The E-ALE (Embedded Apprentice Linux Engineer) is a series of seminars held at existing conferences covering topics which are fundamental to a Linux professional in the field of Embedded Linux.

This seminar will spend equal time on lecture and hands on labs at the end of each seminar which allow you to practice the material you’ve learned.

This material makes the assumption that you have minimal experience with using Linux in general, and a basic understanding of general industry terms. The assumption is also made that you have access to your own computers upon which to practice this material.

More information can be found at https://e-ale.org/

This material is licensed under CC-BY SA4
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.7</td>
<td>Updating Files pt3</td>
<td>16</td>
</tr>
<tr>
<td>2.8</td>
<td>Labs</td>
<td>17</td>
</tr>
<tr>
<td>3</td>
<td>Working With Others</td>
<td>19</td>
</tr>
<tr>
<td>3.1</td>
<td>Branches</td>
<td>20</td>
</tr>
<tr>
<td>3.2</td>
<td>Making a Branch</td>
<td>21</td>
</tr>
<tr>
<td>3.3</td>
<td>Branches Flow</td>
<td>22</td>
</tr>
<tr>
<td>3.4</td>
<td>Updating From Upstream</td>
<td>23</td>
</tr>
<tr>
<td>3.5</td>
<td>Submitting Changes</td>
<td>24</td>
</tr>
<tr>
<td>3.6</td>
<td>Labs</td>
<td>25</td>
</tr>
</tbody>
</table>
Chapter 1

Getting Started

1.1 Git Concepts ................................................................. 2
1.2 Git Concepts pt2 ............................................................ 3
1.3 Git Concepts pt3 ............................................................ 4
1.4 Git’ing Started .............................................................. 5
1.5 Git Setup ................................................................. 6
1.6 Labs ................................................................. 7
1.1 Git Concepts

- Git is distributed version control system
- The tool has no concept of a central repository
- You make a copy of a git repo and work locally
- Use one of various method to submit changes to the remote repo
### 1.2 Git Concepts pt2

- The main unit in Git is a commit
- Each commit represents a snapshot of all the directories and files
- Internally it's optimized to save space
- You check-in (commit) logical changes set across multiple files as a single change set
  - e.g. changes in `/etc/xinetd.conf` & `/etc/firewalld` checked in together as a single commit to allow a new service to be used.
1.3 Git Concepts pt3

- Each commit has a parent commit
- Each commit has a unique id
- Can create branches
- Can fetch new commits from the remote repo without losing local changes
1.4 Git’ing Started

- To start, either clone an existing git repo or start fresh with an empty repo
- To clone:
  
  `git clone <remote git url>`
  
  - For example:
    `git clone git://myserver.com/machine-settings.git`

- To start fresh:
  
  `git init`
1.5 Git Setup

- If you are going to do more than just look at the code, you will need to setup git.

- Set your Identity

  $ git config user.name "Vedermi Importante"
  $ git config user.email vimport@goofymail.com

- To see your settings:

  `git config --list`

- When you create commits in git, it will use an editor for you to write a description for the commit. It uses your system default editor, but you can change it. For example:

  `git config core.editor vi`
1.6 Labs

Exercise 1.1: Initialize git

Create a directory for a small project and initialize git

$ mkdir ~/my-etc
$ cd ~/my-etc
$ git init
$ git config user.name "<your name>"
$ git config user.email <your email address>
Chapter 2

Managing File Changes

2.1 Creating Files .................................................. 10
2.2 Creating Files pt2 .................................................. 11
2.3 Adding Files .................................................. 12
2.4 Committing Files .................................................. 13
2.5 Updating Files .................................................. 14
2.6 Updating Files pt2 .................................................. 15
2.7 Updating Files pt3 .................................................. 16
2.8 Labs .................................................. 17
2.1 Creating Files

- General flow is as follows:
  1. create or modify files
  2. add the files to the staging area in git
  3. create a commit from the files in the staging area
2.2 Creating Files pt2

- `git status` will show which files have been added/modified in the directories on disk, and which are in the staging area. For example:

  ```
  $ git status
  On branch master

  No commits yet

  Untracked files:
  (use "git add <file>..." to include in what will be committed)

  before.init
  before.rules
  ufw.conf
  ```
2.3 Adding Files

- Files are added to the staging area with `git add`. For example:

  ```
  $ git add ufw.conf
  $ git status
  On branch master
  No commits yet

  Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

  new file:  ufw.conf

  Untracked files:
  (use "git add <file>..." to include in what will be committed)

  before.init
  before.rules
  ```
2.4 Committing Files

- Files in the staging area are placed into a new commit with `git commit`. For example:

  ```bash
  $ git add before.rules before.init
  $ git commit
  [master (root-commit) 009b5ed] Initial commit
  3 files changed, 125 insertions(+)
  create mode 100644 before.init
  create mode 100644 before.rules
  create mode 100644 ufw.conf
  
  $ git status
  On branch master
  nothing to commit, working tree clean
  
  $ git log
  commit 009b5edd3051740645aa2d8d20ca03e405171e1 (HEAD -> master)
  Author: John Bonesio <bones@bonesio.net>
  Date: Thu Aug 15 14:02:57 2019 -0700

  Initial commit
  ```
2.5 Updating Files

- To create a new commit containing new changes:
  1. edit the file(s)
  2. add the files to the staging area
  3. create a new commit from what's in the staging area

- Example:

  (edit files)
  $ git status
  On branch master
  Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

  modified:   ufw.conf
2.6 Updating Files pt2

- Example (cont’)

  $ git add ufw.conf
  $ git status
  On branch master
  Changes to be committed:
    (use "git reset HEAD <file>..." to unstage)

    modified:   ufw.conf

  $ git commit
  [master 708962b] Updated logging to medium
    1 file changed, 1 insertion(+), 1 deletion(-)

  $ git status
  On branch master
  nothing to commit, working tree clean
2.7 Updating Files pt3

- Example (cont’)

$ git log
commit 708962b7534e9b80aea00d989fcb77807d9bd678 (HEAD -> master)
Author: John Bonesio <bones@bonesio.net>
Date:   Thu Aug 15 14:06:43 2019 -0700

  Updated logging to medium

commit 009b5edda3051740645aa2d8d20ca03e405171e1
Author: John Bonesio <bones@bonesio.net>
Date:   Thu Aug 15 14:02:57 2019 -0700

  Initial commit
2.8 Labs

Exercise 2.1: Create Your First Commit

Store the settings for a

```
$ mdir ~/lab-settings
$ cd ~/lab-settings
$ cp /etc/hosts .
$ git add hosts
$ git commit
```

Exercise 2.2: Add a Host and Commit the Change

Add a host name to the hosts file. For example:
192.168.1.200 MediaBox

Add these changes to the staging area.

And then create a new commit.
Chapter 3

Working With Others

3.1 Branches .................................................. 20
3.2 Making a Branch .......................................... 21
3.3 Branches Flow ............................................. 22
3.4 Updating From Upstream ............................... 23
3.5 Submitting Changes ..................................... 24
3.6 Labs ......................................................... 25
3.1 Branches

- You can make branches for your own use
- Especially useful when working with others
- A branch is a label
- When you create a commit, the branch you are on (the label) advances to the new commit
- You must be on a branch to create new commits.
3.2 Making a Branch

- 2 Steps:
  1. Create branch
     `git branch mybranch`
  2. Switch to branch
     `git checkout mybranch`
- or both in 1 Step
  `git checkout -b mybranch`
3.3 Branches Flow

Branches Flow

initial state

1000

1001

master

1000

1001

master, mybranch

2003

mybranch

1001

master

1000

1002

master

2003

mybranch

git checkout -b mybranch

git commit

git checkout master

git commit
3.4 Updating From Upstream

- The remote repository (where you copied from), may have newer commits than your local repo
- To get these newer commits into your local repo:
  
git fetch
3.5 Submitting Changes

- Depends on how your project receives updates
  - submit patches to a mailing list
  - Tell remote admin to pull changes
  - Push changes to remote repo
3.6 Labs

Exercise 3.1: Create a Branch
Create a branch for your hostname repository. Then add a new host name (maybe a special one for a particular department or site). Add your changes and commit them.

1. $ git branch -b depA
2. Edit the file hosts to have your new hostname: e.g. 192.168.1.201 ProjectorA
3. $ git add hosts
4. $ git commit

Exercise 3.2: Add Changes to the Master Branch
Switch back to the master branch for your hostname repository. Then add a new host name

1. $ git checkout Master
2. Edit the file hosts to have your new hostname: e.g. 192.168.1.205 BackupServer
3. $ git add hosts
4. $ git commit