (from root directory)
  - Mkdocs: mkdocs serve (from root directory)

```bash
$> mkdocs new  # initialize 'mkdocs.yml' and docs/ directory
```

```
# mkdocs.yml -- MkDocs configuration, all *.md files relative to docs/

site_name: UL HPC Developpers Documentaion

pages:
- Home:  'index.md'
- Tools:
  - SSH:  'tools/ssh.md'
  - Git:  'tools/git.md'
- Configuration:
  - CA Certificates:  'config/certificates/README.md'

theme: readthedocs
```
Summary

1 Introduction
   Agenda
   Overview of managed IT Infrastructure

2 IT/Dev[op]s Army Knives Tools
   SSH Secure Shell
   PGP / GPG: Gnu Privacy Guard
   Vagrant
   Puppet
   Ruby / Python / Markdown-based Documentations
   Password Management

3 Research Computing Platform @ UL

4 Git[Lab] @ UL and VCS

5 Git Basics
   Installing Git
   Git theory
   Basic Commands
   Branching and Merging

6 Collaborating / Working together

7 Advanced Git Topics
   Git Submodules
   Rebasings
   Using Git over Subversion Repository
   More Cool stuff
Password Management

Traditional [Strong] Password policy

- ≥ 15 characters, including digits, special chars (#, &, @, $ etc.)
  - mix upper/lower case
- avoid matching dictionary/personal/company/dates info
- renew periodically, typically after 180 days.

Build by selecting words / sentence easy to remember
  - combine them to respect the above rules
Stanford Password Policy

https://itservices.stanford.edu/service/accounts/passwords/quickguide

Which Characters Are Required in My Password?

Hint: it depends on password length!

8-11: requires mixed case letters, numbers, and symbols
12-15: requires mixed case letters and numbers
16-19: requires mixed case letters
20+: any characters you like

Passwords must be at least 8 characters.

Passwords over 20 characters are the gold standard and offer the most protection.
Password Manager

- Ensure a safe and **secure** way to store/organize passwords
  - privilege **random & unique** passwords **everywhere**
  - ideally: cross-platform applications, with browser integration
- encrypted back-end/vault, eventually shared over Cloud storage
  - Dropbox, iCLoud, S3, OneDrive...
Password Manager

- Ensure a safe and secure way to store/organize passwords
  - privilege random & unique passwords everywhere
  - ideally: cross-platform applications, with browser integration
- encrypted back-end/vault, eventually shared over Cloud storage
  - Dropbox, iCloud, S3, OneDrive...

Open-Source / Cloud-based

- KeePassX
- LastPass
Password Manager

- Ensure a safe and **secure** way to store/organize passwords
  - privilege **random & unique** passwords **everywhere**
  - ideally: cross-platform applications, with browser integration
- encrypted back-end/vault, eventually shared over Cloud storage
  - Dropbox, iCloud, S3, OneDrive...

**Open-Source** / **Cloud-based**

- KeePassX
- LastPass

**Commercial**

- 1Password
- dashlane
pass: the standard Unix password manager

- stores passwords as encrypted files – default: ~/.password-store/
- cross-platform GUI clients, incl. iOS/Android / Pass4Win
- multiple recipient can share a sub-directory

Installation: \{ brew | yum | apt-get \} install pass

```bash
$> pass init <ID> && pass git init # Create the store over git
```
**GPG+Git Password Management: pass**

- **pass**: the standard Unix password manager
  - stores passwords as encrypted files – default: `~/.password-store/`
  - cross-platform GUI clients, incl. iOS/Android / Pass4Win
  - multiple recipient can share a sub-directory

**Installation:**

```bash
$> pass init <ID> && pass git init  # Create the store over git
```

```bash
$> pass insert <domain>/<name>  # store <domain>/<name>.gpg
```
GPG+Git Password Management: pass

- **pass**: the standard Unix password manager
  - stores passwords as encrypted files – default: ~/.password-store/
  - cross-platform GUI clients, incl. iOS/Android / Pass4Win
  - multiple recipient can share a sub-directory

Installation: \{ brew | yum | apt-get \} install pass

```bash
$/> pass init <ID> && pass git init # Create the store over git
$/> pass insert <domain>/<name>   # store <domain>/<name>.gpg
$/> pass [<domain>/<name>]        # list / retrieve password <name>
```
Summary

1. Introduction
   - Agenda
   - Overview of managed IT Infrastructure

2. IT/Dev[op]s Army Knives Tools
   - SSH Secure Shell
   - PGP / GPG: Gnu Privacy Guard
   - Vagrant
   - Puppet
   - Ruby / Python / Markdown-based Documentations
   - Password Management

3. Research Computing Platform @ UL

4. Git[Lab] @ UL and VCS

5. Git Basics
   - Installing Git
   - Git theory
   - Basic Commands
   - Branching and Merging

6. Collaborating / Working together

7. Advanced Git Topics
   - Git Submodules
   - Rebasin
   - Using Git over Subversion Repository
   - More Cool stuff

S. Varrette (PCOG Research unit)

IT/Dev[op]s Army Knives Tools for the researcher
Key numbers

- 344 users
- 98 servers
- 492 nodes
- 5300 cores
- 85.543 TFlops
- 5354.4 TB
- 4 sysadmins
- 2 sites

Kirchberg
Belval

http://hpc.uni.lu

S. Varrette (PCOG Research unit)
Evolution of registered users with active accounts within UL internal clusters

- LCSB (Bio-Medicine)
- URPM (Physics and Material Sciences)
- FDEF (Law, Economics and Finance)
- RUES (Engineering Science)
- SnT (Security and Trust)
- CSC (Computer Science and Communications)
- LSRU (Life Sciences)
- Bachelor and Master students
- Others

Number of users

Enables & accelerates scientific discovery and innovation
Largest facility in Luxembourg (after GoodYear R&D Center)

<table>
<thead>
<tr>
<th>Country</th>
<th>Name/Institute</th>
<th>#Cores</th>
<th>$R_{\text{peak}}$</th>
<th>Storage</th>
<th>Manpower</th>
</tr>
</thead>
<tbody>
<tr>
<td>Luxembourg</td>
<td>UL</td>
<td>5300</td>
<td>85.543</td>
<td>5354.4</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>CRP GL</td>
<td>800</td>
<td>6.21</td>
<td>144</td>
<td>1.5</td>
</tr>
<tr>
<td>France</td>
<td>TGCC Curie, CEA</td>
<td>77184</td>
<td>1667.2</td>
<td>5000</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>LORIA, Nancy</td>
<td>3724</td>
<td>29.79</td>
<td>82</td>
<td>5.05</td>
</tr>
<tr>
<td></td>
<td>ROMEO, UCR, Reims</td>
<td>564</td>
<td>4.128</td>
<td>15</td>
<td>2</td>
</tr>
<tr>
<td>Germany</td>
<td>Juqueen, Juelich</td>
<td>393216</td>
<td>5033.2</td>
<td>448</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>MPI, RZG</td>
<td>2556</td>
<td>14.1</td>
<td>n/a</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>URZ, (bwGrid),Heidelberg</td>
<td>1140</td>
<td>10.125</td>
<td>32</td>
<td>9</td>
</tr>
<tr>
<td>Belgium</td>
<td>UGent, VCS</td>
<td>4320</td>
<td>54.541</td>
<td>82</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>CECI, UMons/UCL</td>
<td>2576</td>
<td>25.108</td>
<td>156</td>
<td>&gt; 4</td>
</tr>
<tr>
<td>UK</td>
<td>Darwin, Cambridge Univ</td>
<td>9728</td>
<td>202.3</td>
<td>20</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>Legion, UCLondon</td>
<td>5632</td>
<td>45.056</td>
<td>192</td>
<td>6</td>
</tr>
<tr>
<td>Spain</td>
<td>MareNostrum, BCS</td>
<td>33664</td>
<td>700.2</td>
<td>1900</td>
<td>14</td>
</tr>
</tbody>
</table>

S. Varrette (PCOG Research unit) IT/Dev[op]s Army Knives Tools for the researcher
Horizontal HPC & storage services

- for the three UL Faculties and their Research Units
- for the two UL Inter-disciplinary Centres
  - LCSB, SnT
- ... and their external partners
- on UL strategic research priorities
  - computational sciences
  - systems biomedicine
  - security, reliability and trust
  - finance
UL HPC Services

**Horizontal HPC & storage services**

- for the three UL Faculties and their Research Units
- for the two UL Inter-disciplinary Centres
  - LCSB, SnT
- ... and their external partners
- on UL strategic research priorities
  - computational sciences
  - systems biomedicine
  - security, reliability and trust
  - finance

**Complementary research related services**

- On demand VM hosting for development, frontends, etc.
- Project management & collaboration (GForge, GitLab...)
- Cloud storage (OwnCloud) ... and many others!

Total: 80 servers
Long-running collaboration system, featuring:

- static web hosting for projects
- Git or Subversion repositories etc.

Get an account / information: [https://helpdesk.gforge.uni.lu/](https://helpdesk.gforge.uni.lu/)

Open to anybody **but** separate authentication base
Gitlab @ Uni.lu – https://gitlab.uni.lu

- Similar to Github
  - advanced Git repository management
  - ... incl. private projects
- Open to _UL staff__ with an HPC account
• Web-based platform
• Simplified interface to many popular **bioinformatics tools**
  \[\rightarrow\] and generation of **reproducible workflows**.

---

**Galaxy** – [http://galaxy-server.uni.lu](http://galaxy-server.uni.lu)
Git[Lab] @ UL and VCS

Summary

1 Introduction
   Agenda
   Overview of managed IT Infrastructure

2 IT/Dev[op]s Army Knives Tools
   SSH Secure Shell
   PGP / GPG: Gnu Privacy Guard
   Vagrant
   Puppet
   Ruby / Python / Markdown-based Documentations
   Password Management

3 Research Computing Platform @ UL

4 Git[Lab] @ UL and VCS

5 Git[Lab] Around You
   About Version Control System (VCS)

6 Git Basics
   Installing Git
   Git theory
   Basic Commands
   Branching and Merging

7 Collaborating / Working together

8 Advanced Git Topics
   Git Submodules
   Rebasings
   Using Git over Subversion Repository
   More Cool stuff

S. Varrette (PCOG Research unit)
Git[Lab] @ UL and VCS

Summary

1. Introduction
   Agenda
   Overview of managed IT Infrastructure

2. IT/Dev[op]s Army Knives Tools
   SSH Secure Shell
   PGP / GPG: Gnu Privacy Guard
   Vagrant
   Puppet
   Ruby / Python / Markdown-based Documentations
   Password Management

3. Research Computing Platform @ UL

4. Git[Lab] @ UL and VCS

5. Git[Lab] Around You
   About Version Control System (VCS)

6. Git Basics
   Installing Git
   Git theory
   Basic Commands
   Branching and Merging

7. Collaborating / Working together

8. Advanced Git Topics
   Git Submodules
   Rebasings
   Using Git over Subversion Repository
   More Cool stuff

S. Varrette (PCOG Research unit)
What Git will now mean to you...
(Reference) web-based Git repository hosting service

1. Set up Git
2. Create Repository
3. Fork repository
4. Work together

S. Varrette (PCOG Research unit)
Git is a **free and open source** distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Git is **easy to learn** and has a **tiny footprint with lightning fast performance**. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like **cheap local branching**, convenient **staging areas**, and **multiple workflows**.

**Learn Git in your browser for free with Try Git.**

---

**About**

The advantages of Git compared to other source control systems.

**Documentation**

Command reference pages, Pro Git book content, videos and other material.

**Downloads**

GUI clients and binary releases for all major platforms.

**Community**

Get involved! Bug reporting, mailing list, chat, development and more.

**Latest source Release 2.4.3**

Release Notes (2015-06-05)

Downloads for Mac

---

**Pro Git** by Scott Chacon and Ben Straub is available to [read online for free](https://git-scm.com/download/progit). Dead tree versions are available on [Amazon.com](https://www.amazon.com).

**Companies & Projects Using Git**

- Google
- Facebook
- Microsoft
- Twitter
- LinkedIn
- Netflix
- Apple
- Amazon
- Starbucks
- Postmates
- Amazon Web Services

---

S. Varrette (PCOG Research unit)  
IT/Dev[op]s Army Knives Tools for the researcher
git - the simple guide

just a simple guide for getting started with git. no deep shit ;)

by Roger Dudler

credits to @tnico, @fhd and Namics

this guide in deutsch, español, français, indonesian, italiano, nederlands, polski, português, русский, тürkçe,

한국어, 日本語, 中文, 한국어 Vietnamese

please report issues on github

---

Are You a Front-End Developer?
by Roger Dudler, Author of the Git Simple Guide

Try Frontify

Now Free with Github Integration

---

want a simple but powerful git client for your mac?
Atlassian Tutorials

https://www.atlassian.com/git/tutorials/

Become a git guru.

Getting Started
- Setting up a repository
- Saving changes
- Inspecting a repository
- Viewing old commits
- Undoing Changes
- Rewriting history

Collaborating
- Syncing
- Making a Pull Request
- Using Branches
- Comparing Workflows

Migrating to Git
- Migrate to Git from SVN
- Prepare
- Convert
- Synchronize
- Share
- Migrate

Advanced Tips
- Advanced Git Tutorials
- Merging vs. Rebasin
- Reset, Checkout, and Revert
- Advanced Git log
- Git Hooks
- Refs and the Reflog
Open-Source Book on Git by S. Chacon and B. Straub

- Sources (on Github)
- Online Reading – PDF

See also Git Internal, also by S. Chacon

**Note**: Most images of this talk comes from this book

- more precisely the first edition
Gitlab @ Uni.lu

- Gitlab \(\simeq\) Github Clone, for deployment on internal servers
  - web-based Git repository manager, wiki & issue tracking
  - GitLab CI for continuous integration and delivery.

- Open to **UL staff** with an **HPC account**
Gitlab Features

- Activity Stream

Dashboard

- Push events
- Merge events
- Comments
- Team

Marc Radulescu commented on commit 1c5cf4d0 at GitLab.com / www... less than a minute ago
@sysses @Haydn FYI I have fixed the mailchimp link and put the sales sheet item back where it belonged with: https://gitlab.com/gitlab-com/www-gitla...

Marc Radulescu pushed to branch master at GitLab.com / www-gitla...
e92b3717 fix typo

Marc Radulescu pushed to branch master at GitLab.com / www-gitla...
1c5cf4d0 Fix Mailchimp link

Sytse Sijbrandij commented on issue #1517 at GitLab.org / GitLab Co...

Marin Jankovski commented on issue #674 at GitLab.org / omnibus-gitla...

Marin Jankovski closed issue #674 at GitLab.org / omnibus-gitlab
Compile error in default.rb.

Jacob Vosmaer commented on issue #681 at GitLab.org / omnibus-gitla...

However, this doesn’t work well for upgrades where gitlab.rb is already generated...
Gitlab Features

- File Browser

![GitLab File Browser](image)
Gitlab Features

- **Git/Markdown powered Wiki**

---

Awesome! You’re about to become a GitLab developer! Make sure you’ve checked out our [handbook] beforehand, so you get a feeling of how we work at GitLab. Below you’ll find everything you need to start developing. If something is missing, add it (as goes with everything at GitLab)!

**GitLab instances**

We have two GitLab instances that we use primarily:

- **dev.gitlab.org**
  - This server is only accessible to people from GitLab the company.
  - This is the instance we use for customers development.
  - In addition, all our internal (company) issues are found here as well.
  - This server is updated from master every night, so we quickly see if we broke something.
  - Often referred to as **dev**.

- **GitLab.com**
  - This is the SaaS of GitLab. Everyone can host their repository for free here and this is where the majority of open source contributions come in. If you can, do your development and push your open source here.
Gitlab Features

- Powerful Code Review
Gitlab Features

- Issue Management

![GitHub Issue Management](image)

- Milestone #60 expires at Sep 22, 2015
- Progress: 1 closed – 18 open 5% complete

- Unstarted Issues
  - #2517 Online editor should not remove newline at the end of the file
  - #2489 Add a backup option to dump only objects from a specific Postgres 'schema'
  - #2487 Uploaded images don't show anymore when project is moved or path is changed
  - #2482 User search feature in admin area does not respect filters

- Ongoing Issues
  - Drag and drop available

- Completed Issues
  - #2478 Propose to create merge request when file was committed via web editor
  - Drag and drop available
Gitlab Features

- Code Snippets

![Gitlab Snippets](image-url)

```php
// local_config.php

<?php
if (!defined('BASEPATH')) exit('No direct script access allowed');

/* START OF LOCAL CONFIGURATION FILE */

/*
# CONFIG
# DATABASE
# BASE URL
# USER NAME
# PASSWORD
*/

$base_url = 'http://192.168.33.11/RUSI';

/*
# BASE URL
*/

$hostname = Hostname;
$database = Default DB;
$username = MySql username;
$password = MySql Password;
```
Gitlab Features

- Web/Service Hooks

Web hooks can be used for binding events when something is happening within the project.

- URL: http://example.com/trigger-ci.json
- Trigger:
  - Push events: This URL will be triggered by a push to the repository
  - Tag push events: This URL will be triggered when a new tag is pushed to the repository
  - Comments: This URL will be triggered when someone adds a comment
  - Issues events: This URL will be triggered when an issue is created
  - Merge Request events: This URL will be triggered when a merge request is created

Add Web Hook
Summary

1 Introduction
   Agenda
   Overview of managed IT Infrastructure

2 IT/Dev[op]s Army Knives Tools
   SSH Secure Shell
   PGP / GPG: Gnu Privacy Guard
   Vagrant
   Puppet
   Ruby / Python / Markdown-based Documentations
   Password Management

3 Research Computing Platform @ UL

4 Git[Lab] @ UL and VCS

5 Git[Lab] Around You
   About Version Control System (VCS)

6 Git Basics
   Installing Git
   Git theory
   Basic Commands
   Branching and Merging

7 Collaborating / Working together

8 Advanced Git Topics
   Git Submodules
   Rebasining
   Using Git over Subversion Repository
   More Cool stuff

S. Varrette (PCOG Research unit)
Why use Version Control?

- Version Control = Revision Control = Source Control
  - lets you track your files over time.
- you probably cooked up your own!
  - ever get files like main-v2.tex, CORE-proposal.doc.old or 2015-03-cv.pdf?
Why use Version Control?

- **Version Control = Revision Control = Source Control**
  - lets you track your files over time.
- you probably cooked up your own!
  - ever get files like main-v2.tex, CORE-proposal.doc.old or 2015-03-cv.pdf?

**Version Control System (VCS)**

- Integrated fool-proof framework for:
  - Backup and Restore
  - Synchronization / Collaborating
  - Short and long-term undo / Tracking changes
  - Sandboxing
Typical VCS Workflow

Basic Checkins

Checkout and Edit
Typical VCS Workflow

**Branching**

- Milk Eggs Soup
- Milk Eggs Soup Rice

New Features

**Merging**

- Milk Eggs Soup +Rice
- Milk Eggs Soup Rice

New Features

S. Varrette (PCOG Research unit)
Local VCS – RCS, Mac OS Versions

- Checkout
- Version Database
  - Version 3
  - Version 2
  - Version 1

Local Computer

S. Varrette  (PCOG Research unit)
Centralized VCS – CVS, SVN

- CVS
- SVN

Central VCS Server

Version Database

- Version 3
- Version 2
- Version 1

Computer A

Checkout

File
Centralized VCS – CVS, SVN
Everybody has the full history of commits
Git[Lab] @ UL and VCS

Tracking changes (most VCS)

Checkins over Time

- C1
- file A
- file B
- file C

S. Varrette (PCOG Research unit)
Tracking changes (most VCS)

Checkins over Time

C1 -> C2

file A -> Δ1

file B

file C -> Δ1
Tracking changes (most VCS)

Checkins over Time

- C1
- C2
- C3

file A → Δ1

file B

file C → Δ1 → Δ2
Tracking changes (most VCS)

Checkins over Time

file A → Δ1 → Δ2
file B → Δ1
file C → Δ1 → Δ2
Tracking changes (most VCS)

Checkins over Time

- C1
- C2
- C3
- C4
- C5

File A:
- Δ1
- Δ2

File B:
- Δ1
- Δ2

File C:
- Δ1
- Δ2
- Δ3

S. Varrette (PCOG Research unit)
Git[Lab] @ UL and VCS

Tracking changes (most VCS)

Checkins over Time

delta storage

file A

Δ1

Δ2

file B

Δ1

Δ2

file C

Δ1

Δ2

Δ3

S. Varrette (PCOG Research unit)
Tracking changes (Git)

- Checkins over Time: C1, C2, C3, C4, C5
- Delta storage:
  - File A: $\Delta_1 \rightarrow \Delta_2$
  - File B: $\Delta_1 \rightarrow \Delta_2$
  - File C: $\Delta_1 \rightarrow \Delta_2 \rightarrow \Delta_3$
- Snapshot (DAG) storage

S. Varrette (PCOG Research unit)

IT/Dev[op]s Army Knives Tools for the researcher
Tracking changes (Git)

Checkins over Time

file A → Δ1 → Δ2

file B → Δ1 → Δ2

file C → Δ1 → Δ2 → Δ3

delta storage

snapshot (DAG) storage

S. Varrette (PCOG Research unit)
Tracking changes (Git)

Checkins over Time

delta storage

snapshot (DAG) storage

file A → Δ1 → Δ2
file B → Δ1 → Δ2
file C → Δ1 → Δ2 → Δ3

Checkins over Time

C1, C2, C3, C4, C5

Δ1, Δ2, Δ3

A → A1
B → B
C → C1

S. Varrette (PCOG Research unit)
Tracking changes (Git)

Checkins over Time

delta storage

snapshot (DAG) storage

file A  \(\Delta 1\)  \(\Delta 2\)

file B  \(\Delta 1\)  \(\Delta 2\)

file C  \(\Delta 1\)  \(\Delta 2\)  \(\Delta 3\)

Checkins over Time
Tracking changes (Git)

Checkins over Time

delta storage

snapshot (DAG) storage

file A

Δ1

Δ2

file B

Δ1

Δ2

file C

Δ1

Δ2

Δ3

Checkins over Time

S. Varrette (PCOG Research unit)
Tracking changes (Git)

Checkins over Time

- C1
- C2
- C3
- C4
- C5

Delta storage

- File A: \( \Delta 1 \) → \( \Delta 2 \)
- File B: \( \Delta 1 \) → \( \Delta 2 \)
- File C: \( \Delta 1 \) → \( \Delta 2 \) → \( \Delta 3 \)

Snapshot (DAG) storage

- C1
- C2
- C3
- A
- A1
- B
- C
- C1
- C2
Tracking changes (Git)

Checkins over Time

- C1
- C2
- C3
- C4
- C5

delta storage

- file A
  - Δ1
  - Δ2
- file B
  - Δ1
  - Δ2
- file C
  - Δ1
  - Δ2
  - Δ3

Checkins over Time

- C1
- C2
- C3
- C4

snapshot (DAG) storage

- A
  - A1
  - A2
- B
  - B
  - B1
- C
  - C1
  - C2
Tracking changes (Git)

Checkins over Time

delta storage

file A

Δ1

file B

Δ1

file C

Δ1

Δ2

Checkins over Time

snapshot (DAG) storage

C1

A

B

C

C2

A1

B

C1

C3

A1

B

C2

C4

A2

B1

C2

C5

Δ2

Δ3

98 / 171

S. Varrette (PCOG Research unit)

IT/Dev[op]s Army Knives Tools for the researcher
Galaxy @ UL and VCS

Tracking changes (Git)

Checkins over Time

delta storage

file A \( \Delta_1 \)
file B \( \Delta_1 \)
file C \( \Delta_1 \)

Checkins over Time

snapshot (DAG) storage

file A

file B

file C

S. Varrette (PCOG Research unit)

IT/Dev[op]s Army Knives Tools for the researcher
Tracking changes (Git)

Checkins over Time

delta storage

snapshot (DAG) storage

file A

Δ1

file B

Δ1

file C

Δ1

Δ2

Δ3

Checkins over Time

delta storage

snapshot (DAG) storage

file A

Δ1

file B

Δ1

file C

Δ1

Δ2

Δ3

S. Varrette (PCOG Research unit)
VCS Taxonomy

- **delta storage**
  - local
  - centralized
  - distributed
  - rcs
  - cvs
  - mercurial
  - Subversion
  - Mac OS File Versions

- **snapshot (DAG) storage**
  - local
  - centralized
  - distributed
  - cp -r
  - rsync
  - duplicity
  - bontmia
  - backupninja
  - duplicity
  - time
  - machine
  - bitkeeper
  - git
  - bazaar
  - bzr
So what makes Git so useful?

(almost) Everything is local

- everything is fast
- every clone is a backup
- you work *mainly offline*

Ultra Fast, Efficient & Robust

- Snapshots, not patches (deltas)
- **Cheap branching and merging**
  - Strong support for thousands of parallel branches
- Cryptographic integrity everywhere
Other Git features

- **Git doesn’t delete**
  - Immutable objects, Git generally only adds data
  - If you mess up, you can usually recover your stuff
    - ✓ Recovery can be tricky though
Other Git features

- **Git doesn’t delete**
  - Immutable objects, Git generally only adds data
  - If you mess up, you can usually recover your stuff
    - Recovery can be tricky though

**Git Tools / Extension**

- cf. **Git submodules** or **subtrees**
- **Introducing git-flow**
  - workflow with a strict branching model
  - offers the git commands to follow the workflow

```bash
$> git flow init
$> git flow feature { start, publish, finish } <name>
$> git flow release { start, publish, finish } <version>
```
Introduction
Agenda
Overview of managed IT Infrastructure

IT/Dev[op]s Army Knives Tools
SSH Secure Shell
PGP / GPG: Gnu Privacy Guard
Vagrant
Puppet
Ruby / Python / Markdown-based Documentations
Password Management

Research Computing Platform @ UL

Git[Lab] @ UL and VCS

Git[Lab] Around You
About Version Control System (VCS)

Git Basics
Installing Git
Git theory
Basic Commands
Branching and Merging

Collaborating / Working together

Advanced Git Topics
Git Submodules
Rebasing
Using Git over Subversion Repository
More Cool stuff
Git Basics

Summary

1. Introduction
   - Agenda
   - Overview of managed IT Infrastructure

2. IT/Dev[op]s Army Knives Tools
   - SSH Secure Shell
   - PGP / GPG: Gnu Privacy Guard
   - Vagrant
   - Puppet
   - Ruby / Python / Markdown-based Documentations
   - Password Management

3. Research Computing Platform @ UL

4. Git[Lab] @ UL and VCS

5. Git Basics
   - Installing Git
   - Git theory
   - Basic Commands
   - Branching and Merging

6. Collaborating / Working together

7. Advanced Git Topics
   - Git Submodules
   - Rebasing
   - Using Git over Subversion Repository
   - More Cool stuff

Git[Lab] Around You
About Version Control System (VCS)

S. Varrette (PCOG Research unit)
### Installation Notes

**Linux / Mac OS**

- `$ apt-get install git-core git-flow`  
  # On Debian-like systems
- `$ yum install git gitflow`  
  # On CentOS-like systems
- `$ brew install git git-flow`  
  # On Mac OS, using Homebrew
### Installation Notes

#### Linux / Mac OS

```bash
$> apt-get install git-core git-flow  # On Debian-like systems
$> yum install git gitflow          # On CentOS-like systems
$> brew install git git-flow        # On Mac OS, using Homebrew
```

#### Windows

**MsysGit**

- Incl. Git Bash/GUI & Shell Integration
- Use PLINk from Putty
- Install **Git bash** + command prompt
- Select checkout windows / commit unix
## Git Basics

### Installation Notes

#### Linux / Mac OS

```bash
$> apt-get install git-core git-flow  # On Debian-like systems
$> yum install git gitflow          # On CentOS-like systems
$> brew install git git-flow         # On Mac OS, using Homebrew
```

#### Windows

- Include Git Bash/GUI & Shell Integration
- Use PLINk from Putty
- Install **Git bash** + command prompt
- Select checkout windows / commit unix

---

Your Turn! Ensure you have git installed
Git Basics

Git GUI

(default) Gitk

S. Varrette (PCOG Research unit)

IT/Dev[op]s Army Knives Tools for the researcher
Git Basics

Git GUI (Mac OS) GitX-dev

http://rowanj.github.io/gitx/
Git Basics

Git GUI (Windows/Mac) SourceTree

1. Let it install a default git ignore file
2. Make it load your SSH key created with Putty

http://www.sourcetreeapp.com/
Global Git configuration are stored in `~/.gitconfig`

You **SHOULD** at least configure your name and email to commit

open a terminal (Git bash under windows) for the below commands

```bash
$> git config --global user.name "Firstname LastName"
$> git config --global user.email "Firstname.Lastname@uni.lu"
$> git config --global color.ui true
$> git config --global core.editor vim
```

# Colors

# Editor
Global Git configuration are stored in ~/.gitconfig

Ex: see my personal .gitconfig

You SHOULD at least configure your name and email to commit

open a terminal (Git bash under Windows) for the below commands

```bash
$> git config --global user.name "Firstname LastName"
$> git config --global user.email "Firstname.Lastname@uni.lu"
$> git config --global color.ui true # Colors
$> git config --global core.editor vim # Editor
```

Your Turn!

Then check the changes by: git config -l | grep user
You can also create git command aliases in `~/.gitconfig`.

```
[alias]
up = pull origin
pu = push origin
st = status
df = diff
ci = commit -s
c0 = checkout
br = branch
w = whatchanged --abbrev-commit
ls = ls-files
gr = log --graph --oneline --decorate
amend = commit --amend
```
Git Basics

Summary

1. Introduction
   - Agenda
   - Overview of managed IT Infrastructure

2. IT/Dev[op]s Army Knives Tools
   - SSH Secure Shell
   - PGP / GPG: Gnu Privacy Guard
   - Vagrant
   - Puppet
   - Ruby / Python / Markdown-based Documentations
   - Password Management

3. Research Computing Platform @ UL

4. Git[Lab] @ UL and VCS

5. Git Basics
   - Installing Git
   - Git theory
   - Basic Commands
   - Branching and Merging

6. Collaborating / Working together

7. Advanced Git Topics
   - Git Submodules
   - Rebasings
   - Using Git over Subversion Repository
   - More Cool stuff

S. Varrette (PCOG Research unit)
The local repository lives in the .git directory.

The **staging area** tracks what will go into the next commit  
→ AKA “the index”
Git Basics

Summary

1 Introduction
   Agenda
   Overview of managed IT Infrastructure

2 IT/Dev[op]s Army Knives Tools
   SSH Secure Shell
   PGP / GPG: Gnu Privacy Guard
   Vagrant
   Puppet
   Ruby / Python / Markdown-based Documentations
   Password Management

3 Research Computing Platform @ UL

4 Git[Lab] @ UL and VCS

5 Git Basics
   Installing Git
   Git theory
   Basic Commands
   Branching and Merging

6 Collaborating / Working together

7 Advanced Git Topics
   Git Submodules
   Rebasing
   Using Git over Subversion Repository
   More Cool stuff

S. Varrette (PCOG Research unit)
Creating a Repository

$> git [flow] init

- Initializes a new git (flow) repository in the current directory
Creating a Repository

$> git [flow] init

- Initializes a new git (flow) repository in the current directory

Your Turn!

$> cd /tmp
$> mkdir firstproject
$> cd firstproject

$> git init
Initialized empty Git repository in /private/tmp/firstproject/.git/
Cloning a Repository

$> \text{git\ clone\ [\text{--recursive}] <url\> \ [<path>]}$

<table>
<thead>
<tr>
<th>Type</th>
<th>URL Format / Example</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local</td>
<td>/path/to/project.git</td>
<td>n/a</td>
</tr>
<tr>
<td>SSH</td>
<td>git+ssh://user@server:port/project.git</td>
<td>22</td>
</tr>
<tr>
<td>Git</td>
<td>git://server/project.git</td>
<td>9418</td>
</tr>
<tr>
<td>HTTPS</td>
<td><a href="https://github.com/Falkor/falkorlib.git">https://github.com/Falkor/falkorlib.git</a></td>
<td>443</td>
</tr>
</tbody>
</table>
Git Basics

Cloning a Repository

$> git clone [--recursive] <url> [<path>]

Your Turn!

$> cd /tmp
$> git clone https://github.com/ULHPC/tutorials.git
Cloning into 'tutorials'...
remote: Counting objects: 1247, done.
remote: Compressing objects: 100% (63/63), done.
remote: Total 1247 (delta 32), reused 0 (delta 0), pack-reused 1181
Receiving objects: 100% (1247/1247), 15.74 MiB | 3.08 MiB/s, done.
Resolving deltas: 100% (588/588), done.
Checking connectivity... done.
$> git clone --recursive \
    https://github.com/ULHPC/tutorials.git /tmp/tutorials2
Inspecting a Repository

$> \text{git status [-s]} \quad \# \text{-s: short / simplified output}$
$> git status [-s]                     # -s: short / simplified output

Your Turn!

$> cd /tmp/firstproject
$> git status
On branch master

Initial commit

nothing to commit

# Create an empty file
$> touch README.md

$> git status
On branch master

Initial commit

Untracked files:
   README

nothing added to commit but untracked files present
$> git status -s
?? README
Add / Tracking [new] file(s)

```bash
$ git add [-f] <pattern>
```

- Adds changes to the index
  - Add a specific file: `git add README`
  - Add a set of files: `git add *.py`

- Beware that empty directory cannot be added **directly**
  - due to the internal file representation (**blobs**)
  - **Tips**: add an hidden file `.empty` (or `.gitignore`)
Add / Tracking [new] file(s)

$> \text{git add } [-f] \text{ <pattern>}

- Adds changes to the index
  - Add a specific file: \text{git add README}
  - Add a set of files: \text{git add *.py}

- Beware that empty directory cannot be added \textbf{directly}
  - due to the internal file representation (\textit{blobs})
  - \textbf{Tips}: add an hidden file .empty (or .gitignore)

Your Turn!

$> \text{cd /tmp/firstproject}$
$> \text{git status -s}$
?? \text{README}$

$> \text{git add README}$
$> \text{git status -s}$
A \text{ README}
Committing your changes

$> \text{git commit} \ [-s] \ [-m} \ \text{"msg"}]

- Commit all changes: \text{git commit} -a
Git Basics

Committing your changes

$> git commit [-s] [-m "msg"]

- Commit all changes: git commit -a

Your Turn!

$> cd /tmp/firstproject
$> git commit -s -m "add README"  # OR git ci -m "add README"
[master (root-commit) ee60f53] add README
  1 file changed, 0 insertions(+), 0 deletions(-)
  create mode 100644 README
$> git status                # OR git st
On branch master
nothing to commit, working directory clean
Removing Files

```bash
$ git rm [-rf] [--cached] <file>
```

- `--cached`: remove from Staging area
  - otherwise (default): from index and file system
Ignoring files from staging: `.gitignore`

- you can create a `.gitignore` file listing patterns to ignore
  - Blank lines or lines starting with `#` are ignored
  - End pattern with slash (`/`) to specify a directory
  - Negate pattern with exclamation point (`!`)
- Collection of useful `.gitignore` templates

- `.DS_Store`
- `*~`
- `*.asv`
- `*.m~`
- `*.mex*`
- `tmp/*`
- `\LaTeX.gitignore`
- `Python.gitignore`
- `Ruby.gitignore`
Git Basics

Moving Files

$>\text{git mv} \ <\text{source}> \ <\text{destination}>

\texttt{# Equivalent of:}
\texttt{mv \ <source> \ <destination>}
\texttt{git \ rm \ <source>}
\texttt{git \ add \ <destination>
Git Basics

Moving Files

$> \text{git mv} \ <\text{source}> \ <\text{destination}>

# Equivalent of:
\text{mv} \ <\text{source}> \ <\text{destination}>
\text{git rm} \ <\text{source}>
\text{git add} \ <\text{destination}>

Your Turn!

$> \text{cd} \ /\text{tmp/firstproject}$
$> \text{git mv} \ \text{README} \ \text{README.md}$
$> \text{git status}$
\text{On} \ \text{branch master}$
Changes to be committed:
\hspace{1cm} \text{renamed:} \ \text{README} \ -> \ \text{README.md}$
$> \text{git commit} \ -m \ "a first move"$
$> git log [-p] [--stat] [--graph --oneline --decorate]

- `-p` / `--stat`: show the differences introduced in each commit
- You can also perform some date filtering
  $> git log --since=2.weeks
- Ncurses-based text-mode interface: `tig`
## Git Basics

### Check the Commit History

```bash
$ git log [-p] [--stat] [--graph --oneline --decorate]
```

- `-p` / `--stat`: show the differences introduced in each commit
- You can also perform some date filtering
  ```bash
  $ git log --since=2.weeks
  ```
- Ncurses-based text-mode interface: `tig`

### Your Turn!

```bash
$ cd /tmp/firstproject
$ git log --oneline --graph --decorate  # OR git gr
  * f1f0c27 (HEAD -> master) a first move
  * ee60f53 add README
$ git log -p -1  # only the last commit OR git show
$ tig
```
Git Basics

Show differences

$>$ git diff [--cached] [<ref>]

- Check **un-staged** changes: `git diff`  
  `--cached`: check **staged** changes

- Relative to a specific revision:
  
  $>$ git diff 1776f5
  $>$ git diff HEAD^
Undoing Things

$> git commit --amend

# Change the last commit
Git Basics

Undoing Things

$> git commit --amend  # Change the last commit

$> git unstage <file>  # or git reset HEAD <file>
Undoing Things

- `$\texttt{git commit --amend}$` # Change the last commit
- `$\texttt{git unstage <file>}$` # or $\texttt{git reset HEAD <file>}$
- `$\texttt{git checkout -- <file>}$` # DANGER! Un-modify modified file

- Restore to the last committed/cloned version: **all** changes are lost!
Git Basics

Undoing Things

- `$ git commit --amend`  
  # Change the last commit

- `$ git unstage <file>`  
  # or `$ git reset HEAD <file>`

- `$ git checkout -- <file>`  
  # DANGER! Un-modify modified file

- `$ git revert <commit>`  
  # revert a `<commit>`

Make a new commit that undoes all changes made in `<commit>`
Undoing Things

$> \texttt{git commit --amend} \quad \# \textit{Change the last commit}$

$> \texttt{git unstage <file>} \quad \# \textit{or git reset HEAD <file>}$

$> \texttt{git checkout -- <file>} \quad \# \textit{DANGER! Un-modify modified file}$

$> \texttt{git revert <commit>} \quad \# \textit{revert a <commit>}$

Your Turn!

$> \texttt{cd /tmp/firstproject}$

$> \texttt{git commit --amend}$

$> \texttt{echo 'toto' >> README.md}$

$> \texttt{cat README.md && git status}$

$> \texttt{git checkout -- README}$

$> \texttt{git status}$
Git Basics

Summary

Basic Workflow

Edit files
Stage the changes
Review your changes
Commit the changes

vim / emacs / subl ...

```
git add

git status

git commit
```
For cheaters: A Basicerer Workflow

Edit files
Stage & commit the changes

vim / emacs / subl ...
git commit -a
For cheaters: A Basicerer Workflow

Edit files
vim / emacs / subl ...
Stage & commit the changes
git commit -a

Advises

- Commit early, commit often!
  - commits = save points
  - use descriptive commit messages
- Don’t get out of sync with your collaborators
- Commit the sources, not the derived files
Git Basics

Summary

1. Introduction
   - Agenda
   - Overview of managed IT Infrastructure

2. IT/Dev[op]s Army Knives Tools
   - SSH Secure Shell
   - PGP / GPG: Gnu Privacy Guard
   - Vagrant
   - Puppet
   - Ruby / Python / Markdown-based Documentations
   - Password Management

3. Research Computing Platform @ UL

4. Git[Lab] @ UL and VCS

5. Git Basics
   - About Version Control System (VCS)
   - Installing Git
   - Git theory
   - Basic Commands
   - Branching and Merging

6. Collaborating / Working together

7. Advanced Git Topics
   - Git Submodules
   - Rebasings
   - More Cool stuff

S. Varrette (PCOG Research unit)

IT/Dev[op]s Army Knives Tools for the researcher
Git Basics

Data Model

- **Immutable** objects
  - ← **Blob**: File content
  - ← **Tree**: Directory List
  - ← **Commit**: Pointer to a snapshot / tree
  - ← **Tag**: Pointer to commit

- **Git Branch**: Lightweight, movable pointer to a commit (HEAD: current branch)
Git Basics

Data Model Example

README.md

‘Hello’ Project

This is Seb’s first Git project

Licensed under GPL

hello.c

#include <stdio.h>

int main(void) {
    printf("Hello\n");
    return 0;
}
Git Basics

Data Model Example

**README.md**

‘Hello’ Project

This is Seb’s first Git project

Licenced under GPL

**hello.c**

```c
#include <stdio.h>

int main(void) {
    printf("Hello\n");
    return 0;
}
```

(Compress) Checksum

`c3db2`

‘Hello’ Project

This is Seb’s first Git project

Licenced under GPL
Git Basics

Data Model Example

```
#include <stdio.h>

int main(void) {
    printf("Hello\n");
    return 0;
}
```

`Hello` Project

This is Seb’s first Git project

Licenced under GPL

Blob = File Content

`c3db2`

(Compress) Checksum

'Hello' Project

This is Seb’s first Git project

Licenced under GPL

S. Varrette (PCOG Research unit)
Git Basics

Data Model Example

`include <stdio.h>
int main(void) {
    printf("Hello\n");
    return 0;
}

Hello Project
===============
This is Seb's first Git project
Licenced under GPL

README.md

`include <stdio.h>
int main(void) {
    printf("Hello\n");
    return 0;
}

Hello Project
===============
This is Seb's first Git project
Licenced under GPL

checksum: c3db2

checksum: f13eb

S. Varrette (PCOG Research unit)
Git Basics

Data Model Example

```
#include <stdio.h>

int main(void) {
    printf("Hello\n");
    return 0;
}
```

---

This is Seb’s first Git project
Licenced under GPL

---

'Hello' Project
Licenced under GPL

---

README.md

```
Hello

This is Seb’s first Git project
Licenced under GPL
```
Git Basics

Data Model Example

Tree = List of trees AND blobs

```
Tree
/          \  
|          |   |
hello.c   README.md
```

**README.md**

'Hello' Project
===============
This is Seb's first Git project
Licenced under GPL

**hello.c**

```c
#include <stdio.h>

int main(void) {
    printf("Hello\n");
    return 0;
}
```

S. Varrette (PCOG Research unit)
**Git Basics**

**Data Model Example**

```
#include <stdio.h>

int main(void) {
    printf("Hello\n");
    return 0;
}
```

- **README.md**
  - 'Hello' Project
  - This is Seb's first Git project
  - Licenced under GPL

- **hello.c**
  - `#include <stdio.h>`
  - `int main(void) {` `printf("Hello\n");` `return 0;` `}`

```
./
|-- hello.c f13eb
|  `-- README.md c3db2
```

```
./
|-- hello.c f13eb
|  `-- README.md c3db2
```

---

S. Varrette (PCOG Research unit)  
IT/Dev[op]s Army Knives Tools for the researcher
Git Basics

Data Model Example

README.md

‘Hello’ Project

This is Seb’s first Git project

Licenced under GPL

hello.c

#include <stdio.h>

int main(void) {
    printf(“Hello\n”);
    return 0;
}
Git Basics

Data Model Example

**Commit = Pointer to a Tree**

<table>
<thead>
<tr>
<th>commit</th>
<th>size</th>
</tr>
</thead>
<tbody>
<tr>
<td>tree</td>
<td>1a738</td>
</tr>
<tr>
<td>parent</td>
<td>NULL</td>
</tr>
<tr>
<td>author</td>
<td>svarrette</td>
</tr>
<tr>
<td>committer</td>
<td>svarrette</td>
</tr>
<tr>
<td>commit A</td>
<td>that's my first super cool commit message.</td>
</tr>
</tbody>
</table>

- `commit A`: This is Seb’s first Git project
- `HEAD`: the super cool commit message.
- `Hello`: Project

---

**README.md**

`Hello` Project

This is Seb’s first Git project

Licenced under GPL

---

**hello.c**

```c
#include <stdio.h>

int main(void) {
    printf("Hello\n");
    return 0;
}
```

---

S. Varrette (PCOG Research unit)
Git Basics

Data Model Example

**README.md**

`Hello` Project

This is Seb's first Git project

Licenced under GPL

**hello.c**

```c
#include <stdio.h>

int main(void) {
    printf("Hello\n");
    return 0;
}
```

---

S. Varrette (PCOG Research unit)
Git Basics

Data Model Example

**README.md**

- 'Hello' Project
- This is Seb's first Git project
- Licenced under GPL

**hello.c**

```
#include <stdio.h>

int main(void) {
    printf("Hi\n");
    return 0;
}
```

Commit A

Commit B

```
./
|-- hello.c  f13eb
|   |-- README.md  c3db2
```

Licenced under GPL

S. Varrette (PCOG Research unit)
Git Basics

Data Model Example

#include <stdio.h>

int main(void)
{
    printf("Hi\n");
    return 0;
}

6d4a1

#include <stdio.h>

int main(void)
{
    printf("Hello\n");
    return 0;
}

f13eb

Hello Project
This is Seb’s first Git project
Licenced under GPL

c3db2

Hello Project
This is Seb’s first Git project
Licenced under GPL

1a738

Hello Project
This is Seb’s first Git project
Licenced under GPL

S. Varrette (PCOG Research unit)
Git Basics

Data Model Example

```
#include <stdio.h>

int main(void) {
    printf("Hi\n");
    return 0;
}
```

```
#include <stdio.h>

int main(void) {
    printf("Hello\n");
    return 0;
}
```

This is Seb’s first Git project
Licenced under GPL
Git Basics

Data Model Example

**README.md**

‘Hello’ Project

This is Seb’s first Git project

Licenced under GPL

**hello.c**

```c
#include <stdio.h>

int main(void) {
    printf("Hi\n");
    return 0;
}
```

Commit A

Commit B

**File Structure**

- `/hello.c`
  - `f13eb`
  - `c3db2`

- `/README.md`
  - `c3db2`

---

S. Varrette (PCOG Research unit)

IT/Dev[op]s Army Knives Tools for the researcher
Data Model Example

README.md

‘Hello’ Project
==================
This is Seb’s first Git project
Licenced under GPL

hello.c

#include <stdio.h>

int main(void) {
    printf("Hi\n");
    return 0;
}
Git Basics

Data Model Example

```
#include <stdio.h>
int main(void) {
    printf("Hi\n");
    return 0;
}
```

```
#include <stdio.h>
int main(void) {
    printf("Hello\n");
    return 0;
}
```

This is Seb’s first Git project
Licenced under GPL

S. Varrette (PCOG Research unit)
Git Basics

Data Model Example

`README.md`

‘Hello’ Project
---------------
This is Seb’s first
Git project
Licenced under GPL

`hello.c`

```
#include <stdio.h>

int main(void) {
    printf("Hi\n");
    return 0;
}
```

This is Seb’s first
Git project
Licenced under GPL

#include <stdio.h>

int main(void) {
    printf("Hi\n");
    return 0;
}
# Data Model Example

- **README.md**
  - 'Hello' Project
  - =====================
  - This is Seb's first Git project
  - Licence under GPL

- **hello.c**
  ```c
  #include <stdio.h>
  int main(void) {
    printf("Hi\n");
    return 0;
  }
  ```

---

S. Varrette (PCOG Research unit)  
IT/Dev[op]s Army Knives Tools for the researcher
**Git Basics**

### Data Model Example

**README.md**

‘Hello’ Project

This is Seb’s first Git project

Licenced under GPL

**hello.c**

```c
#include <stdio.h>

int main(void) {
    printf("Hi\n");
    return 0;
}
```

Branch = pointer to a commit

---

S. Varrette (PCOG Research unit)
Git Basics

Data Model Example

```
#include <stdio.h>

int main(void) {
    printf("Hi\n");
    return 0;
}
```

```
#include <stdio.h>

int main(void) {
    printf("Hello\n");
    return 0;
}
```
Branching

$> git branch <name>  # create a branch <name>
Branching

\$\>$ \texttt{git branch <name>} \hspace{1cm} \# create a branch <name>

\$\>$ \texttt{git branch -d <name>} \hspace{1cm} \# delete the branch <name>

- use \texttt{-D} instead of \texttt{-d} to force deletion
Git Basics

Branching

$> git branch <name>  # create a branch <name>

$> git branch -d <name>  # delete the branch <name>

$> git branch [-a]  # List [all] the branches
Git Basics

Branching

$> git branch <name>  # create a branch <name>

$> git branch -d <name>  # delete the branch <name>

$> git branch [-a]  # List [all] the branches

$> git checkout [-b] <name>  # switch to the branch <name>

- -b: create the branch before switching
- changes committed through git commit are committed to HEAD
Git Basics

Branching

$> git branch <name>  # create a branch <name>

$> git branch -d <name>  # delete the branch <name>

$> git branch [-a]  # List [all] the branches

$> git checkout [-b] <name>  # switch to the branch <name>

- **Switching** branches **changes** the files in your Working directory
  → since you change the HEAD snapshot...
Git Basics

Tags

$> git tag [-s] <name> [-m 'msg']

- `-s`: GPG-signed tag, assuming you have configured your signing key

$> git config --global user.signingkey 0xDD01D5C1
**Tags**

$> \text{git tag [-s]} <\text{name}> [-m 'msg']$

- `-s`: GPG-signed tag, assuming you have configured your signing key

  $> \text{git config --global user.signingkey 0xDD01D5C1}$

**Your Turn!**
Branch and Tags Hands-on

```bash
$> cd /tmp/firstproject

$> git branch
* master

$> git tag v1.0 -m 'first tag'

$> git gr
* f31c173 (HEAD -> master, tag: v1.0) a first move with amend
* ee60f53 add README

$> git branch testing

$> git checkout testing  # Move to the 'testing' branch

$> echo 'testing' >> README.md && git commit -a -m "testing 1"
[testing 7afa96d] testing 1
  1 file changed, 1 insertion(+)

$> git checkout master  # return to 'master'

$> echo 'master' >> README.md && git commit -a -m "master"
[master 72d4d5f] master
  1 file changed, 1 insertion(+)

$> git gr

$> gitx  # or gitk
```

S. Varrette  (PCOG Research unit)  IT/Dev[op]s Army Knives Tools for the researcher
Daily Branching Example

- Snapshot A
- Snapshot B
- Snapshot C
Git Basics

Daily Branching Example

HEAD

master

98ca9  34ac2  f30ab
Daily Branching Example

(master)$> git branch testing  # create a branch named 'testing'

```
HEAD
  ↓
master
  ↓
98ca9
  ↓
34ac2
  ↓
f30ab
  ↓
testing
```
Git Basics

Daily Branching Example

(master)$> git checkout testing  # switch to the 'testing' branch
Daily Branching Example

(testing) $> $ git commit -a -m "made a change"
# and commit them

S. Varrette (PCOG Research unit)
Daily Branching Example

(testing)$> \textit{git checkout master} # \textit{switch back to 'master' branch}
Git Basics

Daily Branching Example

(testing)$> vim README.md

# make some edits

(testing)$> git commit -a -m "intro"

# introduce divergence!

```
DIVERGENCE
```

```
HEAD
  ↓
master
  ↓
c2b9e
  ↓
87ab2
  ↓
testing
```

```
98ca9
  ↓
34ac2
  ↓
f30ab
```

S. Varrette (PCOG Research unit)
Diverging

- Changes are committed into two branches independently
  - Then the branches diverge
Git Basics

Diverging / Converging (Fork-Join)

Diverging

- Changes are committed into two branches independently
  - Then the branches **diverge**

Converging to join branches

1. **merge** (if possible in **fast-forward** mode)
2. **rebase**
Merging

$>$ git merge [--no-ff] <branch>

- Different auto-merge strategies
  - fast-forward (if possible)
  - 3-ways (regular)
- Usually painless ;)

BRANCHED FROM MASTER 3 WEEKS AGO
MERGED BACK WITHOUT ANY CONFLICTS
Git Basics

Merging

```bash
$ git merge [--no-ff] <branch>
```

- Different auto-merge strategies
  - fast-forward (if possible)
  - 3-ways (regular)
- Usually painless ;)

In case of conflicts:
- Resolve the conflicts manually
  - vim / emacs / subl ...
  - check for the sequence <<< in the text
- then mark as resolved
- and trigger the merge commit
Git Basics

Daily Branching Example

```
HEAD
  ↓
master
  ↓
c2b9e
  ↓
87ab2
  ↓
testing
  ↑
f30ab
  ↑
34ac2
  ↑
98ca9
```

S. Varrette (PCOG Research unit)
Daily Branching Example

(master)$> git checkout -b hotfix
# create and switch to 'hotfix'

(hotfix)$> vim test.rb
# make some edits...

(hotfix)$> git commit -a -m "hotfix"
# and commit them
Git Basics

Daily Branching Example

(hotfix)$> git checkout master

# switch back to 'master' branch
Git Basics

Daily Branching Example

(master)$> git merge hotfix  # merge the 'hotfix' branch (fast-forward)
Daily Branching Example

(master)$> git merge hotfix  # merge the 'hotfix' branch (fast-forward)
Git Basics

Daily Branching Example
Daily Branching Example

(master)$> git branch -d hotfix # delete the (useless) 'hotfix' branch
Daily Branching Example

(master)$> git branch -d hotfix  # delete the (useless) 'hotfix' branch
Git Basics

Daily Branching Example

```
98ca9 → 34ac2 → f30ab
```

HEAD

```
master
```

```
c2b9e
```

```
ab126
```

```
87ab2
```

```
1ff54
```

```
testing
```
Git Basics

Daily Branching Example

(master)$> git merge testing

# merge the 'testing' branch (3-ways)

![Diagram of git merge process]

Common Ancestor

98ca9 → 34ac2 → f30ab

HEAD

master

c2b9e → ab126

Snapshot to Merge INTO

87ab2 → 1ff54

Snapshot to Merge In

testing

S. Varrette  (PCOG Research unit)
Daily Branching Example

(master)$> git merge testing  # merge the 'testing' branch (3-ways)

3-ways Merge

Create a new commit
Can be forced using --no-ff
(master)\$ > git merge testing                # merge the 'testing' branch (3-ways)
Merging and solving conflicts Hands-on

Your Turn!
Git Basics

Merging and solving conflicts Hands-on

```
$> cd /tmp/firstproject
$> git checkout master
Switched to branch 'master'
$> git checkout -b hotfix
Switched to a new branch 'hotfix'
$> touch test.rb && git add test.rb && git commit -m "hotfix"
[hotfix ac188bd] hotfix
  1 file changed, 0 insertions(+), 0 deletions(-)
  create mode 100644 test.rb
$> git checkout master
Switched to branch 'master'
$> git gr
* 72d4d5f (HEAD -> master) master
* f31c173 (tag: v1.0) a first move with amend
* ee60f53 add README
```
Fast-Forward Merge

```bash
$> git merge hotfix
Updating 72d4d5f..ac188bd
Fast-forward
test.rb | 0
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 test.rb
$> git gr
* ac188bd (HEAD -> master, hotfix) hotfix
* 72d4d5f master
* f31c173 (tag: v1.0) a first move with amend
* ee60f53 add README

$> git branch -d hotfix
Deleted branch hotfix (was ac188bd)
```
Merging and solving conflicts Hands-on

Solving conflicts

```bash
$> git merge testing
CONFLICT (content): Merge conflict in README.md
Automatic merge failed; fix conflicts and then commit the result.
$> cat README.md
<<<<<<< HEAD
master
=======
testing
>>>>>>> testing
$> vim README.md  # Edit to solve the conflicts
$> cat README.md
master corrected
```
Solving conflicts

$> git status  # OR git st
On branch master
You have unmerged paths.

Unmerged paths:
    both modified: README.md

no changes added to commit
Solving conflicts and 3-way merge

```bash
$> git add README.md  # Mark as corrected / conflict solved
$> git commit
Recorded resolution for 'README.md'.
[master ef299b7] Merge branch 'testing'
$> git gr
*  ef299b7 (HEAD -> master) Merge branch 'testing'
|\  
| * 7afa96d (testing) testing 1
* | ac188bd hotfix
* | 72d4d5f master
|/
* f31c173 (tag: v1.0) a first move with amend
* ee60f53 add README
```
Git Basics

Branching Workflow

production
C1 ← C2 ← C3 ← C4 ← C5 ← C6 ← C7 ← master
feature/topic
Git Basics

Branching Workflow

production
C1

master
C2 → C3 → C4 → C5

feature/topic
C6 ← C7
Git Basics

Branching Workflow

production

master

feature/topic

feature/sc2015
Git Basics

Git-flow to the rescue

http://nvie.com/posts/a-successful-git-branching-model/

$> git flow init

$> git flow feature { start, publish, finish } <name>

$> git flow release { start, publish, finish } <version>

- Ensure two long running branches
  - Production: the stable branch
    - ideally holding only tags of the successive releases
  - Master / Devel: the main branch where the developments occur
- On demand: make a new feature branch feature/<name>
- From time to time, release your code into production and tag
**Git Basics**

**Git-flow Illustrated**

[Source: Nvie]

---

Feature branches | master or devel | release branches | hotfixes | production

**Feature for future release**

**Major feature for next release**

**Severe bug fixed for production: hotfix 0.2**

**Incorporate changes in develop**

**Start of release branch for 1.0**

**Only bugfixes!**

**Bugfixes from rel. branch may be continuously merged back into develop**

**Tag 0.1**

**Tag 0.2**

**Tag 1.0**

---

S. Varrette (PCOG Research unit)

IT/Dev[op]s Army Knives Tools for the researcher
Git Basics

Git-flow Setup using FalkorLib

- Initiate a **Git-flow-ready repository using FalkorLib**
  - [Personnal] Ruby Library offering the `falkor` binary

```bash
>$ falkor new repo [--rake] # setup the current directory
```

- The repository is fed with a root Makefile (or Rakefile)
  - facilitate repository setup upon cloning

```bash
>$ git clone <url> && cd <cloned_dir>
>$ make setup
```

  - project releasing using **Git-flow made easy**

```bash
>$ make start_bump_{major,minor,patch} # bump version with git-flow
>$ make release
```
Summary

1. Introduction
   - Agenda
   - Overview of managed IT Infrastructure

2. IT/Dev[op]s Army Knives Tools
   - SSH Secure Shell
   - PGP / GPG: Gnu Privacy Guard
   - Vagrant
   - Puppet
   - Ruby / Python / Markdown-based Documentations
   - Password Management

3. Research Computing Platform @ UL

4. Git[Lab] @ UL and VCS

5. Git Basics
   - Installing Git
   - Git theory
   - Basic Commands
   - Branching and Merging

6. Collaborating / Working together

7. Advanced Git Topics
   - Git Submodules
   - Rebasings
   - Using Git over Subversion Repository
   - More Cool stuff

S. Varrette (PCOG Research unit)

IT/Dev[op]s Army Knives Tools for the researcher
Collaborating / Working together

Working Together

- Sign-up on Github
  - [http://github.com](http://github.com)
  - Best place to share public repository
Collaborating / Working together

Working Together

- **Sign-up on Github**
  - → [http://github.com](http://github.com)
  - → Best place to share **public** repository

- **Alternative for **private** projects:**
  - → Gitlab @ Uni.lu
    - [https://gitlab.uni.lu](https://gitlab.uni.lu)
Collaborating / Working together

Working Together

- Sign-up on Github
  - http://github.com
  - Best place to share public repository

- Alternative for private projects:
  - Gitlab @ Uni.lu
    - https://gitlab.uni.lu

- Setup your own Git server: gitolite
  - https://github.com/sitaramc/gitolite
  - Management through the gitolite-admins Git repository (!)
  - A single user (git) to interact with all repositories
    - Map users through their (multiple) SSH keys
    - Fine-grained access control

S. Varrette (PCOG Research unit)
Working with remotes

Local

- working directory
- staging area
- git directory (repository)

Remote

- remote repo

- git add
- git commit
- git push
- git fetch / git pull
- git merge
- git checkout
Collaborating / Working together

Remotes

$> git remote [-v]

- Other clones of the same repository
  - Can be local (another checkout) or remote (coworker, central server)
  - default remotes for push and pull actions: origin
    - origin is set upon clone
Collaborating / Working together

Remotes

$> git remote [-v]

- Other clones of the same repository
  - Can be local (another checkout) or remote (coworker, central server)
  - default remotes for push and pull actions: origin
    - origin is set upon clone

Your Turn!

$> cd /tmp/tutorials
$> git remote
origin
$> git remote -v
origin  https://github.com/ULHPC/tutorials.git (fetch)
origin  https://github.com/ULHPC/tutorials.git (push)
Collaborating / Working together

Adding Remotes

```bash
$> git remote add <name> <url>
```
Collaborating / Working together

Adding Remotes

$> git remote add <name> <url>

Your Turn!

- Fork the ULHPC/tutorials (as <yourlogin>/tutorials)
- Clone and add the upstream remote to the original repository

$> git clone https://github.com/<yourlogin>/tutorials.git /tmp/fork
$> cd /tmp/fork
$> git remote add upstream https://github.com/ULHPC/tutorials.git
$> git remote -v

origin https://github.com/<yourlogin>/tutorials.git (fetch)
origin https://github.com/<yourlogin>/tutorials.git (push)
upstream https://github.com/ULHPC/tutorials.git (fetch)
upstream https://github.com/ULHPC/tutorials.git (push)
Collaborating / Working together

Removing Remotes

$> git remote rm <name>
Collaborating / Working together

Removing Remotes

\$> \texttt{git remote rm \textless\texttt{name\textgreater}}

Your Turn!

\$> \texttt{cd /tmp/fork}
\$> \texttt{git remote}
\texttt{origin}
\texttt{upstream}
\$> \texttt{git remove rm upstream}
Remote Branches

- Branches on remotes are represented locally as: `<remote>/<branch>`
  - **Ex:** origin/master
Remote Branches

- Branches on remotes are represented locally as: `<remote>/<branch>`
  - Ex: origin/master

Tracking Remote Branches

- You can track a remote branch `<remote>/<branch>`
  - assuming you have previously `fetch` the remote origin
  - creates the local branch `<branch>`

$> git branch --track <branch> origin/<branch>
Collaborating / Working together

Tracking Remote Branches

```bash
$> git branch --track <branch> origin/<branch>
```

Your Turn!

```bash
$> cd /tmp/tutorials
$> git branch -a
* devel
  remotes/origin/HEAD -> origin/devel
  remotes/origin/devel
  remotes/origin/production
$> git branch --track production origin/production
Branch production set up to track remote branch production from origin.
$> git branch
* devel
 * production
```
Collaborating / Working together

Pushing to your remote

$> \text{git push [<remote>]}$

- Transfer local commits of the \textbf{current} branch to a remote.
  - \textit{push to origin by default, assuming the current branch is tracked}
Collaborating / Working together

Pushing to your remote

$> \text{git push [<remote>]}$

- Transfer local commits of the \textbf{current} branch to a remote.
  - push to \texttt{origin} by default, assuming the current branch is tracked

\textbf{Your Turn!}

$> \text{cd /tmp/fork}$
$> \text{git remote}$
\texttt{origin}
\texttt{upstream}$> \text{touch new-file}$
$> \text{git add new-file}$
$> \text{git commit -m "add"}$

$> \text{git push}$
\texttt{Counting objects: 10, done.}$\texttt{Delta compression using up to 8 threads.}$\texttt{Compressing objects: 100\% (6/6), done.}$\texttt{Writing objects: 100\% (10/10), done.}$\texttt{Total 10 (delta 4), reused 0 (delta 0)}$
\texttt{To git@github.com:<yourlogin>/documents.git}$
\texttt{671eb88..c798919  devel -> devel}$

S. Varrette (PCOG Research unit)
Collaborating / Working together

Pulling from remotes

$> \text{git pull [\text{-\text{-rebase}}] [<\text{remote}>]} \quad \# \quad \text{--rebase = DANGER!}$

- fetch all commits from the remote and merge (or rebase)
  - allows for easy to use, equivalent to the advanced alternative:
    
    $> \text{git fetch [<remote>]}$
    $> \text{git merge <remote>/<branch>} \quad \# \quad \text{'git rebase' if --rebase}$
    
  - fetch: allows for inspection and manual merging of remote changes
Collaborating / Working together

Pulling from remotes

```bash
$> git pull [--rebase] [<remote>]
```

- `git pull` fetches all commits from the remote and merges (or rebase) them.
- `--rebase` is DANGER!

```bash
$> git fetch [<remote>]
$> git merge <remote>/<branch>
```

- `git fetch` allows for inspection and manual merging of remote changes.

Your Turn!

```bash
$> cd /tmp/tutorials
$> git pull
```

```
Updating ae97dae..06576e0
[...]
2 files changed, 4 insertions(+), 5 deletions(-)
```
Collaborating / Working together

**Publish a (local) branch on a remote**

```bash
$> git push -u origin <branch>
```

```bash
$> git flow feature publish <name>
```
Collaborating / Working together

Publish a (local) branch on a remote

$> \text{git push -u origin } \text{<branch>}$

$> \text{git flow feature publish } \text{<name>}$

- If you want to delete a remote branch

$> \text{git push origin --delete } \text{<branch>}$  
  \# DANGER!
Collaborating / Working together

Publish a (local) branch on a remote

```bash
$> git push -u origin <branch>
```

```bash
$> git flow feature publish <name>
```

- If you want to **delete** a **remote** branch

```bash
$> git push origin --delete <branch>
# DANGER!
```

Your Turn!

```bash
$> cd /tmp/fork
$> git flow feature start toto
$> git flow feature publish toto
```

```bash
$> git branch -a
$> git push origin --delete \ feature/toto
```

S. Varrette (PCOG Research unit)
Collaborating / Working together

Putting it all together

-gitlab.uni.lu-

master

0b743 → a6b4c → f42c5
Collaborating / Working together

Putting it all together

```
git clone git+ssh://git@gitlab.uni.lu/project.git
```

My Computer

```
origin/master ➔ Remote Branch
```

```
master ➔ Local Branch
```

---

S. Varrette  (PCOG Research unit)  IT/Dev[op]s Army Knives Tools for the researcher
Collaborating / Working together

Putting it all together

gitlab.uni.lu

My Computer

Someone else pushes
Collaborating / Working together

Putting it all together

```
git fetch origin
```

```
gitlab.uni.lu

0b743  →  a6b4c  →  f42c5  →  31b8e  →  190a3

master

My Computer

0b743  →  a6b4c  →  f42c5  →  31b8e  →  190a3

→ a38de  →  893cf

origin/master

→ master
```
Putting it all together
Collaborating / Working together

Putting it all together

My Computer

gitlab.uni.lu

master

origin

0b743

a6b4c

f42c5

31b8e

190a3

gitlab-lcsb.uni.lu

master

teamone

f42c5

31b8e

origin/master

teamone/master

S. Varrette (PCOG Research unit) IT/Dev[op]s Army Knives Tools for the researcher
Advanced Git Topics

Summary

1. Introduction
   Agenda
   Overview of managed IT Infrastructure

2. IT/Dev[op]s Army Knives Tools
   SSH Secure Shell
   PGP / GPG: Gnu Privacy Guard
   Vagrant
   Puppet
   Ruby / Python / Markdown-based Documentations
   Password Management

3. Research Computing Platform @ UL

4. Git[Lab] @ UL and VCS

Git[Lab] Around You
About Version Control System (VCS)

5. Git Basics
   Installing Git
   Git theory
   Basic Commands
   Branching and Merging

6. Collaborating / Working together

7. Advanced Git Topics
   Git Submodules
   Rebasings
   Using Git over Subversion Repository
   More Cool stuff

S. Varrette (PCOG Research unit)

IT/Dev[op]s Army Knives Tools for the researcher
Summary

1. Introduction
   Agenda
   Overview of managed IT Infrastructure

2. IT/Dev[op]s Army Knives Tools
   SSH Secure Shell
   PGP / GPG: GNU Privacy Guard
   Vagrant
   Puppet
   Ruby / Python / Markdown-based Documentations
   Password Management

3. Research Computing Platform @ UL

4. Git[Lab] @ UL and VCS

5. Git Basics
   Installing Git
   Git theory
   Basic Commands
   Branching and Merging

6. Collaborating / Working together

7. Advanced Git Topics
   Git Submodules
   Rebasings
   Using Git over Subversion Repository
   More Cool stuff
Git Submodules

$> git submodule add [-b <branch>] <url> <subdir>

- **Git submodule**: repository nested within another repository.
  - see it as a read-only snapshot
  - make symbolic links to the submodules files
- State saved in `.gitmodules` (git root directory)

[submodule ".submodules/Makefiles"]
  path = .submodules/Makefiles
  url = https://github.com/Falkor/Makefiles

- Explicit initialization is **mandatory**
  - **before** cloning:
    - `git clone --recursive`
  - **after** cloning:
    - `git submodule init` & `git submodule update`
Advanced Git Topics

Git Submodules - Update

```
$> git submodule add \
    https://github.com/Falkor/Makefiles .submodules/Makefiles
```

- You might need to **update** the submodules after fetch / pull
- You might wish to **upgrade** the submodules to the latest version

```
$> git submodule init
$> git submodule update
$> git submodule foreach \
    'git fetch origin; \
    git checkout $(git rev-parse --abbrev-ref HEAD); \ 
    git reset --hard origin/$(git rev-parse --abbrev-ref HEAD); \ 
    git submodule update --recursive; git clean -dfx'
```

- See make upgrade of this Makefile for repositories
### Advanced Git Topics

#### Summary

<table>
<thead>
<tr>
<th>1</th>
<th>Introduction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Agenda</td>
</tr>
<tr>
<td></td>
<td>Overview of managed IT Infrastructure</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2</th>
<th>IT/Dev[op]s Army Knives Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SSH Secure Shell</td>
</tr>
<tr>
<td></td>
<td>PGP / GPG: Gnu Privacy Guard</td>
</tr>
<tr>
<td></td>
<td>Vagrant</td>
</tr>
<tr>
<td></td>
<td>Puppet</td>
</tr>
<tr>
<td></td>
<td>Ruby / Python / Markdown-based Documentations</td>
</tr>
<tr>
<td></td>
<td>Password Management</td>
</tr>
</tbody>
</table>

| 3 | Research Computing Platform @ UL |

| 4 | Git[Lab] @ UL and VCS |

<table>
<thead>
<tr>
<th>5</th>
<th>Git Basics</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Installing Git</td>
</tr>
<tr>
<td></td>
<td>Git theory</td>
</tr>
<tr>
<td></td>
<td>Basic Commands</td>
</tr>
<tr>
<td></td>
<td>Branching and Merging</td>
</tr>
</tbody>
</table>

| 6 | Collaborating / Working together |

<table>
<thead>
<tr>
<th>7</th>
<th>Advanced Git Topics</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Git Submodules</td>
</tr>
<tr>
<td></td>
<td>Rebasinig</td>
</tr>
<tr>
<td></td>
<td>Using Git over Subversion Repository</td>
</tr>
<tr>
<td></td>
<td>More Cool stuff</td>
</tr>
</tbody>
</table>

---

S. Varrette (PCOG Research unit)
$> git rebase <branch>  # DANGER! Rewrites the tree

- Basic **(3-ways)** merging via `git merge` creates a new commit

(master)$> git merge experiment
Rebasing

```
$> git rebase <branch>   # DANGER! Rewrites the tree
```

- Basic (3-ways) merging via `git merge` creates a new commit

```
(master)$> git merge experiment
```

```
C0 -> C1 -> C2 -> C3 -> C4 -> C5
```

```
experiment -> C4
master
```
Rebasing: **Linear** alternative to merging

- create a patch of the introduced change (in C4)
- reapply it on top (of C3) to create C4'

```bash
$ git rebase <branch>  # DANGER! Rewrites the tree
```
Rebasing

- Rebasing: **Linear** alternative to merging
  - create a patch of the introduced change (in C4)
  - reapply it on top (of C3) to create C4'

```
(master)\$> git checkout experiment
(experiment)\$> git rebase master
```
Rebasing: Linear alternative to merging

- create a patch of the introduced change (in C4)
- reapply it on top (of C3) to create C4'

```
$> git rebase <branch>
```

# DANGER! Rewrites the tree

```
(experiment)$> git checkout master
(master)$> git merge experiment
```
Rebasing ensure your commits apply cleanly on a (remote) branch

Never rebase published code!
Advanced Git Topics

Summary

1. Introduction
   - Agenda
   - Overview of managed IT Infrastructure

2. IT/Dev[op]s Army Knives Tools
   - SSH Secure Shell
   - PGP / GPG: Gnu Privacy Guard
   - Vagrant
   - Puppet
   - Ruby / Python / Markdown-based Documentations
   - Password Management

3. Research Computing Platform @ UL

4. Git[Lab] @ UL and VCS

5. Git Basics
   - Installing Git
   - Git theory
   - Basic Commands
   - Branching and Merging

6. Collaborating / Working together

7. Advanced Git Topics
   - Git Submodules
   - Rebasing
   - Using Git over Subversion Repository
   - More Cool stuff

S. Varrette (PCOG Research unit)
**Git-svn: using Git with Subversion**

```
$> git svn clone [-s] <svn-url>  # checkout an SVN repository
```

- `-s`: standard SVN layout with `trunk/`, `branches/` and `tags/`
- `clone into master` -- you **shall** work in another branch

```
$> git checkout -b work
```

→ `delegate all interactions with SVN repository with master`

→ `thus make all your (local) commits into the work branch`

---

S. Varrette (PCOG Research unit)
Git-svn: using Git with Subversion

$> \text{git svn clone [-s] <svn-url>} \quad \# \text{checkout an SVN repository}

- \textit{-s}: standard SVN layout with trunk/, branches/ and tags/
- \textit{clone into master} – you \textbf{shall} work in another branch \quad \textbf{Ex: work}
  
$> \text{git checkout -b work}

→ delegate all interactions with SVN repository with master
→ thus make all your (local) commits into the \textit{work} branch

$> \text{git svn rebase} \quad \# \text{fetch revisions from SVN and rebase}

- \textbf{Important}: always do that from the \textit{master} branch!
  
(work)$> \text{git checkout master}
(master)$> \text{git svn rebase}
$> \text{git svn dcommit} \quad \# \text{create an SVN revision for each commit}

\text{AFTER you sanitize the 'master' branch!}

(work)$> \text{git checkout master}

(master)$> \text{git svn rebase}

rebase the master branch with the SVN repository
**Git-svn: commit to Subversion**

```bash
$> git svn dcommit  # create an SVN revision for each commit
```

AFTER you sanitize the 'master' branch!

```
(master)$> git checkout work
(work)$> git rebase master
```

1. rebase the master branch with the SVN repository
2. go back to the work branch and rebase with master
Advanced Git Topics

Git-svn: commit to Subversion

$> git svn dcommit  # create an SVN revision for each commit

AFTER you sanitize the 'master' branch!

(work)$> git log -graph -oneline -decorate  # OR git gr

1. rebase the master branch with the SVN repository
2. go back to the work branch and rebase with master
3. ensure everything is fine
$>\text{git svn dcommit} \quad \# \text{create an SVN revision for each commit}$

\text{AFTER you sanitize the 'master' branch!}

(\text{work})$>\text{git checkout master}$

(\text{master})$>\text{git merge --no-ff work}$

1. rebase the master branch with the SVN repository
2. go back to the work branch and rebase with master
3. ensure everything is fine
4. force 3-ways merge your local commit
Git-svn: commit to Subversion

$> git svn dcommit  # create an SVN revision for each commit

AFTER you sanitize the 'master' branch!

(master)$> git commit -amend

1. rebase the master branch with the SVN repository
2. go back to the work branch and rebase with master
3. ensure everything is fine
4. force 3-ways merge your local commit
5. edit (amend) the last commit for your SVN dudes
$> \textit{git svn dcommit} \quad \# \textit{create an SVN revision for each commit}$

\textbf{AFTER} you sanitize the 'master' branch!

\begin{enumerate}
  \item rebase the master branch with the SVN repository
  \item go back to the work branch and rebase with master
  \item ensure everything is fine
  \item force 3-ways merge your local commit
  \item edit (\texttt{amend}) the last commit for your SVN dudes
  \item \textbf{Finally} commit on the SVN server
\end{enumerate}
Git-svn: commit to Subversion

$> git svn dcommit # create an SVN revision for each commit

AFTER you sanitize the 'master' branch!

(master)$> git checkout work

1. rebase the master branch with the SVN repository
2. go back to the work branch and rebase with master
3. ensure everything is fine
4. force 3-ways merge your local commit
5. edit (amend) the last commit for your SVN dudes
6. Finally commit on the SVN server
7. Go back to the ‘work’ branch!
Advanced Git Topics

Summary

1. Introduction
   - Agenda
   - Overview of managed IT Infrastructure

2. IT/Dev[op]s Army Knives Tools
   - SSH Secure Shell
   - PGP / GPG: Gnu Privacy Guard
   - Vagrant
   - Puppet
   - Ruby / Python / Markdown-based Documentations
   - Password Management

3. Research Computing Platform @ UL

4. Git[Lab] @ UL and VCS

5. Git Basics
   - Installing Git
   - Git theory
   - Basic Commands
   - Branching and Merging

6. Collaborating / Working together

7. Advanced Git Topics
   - Git Submodules
   - Rehashing
   - Using Git over Subversion Repository
   - More Cool stuff

Git[Lab] Around You
About Version Control System (VCS)

S. Varrette (PCOG Research unit)
Advanced Git Topics

Shell Integration

- Git Completion – Git flow completion

**Colored PS1**

- bash: integrate `__git_ps1()` function in your PS1 variable
  → normally part of the bash-completion package
  → See integration in the ULHPC/dotfiles repository
    ```
    $>
    export GIT_PS1_SHOWDIRTYSTATE=1
    # you probably want that
    ```
- zsh: agnoster theme / powerline
  → Mac OS instructions

- On **CentOS/Redhat**, you have to source the correct file
  ```
  $>
  ln -s /usr/share/git-core/contrib/completion/git-prompt.sh
  /etc/profile.d/
  ```
Revision Selection and Log Filtering

When referring to a commit `<commit>`:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;commit&gt;^</code></td>
<td>Parent of the commit <code>&lt;commit&gt;</code></td>
</tr>
<tr>
<td>HEAD<code>^</code></td>
<td>Previous commit (parent of HEAD)</td>
</tr>
<tr>
<td><code>&lt;commit&gt;~&lt;n&gt;</code></td>
<td><code>&lt;n&gt;</code>-th parent of <code>&lt;commit&gt;</code></td>
</tr>
</tbody>
</table>

```
$ git log --pretty=format:’%h %s’ --graph
* 734713b fixed refs handling, added gc auto, updated tests  # HEAD
* d921970 Merge commit 'phedders/rdocs'  # HEAD^  
|\  *
| * 35cfb2b Some rdoc changes  
| \  *
| * 1c002dd added some blame and merge stuff  # d921970^ OR HEAD~2
| /  *
| * 1c36188 ignore *.gem  # HEAD~3
```
Git offers visual diff/merge tools, assuming you configured it:

```bash
$> git config --global merge.tool sourcetree
```

```bash
$> git difftool [<commit>]
```

diff GUI
Git offers visual diff/merge tools, assuming you configured it:

```
$> git config --global merge.tool sourcetree
```

```
$> git difftool [<commit>]
```

`diff GUI`

```
$> git mergetool [<path>...]  # resolving merge conflicts
```

S. Varrette (PCOG Research unit)
External Merge and Diff Tools

- Git offers visual diff/merge tools, assuming you configured it:
  
  ```
  $> \text{git config --global merge.tool sourcetree}
  
  $> \text{git difftool [<commit>]} \quad \text{diff GUI}
  
  $> \text{git mergetool [<path>...]} \quad \text{# resolving merge conflicts}
  ```

- You can set up another graphical merge-conflict-resolution tool

  ```
  \rightarrow \text{List the available tools:} \quad \text{git mergetool --tool-help}
  
  \rightarrow \text{Mac OS:} \quad \text{git config --global merge.tool opendiff}
  
  \rightarrow \text{Linux:} \quad \text{git config --global merge.tool kdiff3}
  
  \rightarrow \text{Cross-platform: P4Merge (download)}
  ```

  ```
  $> \text{brew cask install p4merge} \quad \text{# on Mac OS, using Homebrew and Cask}
  ```
Using P4Merge as diff/merge tool

```bash
# Configure P4Merge as Git merge tool

git config --global merge.tool p4mergetool

git config --global mergetool.p4mergetool.trustexitcode false

git config --global mergetool.p4mergetool.keeptemporaries false

git config --global mergetool.p4mergetool.keepbackup false

git config --global mergetool.p4mergetool.cmd \
$BASE $LOCAL $REMOTE $MERGED
```

- **Alternatives** (mostly Mac OS)
  - Kaleidoscope
  - Araxis Merge
  - DeltaWalker
  - DiffMerge (free)
  - SourceTree (free)
Interesting Git plugins

- **Git-extra**: Additional GIT utilities
  - repo summary, repl, changelog population, author commit...

- **Git-crypt**: Transparent file encryption in git
  - file to automatically encrypt specified in `.gitattributes` file
  - beware of a hook to check status

```
$ git-crypt init # Initialize repo with YOUR GPG key
```
Interesting Git plugins

- **Git-extra**: Additional GIT utilities
  - repo summary, repl, changelog population, author commit...

- **Git-crypt**: Transparent file encryption in git
  - file to automatically encrypt specified in .gitattributes file
  - beware of a hook to check status

```
$> git-crypt init # Initialize repo with YOUR GPG key
```

```
$> git-crypt [un]lock # Lock/Unlock the files
```
You need also to enable a git pre-commit hook
to avoid accidentally adding unencrypted files – see issue #45.
Example of such a pre-commit hook: this gist – raw version
✓ to be placed as .git/hooks/pre-commit

**note:** these hooks are local to your working directory

```
$> curl <url/to/raw/gist> -o .git/hooks/pre-commit
$> chmod +x .git/hooks/pre-commit
```
Advanced Git Topics

Git-crypt

$> git-crypt add-gpg-user USER_ID    # Add (GPG) collaborator

$> git-crypt status                 # Status - raise WARNING on problem
$> git-crypt add-gpg-user USER_ID     # Add (GPG) collaborator

$> git-crypt status                # Status - raise WARNING on problem

# .gitattributes
# specify which files to encrypt using git-crypt
# see https://www.agwa.name/projects/git-crypt/

# Certificate private keys
*.key filter=git-crypt diff=git-crypt
# Host SSH private keys
*ssh_*_key filter=git-crypt diff=git-crypt
Other Cool Stuff

Stashing

- Move changes to a separate “stash”.

Interactive Rebase

$> git rebase -i <branch>

$> git stash
$> git stash pop
$> git stash list
$> git stash apply
$> git stash drop
$> git stash clear
**Stashing**

- Move changes to a separate “stash”.

**Interactive Rebase**

```bash
$> git rebase -i <branch>
```

**Git hooks:**

- Located in `.git/hooks/`
- Scripts run at various stages of Git operation
- Useful to perform lint actions for instance before pushing
Thank you for your attention...

Questions?

Sebastien Varrette
mail: Sebastien.Varrette@uni.lu
Office E-007
Campus Kirchberg
6, rue Coudenhove-Kalergi
L-1359 Luxembourg

1 Introduction
Agenda
Overview of managed IT Infrastructure

2 IT/Dev[op]s Army Knives Tools
SSH Secure Shell
PGP / GPG: Gnu Privacy Guard
Vagrant
Puppet
Ruby / Python / Markdown-based Documentations
Password Management

3 Research Computing Platform @ UL

4 Git[Lab] @ UL and VCS

5 Git Basics
Installing Git
Git theory
Basic Commands
Branching and Merging

6 Collaborating / Working together

7 Advanced Git Topics
Git Submodules
Rebasing
Using Git over Subversion Repository
More Cool stuff

Git[Lab] Around You
About Version Control System (VCS)

IT/Dev[op]s Army Knives Tools for the researcher