Procedimentos para Construção de Pacotes

Equipe de Gerenciamento da Coleção de Ports do FreeBSD

$FreeBSD: head/pt_BR.ISO8859-1/articles/portbuild/article.xml 41645 2013-05-17 18:49:52Z gabor $

$FreeBSD: head/pt_BR.ISO8859-1/articles/portbuild/article.xml 41645 2013-05-17 18:49:52Z gabor $

Índice

1. Introdução

Com o objetivo de disponibilizar binários pré-compilados de aplicações de terceiros para o FreeBSD, a Coleção de `Ports` é regularmente compilada em um dos "`Clusters` de Compilação de Pacotes". Atualmente o principal `cluster` em uso é o http://pointyhat.FreeBSD.org.

Este artigo documenta os trabalhos internos do `cluster`

> **Nota:** Muitos dos detalhes deste artigo serão do interesse apenas dos membros da equipe que faz o Gerenciamento da Coleção de `Ports` (http://www.FreeBSD.org/portmgr)

1.1. O código base

A maior parte da mágica na compilação de pacotes ocorre sob o diretório `/var/portbuild`. A menos que seja especificado o contrário, todos os caminhos serão relativos à este diretório. O *`${arch}`* será usado para determinar uma das arquiteturas de pacotes (amd64, i386™, ia64, powerpc, e SPARC64®), e *`${branch}`* será usado para determinar o `branch` (ramo de desenvolvimento) de compilação (7, 7-exp, 8, 8-exp, 9, 9-exp, 10, 10-exp).

> **Nota:** Não são mais compilados pacotes para as versões 4, 5 ou 6, e para a arquitetura alpha

Os `scripts` que controlam todo o processo estão localizados em `/var/portbuild/scripts/`. Eles são cópias obtidas do repositório Subversion `base/projects/portbuild/scripts/` (http://svnweb.freebsd.org/base/projects/portbuild/scripts/).

Normalmente são feitas compilações incrementais que usam pacotes anteriores como dependências; isso toma menos tempo, e coloca menos carga nos sites espelho. Normalmente são feitas compilações completas apenas quando:

- logo depois de uma nova versão, para o ramo `-STABLE`
- periodicamente, para testar mudanças realizadas no `-CURRENT`
- para compilações experimentais

1.2. Observações sobre o código base

Até meados de 2010, os `scripts` apontavam especificamente para `pointyhat` como o nó principal (dispatch). Durante o verão de 2010, mudanças significativas foram feitas a fim de aceitar outros `hosts` como nós principais. Entre estas mudanças estão:

- remoção da `string` `pointyhat` embutida no código
- fatoração de todas as constantes de configuração (que antes estavam espalhadas por todo o código) em arquivos de configuração (veja abaixo)
- adicionar o `hostname` aos diretórios especificados pelo `buildid` (isto vai permitir que os diretórios sejam inequívocos quando copiados entre máquinas.)
- tornar os `scripts` mais robustos em termos de criação de diretórios e `links` simbólicos
- se necessário, alterar a forma de execução dos `scripts` para tornar os itens acima mais fáceis.

Este documento foi escrito originalmente antes destas mudanças serem feitas. Nas partes em que algo foi modificado, como nas invocações de `scripts`, elas estão denotadas como `novo código base:` em oposição à `antigo código base:`.

> **Nota:** Como em dezembro de 2010, o `pointyhat` ainda está rodando sobre o antigo código base, até que o novo código base seja considerado estável.

> **Nota:** Também durante esse processo, o código base foi migrado para o repositório Subversion (http://svnweb.freebsd.org/base/projects/portbuild/scripts/). Para referência, a versão anterior ainda pode ser encontrada no CVS (http://www.freebsd.org/cgi/cvsweb.cgi/ports/Tools/portbuild/scripts/Attic/).

2. Gerenciamento dos Clientes de Compilação

Os clientes i386 localizados conjuntamente com o `pointyhat`, efetuam o `boot` via rede a partir dele (nós *`conectados`*); todos os outros clientes (nós *`desconectados`*) ou são auto-hospedados ou efetuam `boot` via rede a

partir de outro `host pxe`. Em todos os casos eles se auto configuram durante o `boot` preparando-se para compilar pacotes.

O `cluster` principal copia, através do `rsync`, os dados necessários (a árvore de `ports` e dos fontes, `bindist tarballs`, `scripts`, etc.) para os nós desconectados durante a fase de configuração dos nós. Em seguida, o diretório `portbuild` desconectado é montado como `nullfs` para compilações sob `chroot`.

O usuário `ports-${arch}` pode acessar os nós clientes através do ssh(1) para monitorá-los. Use o `sudo` e verifique o `portbuild.hostname.conf` para o usuário e detalhes do acesso.

O `script scripts/allgohans` pode ser usado para executar um comando em todos os clientes `${arch}`.

O `script scripts/checkmachines` é usado para monitorar a carga em todos os nós do `cluster` de compilação, e agendar quais nós compilarão quais `ports`. Este `script` não é muito robusto e tem uma tendência a morrer. É melhor iniciar este `script` no nó principal (por exemplo, `pointyhat`) depois do `boot` usando um `loop` com while(1).

3. Configuração do Ambiente de Compilação sob `Chroot`

A compilação de pacotes é realizada em um ambiente `chroot`, configurado pelo `script portbuild` usando o arquivo `${arch}/${branch}/builds/${buildid}/bindist.tar`.

O seguinte comando faz o `build world` a partir da árvore de diretórios em `${arch}/${branch}/builds/${buildid}/src/` e o instala em `${worlddir}`. A árvore de diretórios será atualizada primeiro, a menos que a opção `-nocvs` seja especificada.

```
/var/portbuild# scripts/makeworld ${arch} ${branch} ${buildid} [-nocvs]
```

O arquivo `bindist.tar` é criado a partir do `world`, instalado previamente, pelo `script mkbindist`. Este deve ser executado como `root` com o seguinte comando:

```
/var/portbuild# scripts/mkbindist ${arch} ${branch} ${buildid}
```

Os `tarballs` de cada máquina estão localizados em `${arch}/clients`.

O arquivo `bindist.tar` é extraído para cada cliente durante a inicialização dos mesmos, e no início de cada passagem do `script dopackages`.

3.1. Novo Código Base

Para ambos os comandos acima, se o `${buildid}` estiver definido como `latest`, ele pode ser omitido.

4. Customizando Sua Compilação

(O trecho a seguir aplica-se apenas ao novo código base.)

Você pode customizar sua compilação providenciando versões locais do `make.conf` e/ou `src.conf`, localizados em `${arch}/${branch}/builds/${buildid}/make.conf.server` e `${arch}/${branch}/builds/${buildid}/src.conf.server`, respectivamente. Estes serão usados, em vez dos arquivos padrões que estão no lado do servidor.

Da mesma forma, se você também quiser afetar o `make.conf` no *lado do cliente*, você pode usar o `*${arch}*/*${branch}*/builds/*${buildid}*/make.conf.client`.

> **Nota:** Devido ao fato de cada um dos clientes individuais poder ter seu próprio `make.conf`, o conteúdo do `*${arch}*/*${branch}*/builds/*${buildid}*/make.conf.client` vai ser *adicionado* ao `make.conf`, e não substituí-lo, como é feito com o `*${arch}*/*${branch}*/builds/*${buildid}*/make.conf.server`.

> **Nota:** Não existe nenhuma funcionalidade semelhante para `*${arch}*/*${branch}*/builds/*${buildid}*/src.conf.client` (e que efeito teria?).

Exemplo 1. Exemplo de `make.conf.*target*` para testar a nova versão padrão do ruby

(Neste caso, os conteúdos são idênticos para ambos, servidor e cliente.)

```
RUBY_DEFAULT_VER= 1.9
```

Exemplo 2. Exemplo de `make.conf.*target*` para compilação do clang

(Neste caso, os conteúdos também são idênticos para ambos, servidor e cliente.)

```
.if !defined(CC) || ${CC} == "cc"
CC=clang
.endif
.if !defined(CXX) || ${CXX} == "c++"
CXX=clang++
.endif
.if !defined(CPP) || ${CPP} == "cpp"
CPP=clang-cpp
.endif
# Don’t die on warnings
NO_WERROR=
WERROR=
```

Exemplo 3. Exemplo de `make.conf.server` para pkgng

```
WITH_PKGNG=yes
PKG_BIN=/usr/local/sbin/pkg
```

Exemplo 4. Exemplo de `make.conf.client` para pkgng

```
WITH_PKGNG=yes
```

Exemplo 5. Exemplo de `src.conf.server` para testar uma versão nova do código base do sort

```
WITH_BSD_SORT=yes
```

5. Iniciando a Compilação

Várias compilações separadas para cada arquitetura - a combinação de `branchs` é suportada. Todos os dados privados para uma compilação (árvore de `ports`, árvore do `src`, pacotes, `distfiles`, arquivos de `log`, `bindist`, `Makefile`, etc) estão localizados sob *`${arch}/${branch}`*`/builds/`*`${buildid}`*. Alternativamente, a última compilação pode ser referenciada sob o `buildid latest`, e a anterior a esta é chamada `previous`.

Novas compilações são clonadas a partir da `latest`, o que é rápido, uma vez que ele usa ZFS.

5.1. Os `Scripts dopackages`

Os `scripts scripts/dopackages` são usados para executar as compilações.

5.1.1. Código base antigo

Para o código base antigo, os mais úteis são:

- `dopackages.7` - Executa a compilação para a série 7.X
- `dopackages.7-exp` - Executa a compilação para a série 7.X com `patches` experimentais (`branch` 7-exp)
- `dopackages.8` - Executa a compilação para a série 8.X.
- `dopackages.8-exp` - Executa a compilação para a série 8.X com `patches` experimentais (`branch` 8-exp)
- `dopackages.9` - Executa a compilação para a série 9.X.
- `dopackages.9-exp` - Executa a compilação para a série 9.X com `patches` experimentais (`branch` 9-exp)
- `dopackages.10` - Executa a compilação para a série 10.X.
- `dopackages.10-exp` - Executa a compilação para a série 10.X com `patches` experimentais (`branch` 10-exp)

Esses são `wrappers` para o `dopackages` e todos são `links` simbólicos para `dopackages.wrapper`. `Wrappers` de `scripts` para um novo `branch` podem ser criados com `links` simbólicos `dopackages.${branch}` para `dopackages.wrapper`. Esses `scripts` tem uma série de argumentos. Por exemplo:

```
dopackages.7 ${arch} ${buildid} [-options]
```

5.1.2. Novo código base

Você pode usar o `dopackages.wrapper` diretamente, ao invés dos `links` simbólicos. Por exemplo:

```
dopackages.wrapper ${arch} ${branch} ${buildid} [-options]
```

5.1.3. Para ambos os códigos base

Frequentemente você usará `latest` como valor para o *`buildid`*.

`[-options]` pode ser nulo, uma ou mais, das opções seguintes:

- `-keep` - Não remove esta compilação no futuro, quando normalmente seria removido como parte do ciclo `latest - previous`. Não se esqueça de efetuar a limpeza manualmente quando ele não for mais necessário.

- `-nofinish` - Não executa o pós-processamento após finalizar a compilação. Isto é útil se você espera que a compilação precise ser reiniciada depois de concluída. Se você usar esta opção, não se esqueça de limpar os clientes quando você não precisar mais da compilação.
- `-finish` - Executa apenas o pós-processamento.
- `-nocleanup` - Por padrão, quando o estágio `-finish` da compilação é completado, os dados da compilação serão removidos dos clientes. Esta opção vai evitar a remoção dos dados.
- `-restart` - Reinicia uma compilação interrompida (ou não finalizada) a partir do começo. Os `Ports` que falharam na compilação anterior serão recompilados.
- `-continue` - Reinicia uma compilação interrompida (ou não finalizada). Os `Ports` que falharam na compilação anterior não serão recompilados.
- `-incremental` - Compara os campos importantes do novo `INDEX` com a versão anterior, remove pacotes e arquivos de `log` dos `ports` antigos que foram alterados, e recompila o resto. Isso reduz o tempo de compilação substancialmente, pois os `ports` inalterados não serão recompilados todas as vezes.
- `-cdrom` - O empacotamento desta compilação será usado em um CD-ROM, então os pacotes marcados como `NO_CDROM` e os `disfiles` deverão ser removidos no pós-processamento.
- `-nobuild` - executa todas as etapas do pré-processamento, mas não a compilação dos pacotes.
- `-noindex` - Não reconstrói o `INDEX` durante o pré-processamento.
- `-noduds` - Não reconstrói o arquivo `duds` (`ports` que nunca são compilados, como por exemplo, aqueles marcados com `IGNORE`, `NO_PACKAGE`, etc.) durante o pré-processamento.
- `-nochecksubdirs` - Não verifica o `SUBDIRS` para os `ports` que não estão ligados à compilação. (Apenas para o novo código base).
- `-trybroken` - Tenta compilar `ports` marcados como `BROKEN` (desativado por padrão, pois os `clusters` amd64/i386 agora são suficientemente rápidos e quando fazem compilações incrementais eles gastam muito mais tempo do que o necessário para compilar tudo. Por outro lado, os outros `clusters` são bastante lentos, e seria um desperdício de tempo tentar compilar `ports` marcados como `BROKEN`).

  > **Nota:** Com `-trybroken`, provavelmente você também vai querer usar `-fetch-original` (e, no novo código base, `-unlimited-errors`).

- `-nosrc` - Não atualiza a árvore do `src` a partir do `snapshot` do ZFS, mantendo a árvore da compilação anterior.
- `-srccvs` - Não atualiza a árvore do `src` a partir do `snapshot` do ZFS, em vez disso ela é atualizada com o `cvs update`.
- `-noports` - Não atualiza a árvore de `ports` a partir do `snapshot` do ZFS, mantendo a árvore da compilação anterior.
- `-portscvs` - Não atualiza a árvore de `ports` a partir do `snapshot` do ZFS, em vez disso ela é atualizada com o `cvs update`.
- `-norestr` - Não tenta compilar `ports` marcados como `RESTRICTED`.
- `-noplistcheck` - Não considera como erro `ports` deixarem arquivos para trás ao serem removidos.

- `-nodistfiles` - Não coleta os `distfiles` que passarem no `make checksum` para depois fazer o *upload* para o `ftp-master`.
- `-fetch-original` - Baixa o `distfile` a partir do `MASTER_SITES` original, em vez do `ftp-master`.
- `-unlimited-errors` (apenas no novo código base) - anula a verificação de limites do qmanager para compilações descontroladas. Você pode querer isso principalmente quando usar `-restart` em uma compilação que provavelmente vai falhar, ou talvez quando executar `-trybroken`. A A limitação é realizada por padrão.

A menos que você especifique `-restart`, `-continue`, ou `-finish`, os `links` simbólicos para as compilações existentes serão rotacionados. Isto é, o `link` simbólico para `previous` será removido; a compilação mais recente terá seu `link` modificado para `previous/`; e a nova compilação será criada e referenciada com um `link` em `latest/`.

Se a última compilação finalizou de forma limpa, você não precisa remover nada. Se ela foi interrompida, ou você usou a opção `-nocleanup`, você precisa limpar os clientes executando:

```
build cleanup ${arch} ${branch} ${buildid} -full
```

Os diretórios `errors/`, `logs/`, `packages/`, e assim por diante, são limpos pelos `scripts`. Se você está com pouco espaço, também pode limpar o `ports/distfiles/`. Não altere o diretório `latest/`; ele é um link simbólico para o servidor web.

> **Nota:** O `dosetupnodes` supostamente é executado pelo `script dopackages` no caso de `-restart`, mas pode ser uma boa idéia executá-lo manualmente e depois verificar se todos os clientes tem a carga de trabalho esperada. Algumas vezes `dosetupnode` não pode limpar uma compilação e você precisará fazer isso manualmente. (Isto é um defeito.)

Verifique se a compilação de pacotes para a arquitetura *`${arch}`* está executando como usuário ports-*`${arch}`* ou ele apresentará um grande número de erros.

> **Nota:** Atualmente, a própria compilação de pacotes ocorre em duas fases idênticas. A razão para isso é que, algumas vezes, problemas temporários (por exemplo, falhas do NFS, sites FTP inalcançáveis, etc.) podem quebrar a compilação. Realizar o processo em duas fases é uma solução alternativa para esse tipo de problema.

Seja cuidadoso com `ports/Makefile` para não especificar qualquer diretório vazio. Isso é especialmente importante se você está realizando uma compilação com `patches` experimentais (-exp). Se o processo de compilação encontrar um diretório vazio, ambas as fases de compilação irão parar rapidamente, e um erro similar ao seguinte será adicionado para *`${arch}/${branch}`*`/make.[0|1]`:

```
don't know how to make dns-all(continuing)
```

Para corrigir este problema, simplesmente comente ou remova as entradas `SUBDIR` que apontam para subdiretórios vazios. Depois de feito isso, você pode reiniciar a compilação executando o comando `dopackages` adequado com a opção `-restart`.

> **Nota:** Este problema também ocorre se você criar uma nova categoria com um `Makefile` sem entradas `SUBDIR`s nele. Isso é, provavelmente, um defeito.

Exemplo 6. Atualize a árvore i386-7 e faça uma compilação completa

```
dopackages.7 i386 -nosrc -norestr -nofinish
dopackages.wrapper i386 7 -nosrc -norestr -nofinish
```

Exemplo 7. Reinicie uma compilação para amd64-8 interrompida sem atualizar

```
dopackages.8 amd64 -nosrc -noports -norestr -continue -noindex -noduds -nofinish
dopackages.wrapper amd64 8 -nosrc -noports -norestr -continue -noindex -noduds
-nofinish
```

Exemplo 8. Realize o pós-processamento de uma árvore sparc64-7 concluída

```
dopackages.7 sparc64 -finish
dopackages.wrapper sparc64 7 -finish
```

Dica: geralmente é melhor executar o comando `dopackages` dentro do `screen(1)`.

5.2. O comando `build`

Você pode precisar manipular os dados da compilação antes de inicia-la, especialmente para compilações experimentais. Isto é feito com o comando `build`. Aqui estão algumas opções úteis para criação:

- `build create` *`arch branch`* [*`newid`*] - Cria um *`newid`* (ou um `datestamp`, se não for especificado). Só é necessário quando da criação de um novo `branch` ou uma nova arquitetura. (TODO: documentar se `newid` deve ser especificado como `latest` no novo código base.)
- `build clone` *`arch branch oldid`* [*`newid`*] - Cria um clone do *`oldid`* para o *`newid`* (ou um `datestamp`, se não for especificado).
- `build srcupdate` *`arch branch buildid`* - Substitui a árvore `src` com um novo `snapshot` do ZFS. Não se esqueça de usar a opção `-nosrc` quando executar o `dopackages` mais tarde!
- `build portsupdate` *`arch branch buildid`* - Substitui a árvore de `ports` com um novo `snapshot` do ZFS. Não se esqueça de usar a opção `-noports` quando executar `dopackages` mais tarde!

5.3. Compilando um único pacote

Algumas vezes é necessário recompilar um único pacote a partir do conjunto de pacotes. Isso pode ser feito executando o seguinte comando:

`path``/qmanager/packagebuild` *`amd64 7-exp 20080904212103 aclock-0.2.3_2.tbz`*

6. Anatomia de uma compilação

Uma compilação completa, sem qualquer opção `-no` que desabilite as opções padrões, executa as seguintes operações na ordem especificada:

1. Atualiza a árvore de `ports` atual a partir de um `snapshot` do ZFS [*]
2. Atualiza o `branch` usado na árvore `src` a partir de um `snapshot` do ZFS [*]
3. Verifica se `ports` não têm uma entrada `SUBDIR` no `Makefile` de suas respectivas categorias [*]
4. Cria o arquivo `duds`, que é uma lista de `ports` que não precisam ser compilados [*] [+]
5. Cria um arquivo `INDEX` atualizado [*] [+]
6. Define os nós que serão usados na compilação [*] [+]
7. Compila uma lista de `ports` restritos [*] [+]
8. Compila os pacotes (fase 1) [++]
9. Executa outra configuração do nó [+]
10. Compila os pacotes (fase 2) [++]

[*] O status destes passos pode ser encontrado em *`${arch}/${branch}`*`/build.log`, bem como no `stderr` do `tty` onde o comando `dopackages` está rodando.

[+] Se qualquer destes passos falhar, a compilação será encerrada.

[++] O status destes passos pode ser encontrado em *`${arch}/${branch}`*`/make` (antigo código base) ou *`${arch}/${branch}`*`/journal` (novo código base). `Ports` individuais irão escrever seus `logs` de compilação em *`${arch}/${branch}`*`/logs` e os seus `logs` de erros em *`${arch}/${branch}`*`/errors`.

Anteriormente, a árvore `docs` também era verificada, no entanto, isso se mostrou desnecessário.

7. Manutenção da Compilação

Existem vários casos onde você precisará limpar manualmente uma compilação:

1. Você a interrompeu manualmente.
2. O `pointyhat` foi reiniciado enquanto uma compilação estava executando.
3. O `qmanager` falhou e reiniciado

7.1. Interrompendo uma Compilação

O processo para interromper de forma manual uma compilação é um tanto quanto confuso. Primeiro você precisa identificar o `tty` em que ela está sendo executada rodando (ou lembrando-se da saída do tty(1) quando você iniciou a compilação, ou usando `ps x` para identificá-lo). Você precisa certificar-se de que não existe mais nada importante rodando neste `tty`, você pode verificar isto executando o comando `ps`, por exemplo, `ps -t p1` lista os processos em execução no tty 1. Se não existir mais nada importante, você pode encerrar o terminal facilmente com `pkill -t pts/1`; ou pode utilizar o `kill -HUP`, por exemplo, `ps -t pts/1 -o pid= | xargs kill -HUP`. Você deve Substitur o *`p1`* pelo `tty` utilizado na compilação.

A compilação de pacote enviada pelo `make` para as máquinas clientes irá se auto limpar após alguns minutos (verifique com `ps x` até que todos finalizem).

Se você não encerrar o make(1), ele irá iniciar novas tarefas de compilação. Se você não encerrar o `dopackages` ele irá reiniciar toda a compilação. Se você não encerrar os processos `pdispatch`, eles irão continuar (ou reaparecer) até concluir a compilação do pacote.

7.2. Limpando uma Compilação

Para liberar recursos, você precisa limpar as máquinas clientes executando o comando `build cleanup`. Por exemplo:

```
% /var/portbuild/scripts/build cleanup i386 8-exp 20080714120411 -full
```

Se você esquecer de fazer isso, então os `chroots` da compilação antiga não serão limpos nas próximas 24 horas, e nenhum novo trabalho será iniciado no seu lugar enquanto o `pointyhat` achar que esta máquina ainda está ocupada.

Para verificar, utilize o comando `cat ~/loads/*` para mostrar o status das máquinas clientes; a primeira coluna é o número de trabalhos que ela pensa estar executando, e isso pode estar bem próximo da carga média. O `loads` é atualizado a cada 2 minutos. Se você executar um `ps x | grep pdispatch` e ele listar menos trabalhos do que os que o `loads` pensa estarem em uso, você está em apuros.

Você pode ter problemas com instâncias do comando `umount` ficando congeladas. Se isto ocorrer, você terá que usar o `script allgohans` para executar um comando ssh(1) em todos os clientes deste ambiente de compilação. Por exemplo:

```
ssh -l root gohan24 df
```

Vai lhe dar um `df`, e

```
allgohans "umount -f pointyhat.freebsd.org:/var/portbuild/i386/8-exp/ports"
allgohans "umount -f pointyhat.freebsd.org:/var/portbuild/i386/8-exp/src"
```

Supostamente irá resolver o problema dos `mounts` que não foram desconectados pelo `umount`. Você terá que continuar executando-os pois podem existir diversas montagens.

> **Nota:** Ignore o seguinte:
>
> ```
> umount: pointyhat.freebsd.org:/var/portbuild/i386/8-exp/ports: statfs: No such file or directory
> umount: pointyhat.freebsd.org:/var/portbuild/i386/8-exp/ports: unknown file system
> umount: Cleanup of /x/tmp/8-exp/chroot/53837/compat/linux/proc failed!
> /x/tmp/8-exp/chroot/53837/compat/linux/proc: not a file system root directory
> ```
>
> Os dois primeiros significam que o cliente não tinha o sistema de arquivos montado; os dois últimos são um defeito.
>
> Você também poderá ver mensagens sobre o `procfs`.

Após concluir tudo que foi exposto acima, remova o arquivo *`${arch}`*`/lock` antes de tentar reiniciar a compilação. Se você não o fizer, o `dopackages` simplesmente será encerrado.

Se você atualizou a árvore de `ports` antes de reiniciar, você pode precisar reconstruir o `duds`, o `INDEX`, ou ambos os arquivos.

7.3. Manutenção de compilações com o comando `build`

Aqui está o resto das opções para o comando `build`:

- `build destroy` *`arch branch`* - Destrói o `id` da compilação.
- `build list` *`arch branch`* - Mostra o conjunto atual de `ids` de compilações.
- `build upload` *`arch branch`* - ainda não implementado.

8. Monitorando a Compilação

Você pode usar o comando `qclient` para monitorar o status dos nós de compilação, e para listar as tarefas de compilação agendadas para execução:

`python` *`path`*`/qmanager/qclient jobs`

`python` *`path`*`/qmanager/qclient status`

O comando `scripts/stats` *`${branch}`* mostra o número de pacotes cuja compilação já finalizou.

A execução de um `cat /var/portbuild/*/loads/*` irá mostrar o `load` nos clientes e o número de compilações simultâneas em andamento. Os arquivos que foram atualizados recentemente correspondem aos clientes que estão `online`; os demais arquivos são dos clientes que estão `offline`.

> **Nota:** O comando `pdispatch` faz o envio de trabalhos para o cliente, e executa tarefas de pós-processamento a partir do retorno recebido do client. O `ptimeout.host` monitora permanentemente o processo de compilação e a encerra após a ocorrência de `timeouts`. Desta forma, se você tiver 50 processos `pdispatch`, mas apenas 4 processos ssh(1), significa que 46 processos `pdispatch`es estão ociosos, esperando que um nó fique livre.

Executar `tail -f` *`${arch}/${branch}`*`/build.log` irá mostrar o progresso geral da compilação.

Se a compilação do `port` falhar, e o motivo não ficar imediatamente óbvio a partir da análise do `log`, você pode preservar o `WRKDIR` para uma análise mais aprofundada. Para fazer isso, crie um arquivo vazio chamado `.keep` no diretório do `port`, isso vai arquivar, comprimir, e copiar o `WRKDIR` para *`${arch}/${branch}`*`/wrkdirs`.

Se você verificar que o sistema está compilando o mesmo pacote de forma ciclica, repetindo o processo indefinidamente, você pode ser capaz de corrigir o problema reconstruindo manualmente o pacote ofensor.

Se todas as compilações iniciam reclamando de que não pode carregar os pacotes dos quais ela depende, verifique se o **httpd** ainda está rodando, e o i reinicie se não estiver.

Mantenha o olho na saída do df(1). Se o sistema de arquivos do `/var/portbuild` ficar cheio, coisas ruins acontecem.

O status de todas as compilações é gerado duas vezes por hora e postado em http://pointyhat.FreeBSD.org/errorlogs/packagestats.html. Para cada `buildenv` é apresentado o seguinte:

- `cvs date` é o conteúdo do `cvsdone`. É por isso que nós recomendamos que você atualize o `cvsdone` para executar compilações experimentais, `-exp` (veja abaixo).
- data do último `log` (`latest log`)
- número de linhas no `INDEX`
- o número atual de `logs` de compilações (`build logs`)
- o número de pacotes concluídos (`packages`)
- o número de erros (`errors`)
- o número de `ports` ignorados (duds) (listados como `skipped`)
- A coluna `missing` mostra a diferença entre o `INDEX` e as outras colunas. Se você reiniciou uma compilação após um `cvs update`, provavelmente haverá duplicatas nos pacotes e colunas de erros, e esta coluna será inútil. (O `script` é ingênuo).
- Os valores das colunas `running` e `completed` são palpites baseados em um grep(1) do `build.log`.

9. Lidando com Erros de Compilação

A maneira mais fácil de rastrear falhas na compilação é receber os `logs` enviados por e-mail e organizá-los em uma pasta, assim você pode manter uma lista com as falhas atuais e detectar facilmente as novas. Para fazer isto, adicione um endereço de e-mail ao *`${branch}`*`/portbuild.conf`. Você pode encaminhar facilmente os novos erros para os mantenedores.

Quando um `port` passa a não compilar corretamente durante varios ciclos de compilação seguidos, é hora de marcá-lo como quebrado (`BROKEN`). Recomenda-se notificar os mantenedores durante duas semanas, antes de fazê-lo.

> **Nota:** Para evitar erros de compilação dos `ports` cujo código fonte precisa ser baixado manualmente, coloque os `distfiles` em `~ftp/pub/FreeBSD/distfiles`. Restrições de acesso foram implementadas para garantir que apenas os clientes de compilação tenham acesso a este diretório.

10. Compilando Pacotes para uma Versão Específica

Ao compilar pacotes para uma versão específica do FreeBSD, pode ser necessário atualizar manualmente as árvores do `ports` e do `src` para a `tag` da versão desejada e usar as opções `-nocvs` e `-noportscvs`.

Para compilar conjuntos de pacotes que serão usados em um CD-ROM, use a opção `-cdrom` para o comando `dopackages`.

Se não houver espaço em disco disponível no `cluster`, use `-nodistfiles` para que os `distfiles` não sejam baixados.

Após completar a compilação inicial, reinicie a compilação com `-restart -fetch-original` para baixar os `distfiles` atualizados. Então, uma vez que a compilação tiver sido pós-processada, faça um inventário da lista de arquivos baixados:

```
% cd ${arch}/${branch}
% find distfiles > distfiles-${release}
```

Este arquivo de inventário normalmente fica localizado em `i386/${branch}` no nó principal do `cluster`.

Isto é útil para ajudar na limpeza periódica dos `distfiles` do `ftp-master`. Quando o espaço se torna escasso, os `distfiles` das versões recentes podem ser mantidos, enquanto outros podem ser jogados fora.

Uma vez que o *upload* dos `distfiles` tenha sido feito (veja abaixo), o conjunto de pacotes da versão final deve ser criado. Para se assegurar, execute manualmente o `script` `${arch}/${branch}/cdrom.sh` para certificar-se de que todos os pacotes com distribuição restrita via CD-ROM e todos os `distfiles` foram removidos. Então, copie o diretório `${arch}/${branch}/packages` para `${arch}/${branch}/packages-${release}`. Uma vez que os pacotes tenham sido movidos com segurança, contate o Time de engenharia de Lançamento <`re@FreeBSD.org`> e informe-os da localização dos pacotes do `release`.

Lembre-se de coordenar com o Time de engenharia de Lançamento <`re@FreeBSD.org`> sobre o `timing` e o status das compilações do `release`.

11. *Upload* dos Pacotes

Uma vez que a compilação tenha terminado, os pacotes e/ou `distfiles` podem ser transferidos para o `ftp-master` para serem propagados para a rede de espelhos FTP. Se a compilação foi executada com a opção `-nofinish`, então certifique-se de executar em seguida o comando `dopackages -finish` para realizar o pós-processamento dos pacotes (remover pacotes marcados como `RESTRICTED` ou como `NO_CDROM` onde for apropriado, remover pacotes não listados no `INDEX`, remover do `INDEX` as referências para pacotes não compilados, e gerar um sumário `CHECKSUM.MD5`); e dos `distfiles` (movê-los do diretório temporário `distfiles/.pbtmp` para o diretório `distfiles/` e remover os `distfiles` marcados como `RESTRICTED` e `NO_CDROM`).

É recomendado que se execute manualmente os `scripts` `restricted.sh` e/ou `cdrom.sh` após a finalização do `dopackages`, apenas por segurança. Execute o `script` `restricted.sh` antes de fazer o *upload* para o `ftp-master`, em seguida, execute `cdrom.sh` antes de preparar o conjunto final de pacotes para um `release`.

Os subdiretórios de pacotes são nomeados de acordo com a versão e `branch` ao qual se destinam. Por exemplo:

- `packages-7.2-release`
- `packages-7-stable`
- `packages-8-stable`
- `packages-9-stable`
- `packages-10-current`

> **Nota:** Alguns destes diretórios no `ftp-master` são na verdade `links` simbólicos. Por exemplo:
>
> - `packages-stable`
> - `packages-current`
>
> Certifique-se de que você está movendo os novos pacotes para um diretório de destino *real*, e não para um dos `links` simbólicos que apontam para ele.

Se você está preparando um conjunto de pacotes completamente novo (por exemplo, para um novo `release`), copie os pacotes para a área de teste do `ftp-master` executando algo como mostrado abaixo:

```
# cd /var/portbuild/${arch}/${branch}
# tar cfv - packages/ | ssh portmgr@ftp-master tar xfC - w/ports/${arch}/tmp/${subdir}
```

Em seguida, entre no `ftp-master` e verifique se o conjunto de pacotes foi transferido com sucesso, remova o conjunto de pacotes que o novo conjunto vai substituir (em `~/w/ports/${arch}`), e mova o novo conjunto para o local. (`w/` é apenas um atalho.)

Para compilações incrementais, o *upload* deve ser feito usando o `rsync` para não colocar muita pressão nos espelhos.

SEMPRE use o `rsync` primeiro com a opção `-n` e verifique a saída do comando para assegurar-se que não existem problemas. Se parece estar tudo bem, execute novamente o `rsync` sem a opção `-n`.

Exemplo de sintaxe do comando `rsync` para o *upload* incremental de pacotes:

```
# rsync -n -r -v -l -t -p --delete packages/ portmgr@ftp-master:w/ports/${arch}/${subdir}/ | tee log
```

Os `distfiles` devem ser transferidos utilizando-se o `script` `cpdistfiles`:

```
# /var/portbuild/scripts/cpdistfiles ${arch} ${branch} ${buildid} [-yesreally] | tee log2
```

A execução manual deste processo é um procedimento obsoleto.

12. Compilação de `Patches` Experimentais

Compilações de `patches` experimentais são executadas de tempos em tempos para novas funções ou correções de defeitos na infraestrutura do `ports` (isto é, `bsd.port.mk`), ou para testar atualizações em grande escala. A qualquer momento podem haver vários `patches` de `branchs` experimentais simultâneos, como o `8-exp` na arquitetura amd64.

Geralmente, a compilação de `patches` experimentais é executada da mesma forma que qualquer outra compilação, exceto que você deve primeiro atualizar a árvore de `ports` para a última versão e, em seguida, aplicar os seus `patches`. Para fazer o primeiro, você pode usar o seguinte:

```
% cvs -R update -dP > update.out
% date > cvsdone
```

Essa é a simulação mais próxima do que o `script` `dopackages` faz. (Embora o `cvsdone` seja meramente informativo, ele pode ser útil.)

Você precisará editar o `update.out` para procurar por linhas que comecem com `^M`, `^C`, ou `^?` para que possa corrigi-las.

É sempre uma boa idéia salvar cópias do original de todos os arquivos modificados, bem como uma lista do que você está modificando. Você pode consultar a lista ao fazer o `commit` final, para se certificar de que você está realizando o `commit` exatamente daquilo que testou.

Pelo fato da máquina ser compartilhada, alguém pode excluir suas alterações por engano, então mantenha cópias destas, por exemplo, no seu diretório `home` freefall `freefall`. Não use o `tmp/`; pois a `pointyhat` executa ele mesmo alguma versão do `-CURRENT`, você pode esperar por reinicializações (no mínimo para atualizações).

Para que você tenha uma compilação de controle com a qual possa comparar eventuais falhas, você deve primeiro executar a compilação de pacote no `branch` em que os `patches` experimentais foram baseados para a arquitetura i386 (atualmente esta é o `8`). Quando estiver preparando a compilação dos `patches` experimentais, faça o `checkout`

da árvore do `ports` e do `src` com a mesma data da que foi usada para a compilação de controle. Isso vai garantir uma comparação válida entre as compilações depois.

Uma vez terminada a compilação, compare as falhas da compilação de controle com as da compilação dos `patches` experimentais. Para facilitar, use os seguintes comandos (assumindo o `branch 8` como `branch` de controle, e o `8-exp` como `branch` experimental):

```
% cd /var/portbuild/i386/8-exp/errors
% find . -name \*.log\* | sort > /tmp/8-exp-errs
% cd /var/portbuild/i386/8/errors
% find . -name \*.log\* | sort > /tmp/8-errs
```

> **Nota:** Se já faz muito tempo desde que a última compilação foi finalizada, os `logs` podem ter sido compactados automaticamente com **bzip2**. Nesse caso você deve usar `sort | sed 's,\.bz2,,g'` em seu lugar.

```
% comm -3 /tmp/8-errs /tmp/8-exp-errs | less
```

Este último comando vai gerar um relatório com duas colunas. A primeira coluna contém os `ports` que falharam na compilação de controle, mas não na compilação com `patches` experimentais; a segunda é o inverso As razões para o `port` estar na primeira coluna incluem:

- O `port` foi corrigido desde que a compilação de controle foi executada, ou foi atualizado para uma nova versão que também está quebrada (assim a nova versão também deve aparecer na segunda coluna)
- O `port` foi corrigido pelos `patches` experimentais na compilação experimental
- O `port` não foi compilado na compilação com `patches` experimentais devido a falha de uma dependência

Razões para o `port` aparecer na segunda coluna incluem:

- O `port` foi quebrado pelos `patches` experimentais [1]
- O `port` foi atualizado desde a compilação de controle e deixou de compilar [2]
- O `port` foi quebrado devido a um erro temporário (por exemplo, site FTP fora do ar, erro do pacote cliente, etc.)

Ambas as colunas devem ser investigadas e as razões para os erros entendidas antes do `commit` do conjunto de `patches` experimentais. Para diferenciar entre o [1] e o [2] acima, você pode recompilar os pacotes afetados sob o `branch` de controle:

```
% cd /var/portbuild/i386/8/ports
```

> **Nota:** Certifique-se de atualizar esta árvore com o `cvs update` para a mesma data da árvore dos `patches` experimentais.

O seguinte comando vai configurar o `branch` de controle para a compilação parcial (antigo código base):

```
% /var/portbuild/scripts/dopackages.8 -noportscvs -nobuild -nocvs -nofinish
```

As compilações devem ser executadas a partir do diretório `packages/All`. Este diretório deve estar vazio inicialmente, exceto pelo `link` simbólico do Makefile. Se este `link` simbólico não existir, ele deve ser criado:

```
% cd /var/portbuild/i386/8/packages/All
% ln -sf ../../Makefile .
% make -k -j<#> <list of packages to build>
```

> **Nota:** O <#> é o número de compilações paralelas para tentar. Normalmente isso é a soma dos pesos listados em `/var/portbuild/i386/mlist`, a menos que você tenha uma razão para executar uma compilação mais pesada ou leve.
>
> A lista de pacotes para compilar deve ser uma lista do nome do pacote (incluindo as versões) como aparece no `INDEX`. O `PKGSUFFIX` (isto é, .tgz ou .tbz) é opcional.

Isto vai compilar apenas os pacotes listados, bem como todas as suas dependências.

Você pode verificar o progresso da compilação parcial da mesma forma que você faria com uma compilação normal.

Uma vez que todos os erros tenham sido resolvidos, você pode efetuar o `commit` do conjunto de pacotes. Após efetuar o `commit`, é de costume enviar um e-mail para ports@FreeBSD.org (mailto:ports@FreeBSD.org) e com cópia para ports-developers@FreeBSD.org (mailto:ports-developers@FreeBSD.org), informando as pessoas sobre as mudanças. Um resumo de todas as mudanças também deve registrado no arquivo `/usr/ports/CHANGES`.

13. Como configurar um novo nó de compilação de pacotes

Antes de seguir estes passos, por favor, converse com o `portmgr`.

> **Nota:** Devido à algumas doações generosas, o `portmgr` não está mais procurando por empréstimos de sistemas i386 ou `amd64`. No entanto, nós ainda estamos interessados no empréstimo de sistemas `tier-2`.

13.1. Requisitos do nó

O `portmgr` ainda está trabalhando para definir quais são características que um nó necessita possuir para ser útil.

- Capacidade de CPU: qualquer coisa abaixo de 500MHz geralmente não é útil para a compilação de pacotes.

  > **Nota:** Nós somos capazes de ajustar o número de tarefas enviadas para cada máquina, e nós geralmente ajustamos o número para fazer uso de 100% da CPU.

- RAM: O mínimo utilizável é 2G; o ideal é ter 8G ou mais. Normalmente configuramos uma tarefa para cada 512M de RAM.
- Disco: É necessário um mínimo de 20G para o sistema de arquivos e de 32G para a area de `swap`. O desempenho será melhor se múltiplos discos forem utilizados, e configurados como `geom stripes`. Os dados de desempenho também estão em fase de definição.

Nota: A compilação de pacotes irá estressar as unidades de disco até o seu limite (ou além dele). Esteja consciente do que você está se voluntariando para fazer!

- largura de banda de rede: Ainda não existe um estudo preciso, no entanto uma máquina configurada para 8 tarefas simultâneas se mostrou capaz de saturar um link de internet a cabo.

13.2. Preparação

1. Escolha um `hostname` único. Ele não tem que ser um `hostname` resolvível publicamente (ele pode ser um nome em sua rede interna).
2. Por padrão, a compilação de pacotes necessita que as seguintes portas TCP estejam acessíveis: 22 (`ssh`), 414 (`infoseek`), e 8649 (`ganglia`). Se estas não estiverem acessíveis, escolha outras e assegure-se de que um túnel `ssh` esteja configurado (veja abaixo).

   (Nota: se você tem mais de uma máquina em seu site, você vai precisar de uma porta TCP individual para cada serviço em cada máquina, desta forma serão necessários túneis `ssh`. Portanto, você provavelmente precisará configurar o redirecionamento de portas em seu `firewall`.)
3. Decida se você vai inicializar localmente ou via `pxeboot`. Você vai descobrir que é mais fácil acompanhar as mudanças do `-current` com a última opção, especialmente se você tem várias máquinas em seu site.
4. Escolha um diretório para manter as configurações dos `ports` e os subdiretórios do `chroot`. Pode ser melhor colocá-los em uma partição dedicada. (Por exemplo: `/usr2/`.)

13.3. Configurando o `src`

1. Crie um diretório para armazenar a árvore dos fontes do último `-current` e sincronize ela com o repositório. (Uma vez que sua máquina provavelmente será solicitada para compilar pacotes para o `-current`, o `kernel` que ela executa deve estar razoavelmente atualizado com o `bindist` que será exportado por nossos `scripts`.)
2. Se você está usando `pxeboot`: crie um diretório para armazenar os arquivos de instalação. Você provavelmente vai querer usar um subdiretório do `/pxeroot`, por exemplo, `/pxeroot/${arch}-${branch}`. Exporte como `DESTDIR`.
3. Se você está realizando uma compilação para outra plataforma, que não a instalada na máquina (*cross-building*), exporte `TARGET_ARCH=${arch}`.

   Nota: O procedimento para compilação cruzada de `ports` ainda não está definido.

4. Gere um arquivo de configuração para o `kernel`. Inclua o `GENERIC` (ou, se você está usando mais que 3.5G de memória em um i386, o `PAE`).

   Opção requeridas:

   ```
   options         NULLFS
   ```

```
options         TMPFS
```

Opções sugeridas:

```
options         GEOM_CONCAT
options         GEOM_STRIPE
options         SHMMAXPGS=65536
options         SEMMNI=40
options         SEMMNS=240
options         SEMUME=40
options         SEMMNU=120

options         ALT_BREAK_TO_DEBUGGER
```

Para o `PAE`, atualmente não é possível carregar módulos. Portanto, se você está executando uma arquitetura que suporta emulação binária do Linux, você precisará adicionar:

```
options         COMPAT_LINUX
options         LINPROCFS
```

Também para o `PAE`, a partir de 12/09/2011 você precisa do seguinte. Isso precisa ser investigado:

```
nooption        NFSD                    # New Network Filesystem Server
options         NFSCLIENT               # Network Filesystem Client
options         NFSSERVER               # Network Filesystem Server
```

5. Como `root`, execute os passos usuais de compilação, por exemplo:

   ```
   make -j4 buildworld
   make buildkernel KERNCONF=${kernconf}
   make installkernel KERNCONF=${kernconf}
   make installworld
   ```

   Os passos de instalação usam o caminho especificado na da váriavel `DESTDIR`.

6. Personalize os arquivos em `etc/`. O local no qual você fará isso, se no próprio cliente ou em outra máquina, vai depender se você está usando ou não o `pxeboot`.

   Se você está usando `pxeboot`: crie um subdiretório no *`${DESTDIR}`* chamado `conf/`. Crie um subdiretório `default/etc/`, e (se seu `site` vai hospedar vários nós), subdiretórios *`${ip-address}`*`/etc/` para os arquivos que vão sobrescrever as configurações para os `hosts` individuais. (Você pode achar útil criar um `link` simbólico de cada um destes diretórios para um `hostname`.) Copie todo o conteúdo do *`${DESTDIR}`*`/etc/` para `default/etc/`; que é onde você irá editar seus arquivos. Nos diretórios criados para cada endereço IP, você provavelmente só irá necessitar personalizar os arquivos `rc.conf`.

   Em ambos os casos, execute os seguintes passos:

   - Crie um usuário e grupo `ports-`*`${arch}`*. Adicione o usuário ao grupo `wheel`. Ele pode ter um `'*'` no lugar da senha.

     Crie o `/home/ports-`*`${arch}/.ssh/`* e popule o arquivo `authorized_keys` com as chaves ssh apropriadas.

   - Crie os usuários:

     ```
     squid:*:100:100::0:0:User &:/usr/local/squid:/bin/sh
     ganglia:*:102:102::0:0:User &:/usr/local/ganglia:/bin/sh
     ```

     E também os adicione ao arquivo `etc/group`.

   - Crie os arquivos apropriados em `etc/.ssh/`.

- Edite o `etc/crontab` e adicione o seguinte:

```
*       *       *       *       *       root    /var/portbuild/scripts/client-metrics
```

- Crie um `etc/fstab` apropriado. (Se você tem várias máquinas diferentes, você precisará colocar este arquivo nos diretórios específicos de cada uma.)
- Edite o `etc/inetd.conf` e adicione o seguinte:

```
infoseek        stream  tcp     nowait  nobody  /var/portbuild/scripts/reportload
```

- Nós utilizamos o timezone UTC no `cluster`:

```
cp /usr/share/zoneinfo/Etc/UTC etc/localtime
```

- Crie um `etc/rc.conf` apropriado. (Se você está usando `pxeboot`, e tem várias máquinas diferentes, você precisará colocar este arquivo nos diretórios específico de cada uma.)

  Configurações recomendadas para nós físicos:

```
hostname="${hostname}"
inetd_enable="YES"
linux_enable="YES"
nfs_client_enable="YES"
ntpd_enable="YES"
ntpdate_enable="YES"
ntpdate_flags="north-america.pool.ntp.org"
sendmail_enable="NONE"
sshd_enable="YES"
sshd_program="/usr/local/sbin/sshd"

gmond_enable="YES"
squid_enable="YES"
squid_chdir="/usr2/squid/logs"
squid_pidfile="/usr2/squid/logs/squid.pid"
```

  Configurações obrigatórias para nós baseados no VMWare:

```
vmware_guest_vmmemctl_enable="YES"
vmware_guest_guestd_enable="YES"
```

  Configurações recomendadas para nós baseados no VMWare:

```
hostname=""
ifconfig_em0="DHCP"
fsck_y_enable="YES"

inetd_enable="YES"
linux_enable="YES"
nfs_client_enable="YES"
sendmail_enable="NONE"
sshd_enable="YES"
sshd_program="/usr/local/sbin/sshd"

gmond_enable="YES"
squid_enable="YES"
squid_chdir="/usr2/squid/logs"
squid_pidfile="/usr2/squid/logs/squid.pid"
```

  O ntpd(8) *não* deve ser habilitado para os nós baseados no VMWare.

Além disso, você pode optar por deixar o **squid** desabilitado por padrão, de modo a não ter um */usr2* persistente (o que deve economizar tempo na criação da instância.) O trabalho ainda está em andamento.

- Crie o `etc/resolv.conf`, se necessário.
- Modifique o `etc/sysctl.conf`:

```
9a10,30
> kern.corefile=/usr2/%N.core
> kern.sugid_coredump=1
> #debug.witness_ddb=0
> #debug.witness_watch=0
>
> # squid needs a lot of fds (leak?)
> kern.maxfiles=40000
> kern.maxfilesperproc=30000
>
> # Since the NFS root is static we don't need to check frequently for file changes
> # This saves >75% of NFS traffic
> vfs.nfs.access_cache_timeout=300
> debug.debugger_on_panic=1
>
> # For jailing
> security.jail.sysvipc_allowed=1
> security.jail.allow_raw_sockets=1
> security.jail.chflags_allowed=1
> security.jail.enforce_statfs=1
>
> vfs.lookup_shared=1
```

- Se desejar, modifique o `etc/syslog.conf` para mudar o destino dos `logs` para `@pointyhat.freebsd.org`.

13.4. Configurando os `ports`

1. Instale os seguintes `ports`:

```
net/rsync
security/openssh-portable (with HPN on)
security/sudo
sysutils/ganglia-monitor-core (with GMETAD off)
www/squid (with SQUID_AUFS on)
```

   Existe um trabalho em andamento para criar um `meta-port`, mas ainda não está completo.

2. Customize os arquivos em `usr/local/etc/`. O local no qual você fará isso, se no próprio cliente ou em outra máquina, vai depender se você está usando ou não o `pxeboot`.

> **Nota:** O truque de usar subdiretórios`conf` para sobreescrever as opções padrões é menos eficaz aqui, pois você precisa copiar todos os subdiretórios do `usr/`. Este é um detalhe da implementação de como o `pxeboot` funciona.

Execute os seguintes passos:

- Modifique o `usr/local/etc/gmond.conf`:

```
21,22c21,22
<   name = "unspecified"
<   owner = "unspecified"
---
>   name = "${arch} package build cluster"
>   owner = "portmgr@FreeBSD.org"
24c24
<   url = "unspecified"
---
>   url = "http://pointyhat.freebsd.org"
```

  Se existirem máquinas de mais de um `cluster` no mesmo domínio `multicast` (basicamente = LAN), então altere os grupos de `multicast` para valores diferentes (.71, .72, etc).

- Crie o `usr/local/etc/rc.d/portbuild.sh`, usando um valor apropriado para `scratchdir`:

```
#!/bin/sh
#
# Configure a package build system post-boot

scratchdir=/usr2

ln -sf ${scratchdir}/portbuild /var/

# Identify builds ready for use
cd /var/portbuild/${arch}
for i in */builds/*; do
    if [ -f ${i}/.ready ]; then
        mkdir /tmp/.setup-${i##*/}
    fi
done

# Flag that we are ready to accept jobs
touch /tmp/.boot_finished
```

- Modifique o `usr/local/etc/squid/squid.conf`:

```
288,290c288,290
< #auth_param basic children 5
< #auth_param basic realm Squid proxy-caching web server
< #auth_param basic credentialsttl 2 hours
---
> auth_param basic children 5
> auth_param basic realm Squid proxy-caching web server
> auth_param basic credentialsttl 2 hours
611a612
> acl localnet src 127.0.0.0/255.0.0.0
655a657
> http_access allow localnet
2007a2011
> maximum_object_size 400 MB
2828a2838
```

```
> negative_ttl 0 minutes
```

Modifique também o `usr/local` para *usr2* em `cache_dir`, `access_log`, `cache_log`, `cache_store_log`, `pid_filename`, `netdb_filename`, `coredump_dir`.

E finalmente, mude o esquema de armazenamento do `cache_dir`, de `ufs` para `aufs` (o qual oferece uma melhor performance).

- Configure o `ssh`: copie os arquivos do `/etc/ssh` para `/usr/local/etc/ssh` e adicione `NoneEnabled yes` ao `sshd_config`.
- Modifique o `usr/local/etc/sudoers`:

```
38a39,42
>
> # local changes for package building
> %wheel        ALL=(ALL) ALL
> ports-${arch}    ALL=(ALL) NOPASSWD: ALL
```

13.5. Configuração no próprio cliente

1. Entre no diretório `port/package` que você escolheu acima, por exemplo, `cd /usr2`.
2. Execute como root:

```
mkdir portbuild
chown ports-${arch}:ports-${arch} portbuild
mkdir pkgbuild
chown ports-${arch}:ports-${arch} pkgbuild
mkdir squid
mkdir squid/cache
mkdir squid/logs
chown -R squid:squid squid
```

3. Se os clientes preservam o conteúdo do `/var/portbuild` entre as suas inicializações, então eles também deverão preservar o `/tmp` ou então revalidar as compilações disponíveis no momento do boot (veja o `script` nas máquinas `amd64`). Eles também devem limpar os `chroots` obsoletos das compilações anteriores antes de criar o `/tmp/.boot_finished`.
4. Inicie o cliente.
5. Como root, crie a estrutura de diretórios do `squid`:

```
squid -z
```

13.6. Configuração no `pointyhat`

Estes passos precisam ser feitos por um `portmgr`, autenticado como o usuário `ports-${arch}`, no `pointyhat`.

1. Se alguma das portas TCP padrão não estiver disponível (veja acima), você precisará criar um túnel `ssh` para ela e deverá incluí-lo no `crontab`.

2. Adicione uma entrada em `/home/ports-${arch}/.ssh/config` para especificar o endereço IP público, a porta TCP para o `ssh`, o usuário, e qualquer outra informação necessária.
3. Crie o `/var/portbuild/${arch}/clients/bindist-${hostname}.tar`.
   - Copie um arquivos dos existentes para usar como modelo e descompacte-o em um diretório temporário.
   - Personalize o `etc/resolv.conf` para o `site` local.
   - Personalize o `etc/make.conf` para a busca de arquivo no FTP local. Nota: a anulação da variável `MASTER_SITE_BACKUP` deve ser comum para todos os nós, mas a primeira entrada em `MASTER_SITE_OVERRIDE` deve ser o espelho FTP mais próximo. Por exemplo:

     ```
     .if defined(FETCH_ORIGINAL)
     MASTER_SITE_BACKUP=
     .else
     MASTER_SITE_OVERRIDE= \
      ftp://friendly-local-ftp-mirror/pub/FreeBSD/ports/distfiles/${DIST_SUBDIR}/ \
      ftp://${BACKUP_FTP_SITE}/pub/FreeBSD/distfiles/${DIST_SUBDIR}/
     .endif
     ```

   - Empacote-o com `tar` e mova para o local correto.

   Dica: você precisará de um destes para cada máquina; no entanto, se você tem várias máquinas no mesmo site, você deve criar um local específico para este `site` (por exemplo, em `/var/portbuild/conf/clients/`) e criar um `link` simbólico para ele.
4. Crie o `/var/portbuild/${arch}/portbuild-${hostname}` utilizando um dos existentes como guia. O conteúdo deste arquivo sobrescreve as configurações de `/var/portbuild/${arch}/portbuild.conf`.

   Sugestão de valores:

   ```
   disconnected=1
   http_proxy="http://localhost:3128/"
   squid_dir=/usr2/squid
   scratchdir=/usr2/pkgbuild
   client_user=ports-${arch}
   sudo_cmd="sudo -H"
   rsync_gzip=-z

   infoseek_host=localhost
   infoseek_port=${tunelled-tcp-port}
   ```

   Outros valores possíveis:

   ```
   use_md_swap=1
   md_size=9g
   use_zfs=1
   scp_cmd="/usr/local/bin/scp"
   ssh_cmd="/usr/local/bin/ssh"
   ```

Os passos abaixo precisam ser executados por um `portmgr` autenticado como `root` no `pointyhat`.

1. Adicione o endereço IP público em `/etc/hosts.allow`. (Lembre-se, várias máquinas podem estar sob o mesmo endereço IP.)
2. Adicione uma entrada `data_source` para `/usr/local/etc/gmetad.conf`:

   ```
   data_source "arch/location Package Build Cluster" 30 hostname
   ```

Você precisará reiniciar o `gmetad`.

13.7. Habilitando o nó

Estes passos precisam ser executados por um `portmgr` autenticado como `ports-`*`arch`* no `pointyhat`.

1. Certifique-se que o `ssh` está funcionando executando `ssh` *`hostname`*.
2. Crie os arquivos em `/var/portbuild/scripts/` executando algo como `/var/portbuild/scripts/dosetupnode` *`arch major`* `latest` *`hostname`*. Verifique se os arquivos foram criados no diretório.
3. Teste as outras portas TCP executando `telnet` *`hostname portnumber`*. A porta `414` (ou seu túnel) deve dar-lhe algumas linhas com informações de status, incluindo `arch` e `osversion`; A porta `8649` deve retornar um `XML` do `ganglia`.

Esses passos precisam ser executados por um `portmgr` autenticado como `root` no `pointyhat`.

1. Informe o `qmanager` sobre o nó. Por exemplo:

   ```
   python path/qmanager/qclient add name=uniquename arch=arch osversion=osversion
   numcpus=number haszfs=0 online=1 domain=domain primarypool=package pools="package
   all" maxjobs=1 acl="ports-arch,deny_all"
   ```

14. Como configurar um novo `branch` do FreeBSD

Quando um novo `branch` é criado, é necessário efetuar alguns ajustes no sistema para especificar que o `branch` anterior não mais corresponde ao `HEAD`. As seguintes instruções se aplicam ao número do `branch` *anterior*:

- (novo código base) Edite o `/var/portbuild/conf/server.conf` e faça as seguintes alterações:
  - Adicione o *`new-branch`* na variável `SRC_BRANCHES`.
  - Para o `branch` que anteriormente era o `head`, mude o `SRC_BRANCH_`*`branch`*`_TAG` para `RELENG_`*`branch`*`_0`.
  - Adicione `SRC_BRANCH_`*`new-branch`*`_TAG=.` (o ponto é literal).

- (novo código base) Execute o `/var/portbuild/updatesnap` manualmente.
- (Apenas para o antigo código base) Crie um novo sistema de arquivos **zfs** para os fontes:

  ```
  zfs create a/snap/src-branch
  ```

- (Necessário apenas para o antigo código base): Obtenha uma cópia da árvore de fontes do `src` apartir do SVN e deposite a mesma no novo sistema de arquivos:

  ```
  cvs -Rq -d /r/ncvs co -d src-branch-r RELENG_branch
  ```

- (Necessário apenas para o antigo código base): Edite a cópia principal do `Tools/portbuild/portbuild.conf`.

- (Necessário apenas para o antigo código base): Edite a cópia do arquivo acima para cada uma das arquiteturas em `/var/portbuild/`*`arch`*`/portbuild.conf`.
- (Necessário apenas para o antigo código base): Edite o `/var/portbuild/scripts/buildenv`.
- (Necessário apenas para o antigo código base): Adicione um `link` simbólico de `/var/portbuild/scripts/dopackages` para `/var/portbuild/scripts/dopackages.`*`branch`*.
- (Necessário apenas para o antigo código base): Modifique as variáveis `HEAD_BRANCH` e `NON_HEAD_BRANCHES` no arquivo `/var/portbuild/scripts/updatesnap`.
- (Necessário apenas para o antigo código base): Adicione o diretório `snap` ao arquivo `/var/portbuild/scripts/zexpire`.
- (Necessário apenas para o antigo código base): Crie os `links` simbólicos para uso do servidor web no diretório `/var/portbuild/errorlogs/`:

  ```
  ln -s ../arch/branch/builds/latest/bak/errors arch-branch-full
  ln -s ../arch/branch/builds/latest/bak/logs arch-branch-full-logs
  ln -s ../arch/branch/builds/latest/errors arch-branch-latest
  ln -s ../arch/branch/builds/latest/logs arch-branch-latest-logs
  ln -s ../arch/branch/builds/latest/bak/packages arch-branch-packages-full
  ln -s ../arch/branch/builds/latest/packages arch-branch-packages-latest
  ```

- Inicie a compilação para o `branch` executando:

  ```
  build create arch branch
  ```

- Crie o `bindist.tar`.

15. Como excluir um `branch` que deixou de ser suportado pelo FreeBSD

Quando um `branch` antigo deixa de ser suportado, existem algumas coisas a serem feitas para que não fique sujeira para trás.

- (novo código base) Edite o `/var/portbuild/conf/server.conf` e faça as seguintes alterações:
  - Apague o *`old-branch`* da variável `SRC_BRANCHES`.
  - Remova o `SRC_BRANCH_`*`old-branch`*`_TAG=`*`whatever`*

- (novo e antigo código base): `umount a/snap/src-`*`old-branch`*`/src; umount a/snap/src-`*`old-branch`*`; zfs destroy -r a/snap/src-`*`old-branch`*

- (novo e antigo código base) Provavelmente você encontrará os seguintes arquivos e `links` simbólicos em `/var/portbuild/errorlogs/` os quais podem ser removidos:
  - Arquivos chamados `*-`*`old_branch`*`-failure.html`
  - Arquivos chamados `buildlogs_*-`*`old_branch`*`-*-logs.txt`
  - `Links` simbólicos chamados `*-`*`old_branch`*`-previous*`
  - `Links` simbólicos chamados `*-`*`old_branch`*`-latest*`

16. Como regerar pacotes baseados em outra versão menor do FreeBSD

Desde 2011 a filosofia da compilação de pacotes diz que devemos compilá-los baseados *na versão mais antiga suportada* de cada `branch`. Por exemplo: se no `RELENG-8` as seguintes versões são suportadas: 8.1, 8.2, 8.3; então o `packages-8-stable` deve ser compilado a partir da versão 8.1.

Quando uma versão chega ao fim de sua vida (`End-Of-Life`, veja o quadro (http://www.freebsd.org/security/index.html#supported-branches)), uma compilação completa (não incremental!) dos pacotes deve ser realizada e enviada para os servidores de distribuição.

Os procedimentos para o novo código base são os que seguem abaixo:

- Edite o `/var/portbuild/conf/server.conf` e faça as seguintes mudanças:
  - Altere o `SRC_BRANCH_`*`branch`*`_TAG` para `RELENG_`*`branch_N`* no qual o `N` é versão menor mais antiga para este ramo.

- Execute o `/var/portbuild/updatesnap` manualmente.
- Execute o `dopackages` com a opção `-nobuild`.
- Siga os procedimentos de configuração.
- Agora você já pode executar o `dopackages` sem a opção `-nobuild`.

O procedimento para o antigo código base fica como um exercício para o leitor.

17. Como configurar uma nova arquitetura

> **Nota:** Os passos iniciais precisam ser feitos usando **sudo**.

- Crie um novo usuário e grupo `ports-`*`arch`*.
- `mkdir /var/portbuild/`*`arch`*
- Crie um novo sistema de arquivo **zfs**:

  `zfs create -o mountpoint=/a/portbuild/`*`arch`* `a/portbuild/`*`arch`*

- `chown ports-`*`arch`*`:portmgr /var/portbuild/`*`arch`*`;`
`chmod 755 /var/portbuild/`*`arch`*`;`
`cd /var/portbuild/`*`arch`*
- Crie e popule o diretório `.ssh`.
- Crie um diretório para os `logs` de compilação e para os `logs` de erros:

```
mkdir /dumpster/pointyhat/arch/archive
```

> **Nota:** É possível que `/dumpster/pointyhat` não tenha mais espaço suficiente. Neste caso, crie o diretório dos arquivos como `/dumpster/pointyhat/arch/archive` e crie um `link` simbólico para ele. (Isso precisa ser resolvido.)

- Crie um `link` para o diretório acima para o servidor web:

  ```
  ln -s /dumpster/pointyhat/arch/archive archive
  ```

  > **Nota:** Os próximos passos são mais fáceis de serem realizados como o usuário `ports-arch`.

- No diretório `/var/portbuild/arch` execute:

  ```
  mkdir clients
  ```

- Popule o diretório `clients` como de costume.
- `mkdir loads`
- `mkdir lockfiles`
- Crie um `make.conf` local. Nos casos mais comuns você pode executar:

  ```
  ln ../make.conf ./make.conf
  ```

- Crie um arquivo vazio `mlist`.
- (Necessário apenas para o antigo código base) Crie o `pnohang.arch`. (O modo mais fácil é fazer isso em um cliente, e depois copiar o arquivo de volta):

  ```
  cc pnohang.c -o pnohang-arch
  ```

- Crie um novo arquivo `portbuild.conf` a partir de um existente para uma outra arquitetura.
- Crie os arquivos `portbuild.machinename.conf` personalizando-os de forma adequada.
-

  ```
  cd .ssh && ssh-keygen
  ```

- Edite o arquivo `.ssh/config` para tornar mais conveniênte o uso do **ssh**.
- Crie o diretório de configuração privada:

  ```
  mkdir /var/portbuild/conf/arch
  ```

- Crie os `scripts` `dotunnel.*` que forem necessários dentro do diretorio acima.

  > **Nota:** Mais uma vez usando **sudo**:

- Informe o **qmanager** sobre a arquitetura:

```
python path/qmanager/qclient add_acl name=ports-arch uidlist=ports-arch gidlist=portmgr sense=1
```

- (Necessário apenas para o novo código base): Adicione a *arch* na variável SUPPORTED_ARCHS do arquivo /var/portbuild/*arch*/server.conf.
- (Necessário apenas para o antigo código base): Edite o /var/portbuild/scripts/buildenv.
- Adicione o diretório *arch* no /var/portbuild/scripts/zbackup e no /var/portbuild/scripts/zexpire.
- Necessário apenas para o antigo código base): Como no procedimento para criação de um novo branch: crie os links para o servidor web no diretório /var/portbuild/errorlogs/:

```
ln -s ../arch/branch/builds/latest/bak/errors arch-branch-full
ln -s ../arch/branch/builds/latest/bak/logs arch-branch-full-logs
ln -s ../arch/branch/builds/latest/errors arch-branch-latest
ln -s ../arch/branch/builds/latest/logs arch-branch-latest-logs
ln -s ../arch/branch/builds/latest/bak/packages arch-branch-packages-full
ln -s ../arch/branch/builds/latest/packages arch-branch-packages-latest
```

- Crie mais dois links simbólicos para o servidor web dentro do diretório /var/portbuild/errorlogs/:

```
ln -s ../arch/archive/buildlogs arch-buildlogs
ln -s ../arch/archive/errorlogs arch-errorlogs
```

Nota: Novamente como ports-*arch*:

- Para cada branch que será suportado, faça o seguinte:
  - Inicie a compilação para o branch com:

    ```
    build create arch branch
    ```

  - Crie o bindist.tar.

Nota: Uma última vez usando o **sudo**:

- (Necessário apenas para o antigo código base): Só depois que a primeira execução do **dopackages** for feita para a arquitetura: adicione a arquitetura ao /var/portbuild/scripts/dopackagestats.
- Adicione uma entrada *arch* apropriada para o /var/portbuild/scripts/dologs no crontab do usuário root. (Esta é uma solução paliativa)

18. Como configurar um novo nó principal (instância do pointyhat)

Esta seção está em progresso.

Por favor, consulte o Mark Linimon antes de efetuar qualquer mudança.

18.1. Instalação básica

1. Instale o FreeBSD.
2. Para cada arquitetura suportada, adicione um usuário e grupo `ports-${arch}`. Adicione os usuários ao grupo `wheel`. Eles devem ter um `'*'` como senha. Crie também, de modo similar, o usuário `ports` e `portmgr`.
3. Para cada arquitetura suportada, crie o `/home/ports-${arch}/.ssh/` e popule o `authorized_keys`.
4. Crie os arquivos apropriados em `/etc/.ssh/`.
5. Adicione a seguinte linha ao arquivo `/boot/loader.conf`:

   ```
   console="vidconsole,comconsole"
   ```

6. Adicione as seguintes linhas ao arquivo `/etc/sysctl.conf`:

   ```
   kern.maxfiles=40000
   kern.maxfilesperproc=38000
   ```

7. Certifique-se de que as seguintes mudanças foram realizadas no `/etc/ttys`:

   ```
   ttyu0   "/usr/libexec/getty std.9600"   vt100   on secure
   ```

8. Ainda a ser definido.

18.2. Configurando o disco

1. Crie um volume **zfs** chamado a e monte-o em `/a`:

   ```
   # zpool create a mirror da1 da2 mirror da3 da4 mirror da5 da6 mirror da7 da8
   ```

2. Configure o diretório base do `portbuild`:

   ```
   # mkdir -p /a/portbuild
   # cd /a/portbuild
   # chown portmgr:portmgr .
   # chmod 775 .
   ```

3. Ainda a ser definido.

18.3. Configurando o `src`

1. Ainda a ser definido.

18.4. Configurando o `ports`

1. Os seguintes `ports` (ou seus sucessores mais recentes) são obrigatórios:

   ```
   databases/py-pysqlite23
   databases/py-sqlalchemy
   devel/git (WITH_SVN)
   devel/py-configobj
   ```

```
devel/py-setuptools
devel/subversion
net/nc
net/rsync
sysutils/ganglia-monitor-core (with GMETAD off)
sysutils/ganglia-webfrontend (WITHOUT_X11)
www/apache22 (with EXT_FILTER and THREADS)
```

Os ports acima também irão instalar:

```
databases/sqlite3
lang/perl-5.12
lang/python27
```

Os seguintes `ports` (ou seus sucessores mais recentes) são fortemente recomendados:

```
benchmarks/bonnie++
devel/ccache
mail/postfix
net/isc-dhcp41-server
ports-mgmt/pkg_cutleaves
ports-mgmt/pkg_tree
ports-mgmt/portaudit
ports-mgmt/portmaster
security/sudo
shells/bash
shells/zsh
sysutils/screen
sysutils/smartmontools
```

2. Configure o e-mail fazendo o seguinte: (ainda a ser definido).

18.5. Outros

1. Ainda a ser definido.

19. Procedimentos para lidar com falhas de disco

Quando uma máquina tem uma falha de disco (por exemplo, um `panic` devido a erros de leitura, etc.), devemos executar os seguintes procedimentos:

- Anote o tempo e o tipo de falha (por exemplo, colea saída do console que for relevante) no `/var/portbuild/${arch}/reboots`
- Para os clientes gohan i386, limpe o disco criando o arquivo `/SCRUB` no `nfsroot` (por exemplo, `/a/nfs/8.dir1/SCRUB`) e reinicie. Isso vai executar um `dd if=/dev/zero of=/dev/ad0` e forçar a unidade a remapear todos os setores defeituosos que encontrar, isto se ela ainda tiver setores suficientes sobrando. Esta é uma medida temporária para estender o tempo de vida de uma unidade de disco que em breve irá tornar-se inutilizável.

> **Nota:** Para os sistemas `blade` i386, outro sinal de falha nos discos é quando a `blade` fica em espera e não responde a qualquer comando pelo console, ou mesmo pelo NMI.

Para os outros sistemas de compilação que não executam um newfs nos seus discos no momento da inicialização (por exemplo, os sistemas amd64) este procedimento deve ser ignorado.

- Se o problema persistir, então provavelmente o disco está inutilizado. Remova a máquina do `mlist` e (para discos ATA) execute o `smartctl` na unidade:

  ```
  smartctl -t long /dev/ad0
  ```

  Isso vai levar cerca de 30 minutos:

  ```
  gohan51# smartctl -t long /dev/ad0
  smartctl version 5.38 [i386-portbld-freebsd8.0] Copyright (C) 2002-8
  Bruce Allen
  Home page is http://smartmontools.sourceforge.net/

  === START OF OFFLINE IMMEDIATE AND SELF-TEST SECTION ===
  Sending command: "Execute SMART Extended self-test routine immediately in off-line mode".
  Drive command "Execute SMART Extended self-test routine immediately in off-line mode" successful
  Testing has begun.
  Please wait 31 minutes for test to complete.
  Test will complete after Fri Jul  4 03:59:56 2008

  Use smartctl -X to abort test.
  ```

  Quando o comando acima finalizar, execute o comando `smartctl -a /dev/ad0` para verificar o estado da unidade:

  ```
  # SMART Self-test log structure revision number 1
  # Num  Test_Description    Status                  Remaining
  LifeTime(hours)  LBA_of_first_error
  #   1  Extended offline    Completed: read failure       80%     15252    319286
  ```

  Ele também exibirá outros dados, incluindo um `log` dos erros anteriores da unidade. É possível que a unidade mostre erros de DMA embora não apresente falhas no auto-teste (por conta do remapeamento de setores).

Quando um disco falhar, por favor, informe os administradores do `cluster`, para que possamos substituí-lo.